בסיסי נתונים טבלאיים

ושפת SQL

עקרונות ועיצוב

עורך ראשי: **יצחק עמיהוד**

עריכה לשונית ועיצוב: **שרה עמיהוד, ענבל אילני**

הגהה: **יואל ארווץ**

עיצוב עטיפה: **שרון רז**

תוכנת התרגול **FirstSQL** נכתבה על ידי **בועז אביאני**

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

- **טלפון: 09-9564716**
- **פקס: 09-9571582**
- **דואר אלקטרוני: info@hod-ami.co.il**
- **אתר באינטרנט: www.hod-ami.co.il**

בסיסי נתונים טבלאיים

ושפת SQL

עקרונות ועיצוב

רז הייפרמן

הוצאת הוד-עמי

Relational Data Bases and SQL
Principles and Design

By **Raz Heiferman**

Editor: **I. Amihud**

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 6108 הרצליה 46160
טלפון : 09-9564716 פקס : 09-9571582

info@hod-ami.co.il

הודפס בישראל תש"ס, 2000

HOD-AMI Ltd.
P.O.B. 6108, Herzliya
ISRAEL, 2000

ISBN 965-361-219-0 מסת"ב

הקדמה

בשנים שחלפו מאז פרסמתי את הספר "**בסיסי נתונים טבלאיים ושפת SQL**" בשנת 1993, חלו מספר שינויים והתפתחויות שהשפיעו רבות על התפתחות טכנולוגיית בסיסי הנתונים בכלל, ועל טכנולוגיית בסיסי הנתונים הטבלאיים בפרט. השינויים הבולטים הם:

- **התפוצה הרחבה של המודל הטבלאי**: מאז הופעת התיאוריה המגדירה את המודל הטבלאי בתחילת שנות ה-70, נדרשו יותר מעשר שנים למחקר ולפיתוח החומרה והתוכנה שיכולה הייתה להוציא את התיאוריה מן הכוח אל הפועל. שנות ה-80 הבליטו את חשיבות המודל הטבלאי על חשבון המודלים האחרים, וביניהם המודל ההיררכי והמודל הרשתי. בשנות ה-90 אנו עדים לכך שרוב מערכות המידע החדשות מפותחות ופועלות סביב בסיס נתונים טבלאי – RDBMS (Relational DBMS). מערכות אלו הפכו לטכנולוגיית ניהול הנתונים המועדפת עבור רוב מערכות המידע המודרניות, עבור מערכות תומכות החלטה ומחסני נתונים (Data Warehouses) ועבור שרתי אינטרנט (Web Servers). מערכות ניהול בסיסי הנתונים הוותיקות יותר, כדוגמת IMS, IDMS, Adabas ואחרות נדחקו הצידה והשימוש בהן הולך ופוחת.
- **שפת SQL הפכה לשפת ניהול הנתונים התקנית הנפוצה ביותר**: שפת SQL הפכה לשפת הגישה וניהול בסיסי הנתונים הנפוצה ביותר. השפה התפתחה והשתכללה. בשנת 1993 רוב המערכות תמכו בעיקר בתקן SQL1, אך כיום רוב המערכות תומכות ברוב מרכיביו של תקן SQL2 שהתפרסם בשנת 1992. בנוסף, גובשה טיוטת תקן הקרויה SQL3, שיכיל מספר הרחבות משמעותיות בכל הקשור ליכולת המודל הטבלאי לתמוך באובייקטים ובטיפוסי נתונים מופשטים (Abstract Data Types). תקן זה עוד לא אושר, אולם רוב יצרני המערכות המסחריות החלו לאמץ חלק מהרעיונות המופיעים בטיוטה.
- **מעבר מסביבת מחשוב הומוגנית ומרכזית לסביבת מחשוב הטרוגנית מבוזרת:** סביבת המחשוב הארגונית מבוססת כיום על סביבות מחשוב מבוזרות והטרוגניות הפועלות במודל שרת/לקוח (Client/Server). מערכות אלו מבוססות על שילוב של מחשבים אישיים חזקים עם ממשק משתמש מבוסס Windows, על רשתות תקשורת מקומיות (LAN) מהירות ועל שרתים מרובי מעבדים עתירי עוצמה וזולים יחסית, המופעלים על ידי מערכות הפעלה Unix ו- Windows NT. מספר הארגונים המבססים את מודל המחשוב שלהם על מחשבים מרכזיים ומחשבי מיני המבוססים על מערכות הפעלה וותיקות יותר כגון MVS ו- VMS הולך ופוחת. חלה התפתחות עצומה בכל הקשור לתקנים פורמליים ותקנים בפועל, המאפשרים לארגון לנהל את הנתונים בצורה מבוזרת ולשתף נתונים המנוהלים בבסיסי נתונים שונים (Interoperability).
- **התרחבות יישומי מולטימדיה:** יישומים מודרניים מורכבים מבוססים על ממשקים גרפיים ומשלבים מולטימדיה (גרפיקה, טקסט, נתונים, קול, וידאו וכד'). יישומים אלה מפותחים בסביבות פיתוח מוכוונות אובייקטים (Object Oriented) ושפות תכנות כגון C++, Java או שפות דור רביעי (4GL) כגון Visual Basic, PowerBuilder,

Magic ו- Developer2000. שיטות פיתוח אלו הצריכו שיפור במערכות לניהול בסיסי נתונים המבוססות על המודל הטבלאי. חלק מהשיפורים מופיע כבר בטיוטת התקן SQL3.

- **התרחבות מהירה של האינטרנט (Internet) והאינטראנט (Intranet):** השימוש הנרחב באינטרנט ובאינטראנט להפצת מידע, למסחר אלקטרוני ולפיתוח יישומים ארגוניים הפך לאחת התופעות המדהימות של העשור האחרון. יותר ויותר יישומים ארגוניים מפותחים כיום באוסף הטכנולוגיות שנוצרו לטובת האינטרנט. יישומים אלה פועלים על בסיסי הנתונים הארגוניים והביאו לדרישה גוברת לשיפור הביצועים, להגדלת הזמינות (24 x 7 x 365) והתמיכה של מערכות לניהול בסיסי הנתונים ביישומים חדשים.
- **הופעת תפיסת מחסני נתונים (Data Warehouses):** בשנים האחרונות חלה התפתחות עצומה בתפיסת מחסן הנתונים, שמטרתה לבנות סביבת מחשוב ייעודית עבור אספקת מידע תומך החלטות. סביבת מחשוב זו היא סביבה נפרדת מהמערכות התפעוליות וברוב המקרים משתמשת בבסיס נתונים טבלאי ובאוסף הולך וגדל של כלי תחקור, כלי ניתוח נתונים וכלי כריית נתונים (Data Mining) המבוססים על שיטת ניהול נתונים זו.
- **שיפור ושכלול תהליכי עיצוב בסיס הנתונים:** במקביל להתפתחויות אלו נצבר ניסיון וידע רב בכל הקשור לשיטות עיצוב בסיס הנתונים. נוצרה אבחנה ברורה בין העיצוב התפישתי של בסיס הנתונים לבין המבנה הפיסי שלו. בעיצוב המודל אנו פוגשים מושגים הלקוחים ממודל האובייקטים: היררכיות סמנטיות, הורשה ועוד.

שינויים אלה בסביבת המחשוב המודרנית והתפישות המתקדמות באחסון ובניהול נתונים חייבו עדכון משמעותי של הספר כדי להבטיח את התאמתו לעולם המשתנה והצגת תמונה עדכנית של טכנולוגיית בסיסי הנתונים הטבלאיים. למעשה זהו ספר חדש לחלוטין, אשר מספר חלקים שבו מקבילים לספר הקודם.

מטרת הספר להציג בפני הקורא את עקרונות טכנולוגיית בסיסי הנתונים בכלל וטכנולוגיית בסיסי הנתונים הטבלאיים בפרט, ואת הצורה שבה עקרונות אלה יושמו במציאות באמצעות מערכות RDBMS מסחריות. הספר נועד כספר לימוד בתחום מערכות מידע או מדעי המחשב, לכל מי שעוסק בניתוח ופיתוח מערכות מידע ולכל מי שמשתמש במערכות מידע ומעוניין להכיר ולהבין את העקרונות של טכנולוגיית בסיסי הנתונים הטבלאיים. הספר מנסה לספק את הידע ולענות על האתגרים איתם מתמודד כיום מנתח המערכות ומהנדס התוכנה בסביבת המחשוב המודרנית.

הספר מניח שלקורא יש רקע מוקדם בנושא ארגון קבצים וניתוח מערכות מידע והוא בעל ניסיון בסיסי בשפת תכנות כלשהי. לשם רענון הידע, הקורא מופנה לספר "**ארגון נתונים וקבצים**", גם הוא מפרי עטו של המחבר, שפורסם בשנת 1995.

לא אוכל לסיים הקדמה זו ללא תודה מיוחדת לאשתי ושני ילדי, אשר גילו סבלנות אין קץ ועודדו אותי לאורך כל תקופת הכנת ספר זה.

ינואר 2000 **רז הייפרמן**

תוכן עניינים מקוצר

תוכן העניינים

על המחבר

רז הייפרמן, בעל תואר שני בחקר ביצועים מביה״ס למנהל עסקים באוניברסיטה העברית, סמנכ״ל טכנולוגיות מידע בחברת בזק. כיהן בעבר במספר תפקידי מפתח וביניהם מנהל חטיבת טכנולוגיות ופיתוח עסקים בחברת קונתהל, מנהל חטיבת תעשיה בחברת קונתהל, מנהל אגף מחשוב ובקרה בחברת אופטרוטק (אורבוטק כיום), מנהל פרויקטים בכיר בחברת טכם, מנהל תחום לוגיסטיקה וכ״א באגף מדעי הניהול ושירותי מידע במשרד התקשורת. מרצה בנושאי בסיסי נתונים וטכנולוגיות מידע במוסדות הוראה שונים וביניהם בפקולטה להנדסה באוניברסיטת תל אביב, ביה״ס למקצועות המחשב בצה״ל (ממר״ם לשעבר) ואחרים.

רז חיבר את הספרים **ארגון נתונים וניהול קבצים** (1982), **בסיסי נתונים – עקרונות, מודלים ויישומים** (1984), **בסיסי נתונים טבלאיים ושפת SQL** (1993), **ארגון נתונים וקבצים** (1995), **מחסני נתונים – עקרונות, ארכיטקטורה, עיצוב ויישום** (1999) וערך את הספר **טכנולוגיות שרת/לקוח** (1995), כולם ב**הוצאת הוד-עמי**.

חלק א'

מושגי יסוד בטכנולוגיית בסיסי הנתונים

חלק א' של הספר מורכב משני פרקים אלה:

- **פרק 1 – רקע ומושגי יסוד:** פרק זה מציג את מושגי היסוד והסביבה שבה פועלת מערכת לניהול בסיסי נתונים. הפרק מציג בצורה תמציתית את המושגים האלה: נתונים, מידע, מעגל המידע, מערכת מידע, תנועה עסקית, עיבוד נתונים, הצגת נתונים, יישום ותוכנית יישום. הפרק מציג את מערכת המידע כמושג מרכזי. זוהי מערכת העוסקת בקליטת התנועות העסקיות אשר משקפות את האירועים במציאות ובניהול הנתונים, בעיבוד הנתונים והצגתם לצורך קבלת החלטות. רכיב ניהול הנתונים מוצג כחלק מהמרכיבים העיקריים של כל מערכת מידע. הפרק מסביר את ההבדל בין סוגי מערכות שונות: מערכת מידע תפעולית התומכת במיגוון הפעילויות השוטפות של הארגון ומחסן הנתונים המוזן בנתונים מהמערכות התפעוליות ומספק את הנתונים במבנה ייעודי לתמיכה בתהליכי קבלת החלטות.

- **פרק 2 – עקרונות מערכות לניהול בסיסי נתונים:** פרק זה סוקר בקצרה את ההתפתחות שחלה במערכות ניהול נתונים מאז הופעת מערכות ניהול הקבצים, דרך הופעת טכנולוגיית בסיסי הנתונים ועד להפיכת מודל הנתונים הטבלאי ומערכות RDBMS המבוססות עליו, למערכות נפוצות. נסקור את טכנולוגיית המערכות לניהול קבצים ואת הבעיות העיקריות בטכנולוגיה זו. בעיות אלו הביאו לצורך בפיתוח טכנולוגיה חדשה, טכנולוגיית בסיסי הנתונים. הפרק מציג את ההגדרה של בסיס הנתונים, של המערכת לניהול בסיסי הנתונים ומשתמש בארכיטקטורת ANSI/SPARC כמצע לדיון במושגים המרכזיים של טכנולוגיית בסיסי הנתונים – סכימה, תת-סכימה וסכימה פנימית כמנגנוני הפשטה – שתפקידם להסתיר את מבנה הנתונים מהמשתמשים בו. הפרק סוקר את עקרון הפעולה של מערכת RDBMS, תוך אבחנה בין שלב הגדרת הנתונים לבין שלב השימוש בהם.

פרק 1

רקע ומושגי יסוד

1. מבוא
2. רקע
3. מערכת מידע
4. מערכות תפעוליות ומערכות מחסן נתונים
5. סיכום
6. שאלות חזרה ותרגילים

מבוא

במהלך שלושת העשורים האחרונים נמצא העולם בתהליך של מעבר מכלכלה תעשייתית לכלכלת מידע והמומחים צופים שבשנים הבאות **המידע** – יותר מאשר משאבים קלאסיים כגון הון, אדמה ומכונות – הוא שיניע את ההתקדמות הכלכלית ואת יצירת העושר והרווחה. בכלכלה מסוג זה, אחד הגורמים העיקריים להצלחה הוא **מה שאתה יודע** ופחות **מה ששייך לך.** עד כמה שזה נשמע מדהים, ארגונים למדו בדרך הקשה שהתחרות הקשה ביותר הניצבת לעומתם באה מאותם ארגונים שהיו מצוידים במידע טוב יותר על לקוחותיהם ועל השינויים בביקושים שלהם ואשר היה להם מידע על התהליכים בסביבה העסקית שבה הם פועלים. ארגונים אלה ידעו לרתום את המידע לצרכיהם והשתמשו בו כדי להתאים את פעילותם במהירות לשינויים בדרישות לקוחותיהם, והיו ערוכים לקבלת החלטות מהירות המבוססות על מידע זה. בסביבה העסקית המודרנית הארגונים מתחרים ביניהם על בסיס יכולתם להשיג, לפענח ולהשתמש במידע באופן יעיל. נתונים על לקוחות, מוצרים, הזמנות, ספקים, מלאים, פעילויות, מצב פיננסי, מתחרים ועוד, משמשים את הארגון לתפעול שוטף ולקבלת החלטות. שימוש יעיל בנתונים זמינים במועד ובמידת אמינות גבוהה הפכו לאחד מכלי הנשק העיקריים של הארגון הפועל בסביבה העסקית המודרנית.

יחד עם ההתפתחות המדהימה שחלה במשק ובכלכלה העולמית התפתחה גם **טכנולוגיית המידע** – **IT** (Information Technology). טכנולוגיה זו מאפשרת לארגונים לנהל ולנצל את המשאב החדש והחשוב – **המידע**. הנתונים המנוהלים במסגרת מערכות המחשוב הפכו לאחד הנכסים והמשאבים החשובים של הארגון. כמות הנתונים שהארגון מנהל הולכת וגדלה בקצב מדהים.

אחד המחקרים העדכניים הראה שכמות הנתונים המנוהלת במחשבים האישיים, בשרתים ובמחשבים מרכזיים מוכפלת כמעט מדי שנה. לדוגמה, חברת Capital One העוסקת במתן שירותים פיננסיים ושירותי אשראי, מנהלת כיום נתונים בהיקף של כ- 18 TB במחשבים המרכזיים שלה, ועוד כ- 8 TB במחשבים האישיים של עובדיה. רק במערכות המחשוב המרכזיות החברה אוגרת נתונים חדשים בקצב של 10 GB לשבוע. החברה מספקת כ- 3,000 צירופים שונים של תוכניות אשראי ותעריפים. כל אלה "תפורים" על פי מידת הסיכון והפרופיל הייחודי של הלקוח כפי שנגזרו מתוך מאגרי הנתונים של החברה. דוגמה ידועה אחרת היא חברת התעופה American Airlines שחלק גדול מהצלחתה נובע מיכולתה לנצל את טכנולוגיית המידע ולפתח את מערכת ההזמנות שלה, Sabre. החברה ניצלה את המערכת כדי להגיע לתפוסה טובה יותר של טיסותיה, לתכנון טוב יותר של יעדים ומועדים על פי דרישות לקוחותיה. היא קידמה ופיתחה רעיון מתקדם כמו "הנוסע המתמיד", המבוסס במידה רבה על היכולת לזהות ולצבור נתונים על כל טיסה של כל אחד מלקוחותיה.

ניהול כמויות ענקיות של נתונים מסוגים שונים ובתבניות שונות, הפך לאחד האתגרים העיקריים שכמעט כל ארגון מתמודד איתם. עד לפני שנים לא רבות, מערכות המחשוב היו נחלתם של הארגונים הגדולים והעשירים בלבד. כיום מערכות המחשוב הפכו לאחד מכלי העבודה העיקריים של כל הארגונים, גדולים כקטנים. יכולת הארגון להשתמש במערכות

המחשוב כדי להציע מיגוון חדשני של שירותים, כדי לנהל ולנצל את הנתונים ואת המידע שניתן להפיק מהם, הפך לאחד מסימני ההיכר של הכלכלה המודרנית. השימוש במערכות המידע הלך והתרחב והן חדרו כמעט לכל תחום בארגון. רשת האינטרנט, על דפי המידע שלה וקבצי המולטימדיה (כגון תמונות, קול, מפות ווידאו), הולכת ותופסת מקום מרכזי ביותר בתחומי העסקים. המסחר האלקטרוני וטכנולוגיית מחסני הנתונים הופכים גם הם לאמצעים חשובים ועיקריים עבור כל ארגון. כל אלה הביאו לדרישה גוברת והולכת לניהול נתונים במערכות ממוחשבות. כך, טכנולוגיית בסיסי הנתונים העוסקת בניהול נפחי נתונים גדולים בעלי מבנה מורכב ומקושרים ביניהם, הפכה לאחת מטכנולוגיות התשתית העיקריות שכמעט כל סוגי מערכות המחשוב נשענות עליה – מערכות מידע (Information Systems), מחסני נתונים (Data Warehouses), מערכות משובצות מחשב (Embedded Systems), מערכות תיב"מ (CAD/CAM), דואר אלקטרוני (E-Mail), מערכות מידע גיאוגרפיות (Geographical Information Systems), מסחר אלקטרוני (E-Commerce), מערכות עיבוד תמונה (Imaging), מערכות ניהול תזרים עבודה (Work Flow) וכד'.

בפרק זה נציג את מושגי היסוד שישמשו אותנו בפרקים הבאים של הספר. הצגת המושגים בפרק זה היא תמציתית מאוד ובמידה שיידרש, נחזור עליהם בהרחבה בהמשך.

בפרק זה

- נסביר מדוע נתונים מהווים את אחד הנכסים החשובים ביותר של כל ארגון מודרני ומהו מעגל המידע ההופך נתונים למידע, מידע להחלטות והחלטות לפעולות עסקיות.
- נגדיר מהי מערכת המידע, את רכיבי מערכת המידע – מרכיב קליטת התנועות, מרכיב עיבוד הנתונים, מרכיב הצגת הנתונים ומרכיב ניהול הנתונים ונסביר מהו יישום ומהי מערכת יישום.
- נציג את האבחנה בין מערכת מידע תפעולית המשמשת את הארגון לניהול פעילותו השוטפת לבין מחסן נתונים המשמש את הארגון לאספקת מידע לתהליכי קבלת החלטות.

רקע

נתחיל את סקירת הרקע בסיפור מעשה קצר, אשר מתאר בתמצית סביבה של מכללה – "מכללת ישראל ללימודים מתקדמים". התיאור המובא להלן פשטני ביותר והמציאות בה פועלת מכללה היא כמובן מורכבת פי כמה. סיפור מעשה זה ישמש אותנו במהלך ההסברים ובדוגמאות שונות שנציג בספר זה.

<u>סיפור המעשה – "מכללת ישראל" המכללה ללימודים מתקדמים</u>

"מכללת ישראל" היא מכללה חדשה המשמשת כמוסד הוראה ללימודים אקדמיים. במכללה יש מספר מחלקות אקדמיות הרשאיות להעניק תארים אקדמיים, כמו למשל המחלקה למנהל עסקים, המחלקה למשפטים, המחלקה לכלכלה ואחרות.

בכל מחלקה יש מספר מסלולי לימוד. לדוגמה, במחלקה למנהל עסקים יש מסלול למערכות מידע, לשיווק, לחשבונאות ולמימון. בראש כל מחלקה עומד ראש מחלקה. בכל מחלקה יש מספר אנשי סגל אקדמי העוסקים בהוראה ומספר אנשי סגל מינהלי העוסקים במיגוון פעילויות תומכות – מינהל, תחזוקה, אחזקת מעבדות, ביטחון וכד'.

כל מחלקה אקדמית רשאית להעניק תארים אקדמיים: תואר ראשון ותואר שני. כל מחלקה מציעה מספר קורסים בנושאים שונים הקשורים לתחום התמחותה. אנשי הסגל האקדמי מלמדים בקורסים השונים. קורסים יכולים להיות מסוג שיעור, סמינר או מעבדה. כל קורס מעניק מספר שונה של נקודות זכות לקראת קבלת תואר.

מדי שנה נרשמים למכללה סטודנטים המבקשים לקבל תואר אקדמי במחלקה כלשהי. כל סטודנט חייב ללמוד במספר קורסים, אליהם הוא נרשם בתחילת כל שנה. סטודנט נבחן בכל קורס פעמיים: מבחן סמסטר ומבחן סופי. כדי לקבל תואר אקדמי על הסטודנט לצבור מספר מסוים של נקודות זכות המוגדר על ידי כל מחלקה עבור כל חוג. לדוגמה, כדי לקבל תואר במערכות מידע במסגרת המחלקה למנהל עסקים, יש לצבור 35 נקודות זכות ולקבל ציון ממוצע של לפחות 75 בכל הקורסים.

עד כאן סיפור המעשה. גם מתיאור פשטני זה ניתן לראות שהמציאות שבה פועלת המכללה עתירה באובייקטים שונים: מחלקות, מרצים, סטודנטים, קורסים, כיתות, בניינים ועוד. היא גם עתירה באירועים שונים: קבלת מרצה חדש למחלקה, רישום סטודנט לקורס, פתיחת מחלקה חדשה, הענקת תואר אקדמי לסטודנט ועוד. כדי לתפקד ולספק את השירות שלשמו המכללה קיימת – הוראה והענקת תארים אקדמיים לסטודנטים – עובדי המכללה צריכים לקבל כל הזמן החלטות.

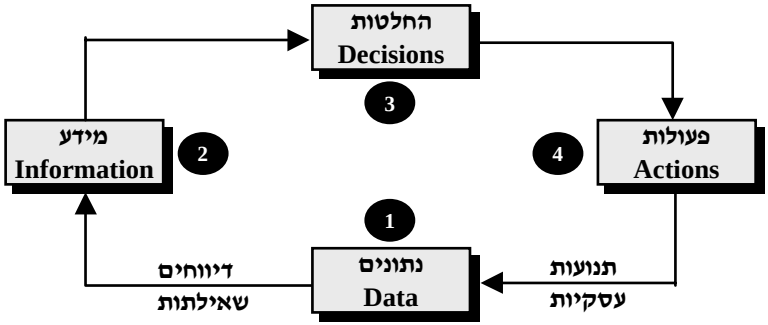
קבלת החלטות הינה אחת הפעולות הבסיסיות שכל פרט וכל ארגון מבצעים באופן שוטף. חלק מההחלטות פשוטות יחסית, כמו למשל: האם לפתוח קבוצת לימוד נוספת בקורס מסוים בסמסטר הבא; וחלק מההחלטות מורכבות יותר, כמו למשל: האם להשקיע סכום כסף גדול בהרחבת מערך הכיתות והמעבדות במחלקה מסוימת כדי להענות לביקוש עתידי. בלי קשר לרמת מורכבות ההחלטות, לכולן יש מכנה משותף אחד: הן צריכות להתבסס על **מידע** רלוונטי כלשהו. לדוגמה, כדי לקבל את ההחלטה האם לפתוח קבוצת לימוד חדשה בקורס מסוים, צריך לדעת מה מספר הסטודנטים הצפוי. הרי לא סביר

שנפתח קבוצה חדשה רק עבור שני סטודנטים, למשל. עלינו גם לבדוק את זמינות המרצים הדרושים לקבוצה החדשה, והאם יש כיתה פנויה וכד'.

כדי שנתונים יוכלו לשרת את תהליך קבלת ההחלטות, על הארגון לנהל אותם לאורך זמן, להבטיח את אמינותם ועדכניותם ולהבטיח את זמינותם בעת הצורך. **מערכת המידע** הממוחשבת היא האמצעי המשמש את הארגון לניהול הנתונים ולאבטחת זמינותם בעת הצורך. כל מערכת מידע מהווה **מודל** מסוים של המציאות. מודל זה מנוהל בצורה ממוחשבת ומאפשר לארגון לבצע את משימותיו השוטפות, להבין ולנתח את מצבו ולקבל החלטות על פי הצורך. מערכת המידע מנהלת נתונים על המציאות ומעדכנת אותם על ידי **תנועות עסקיות** על פי האירועים הקורים במציאות.

התכלית של מערכת מידע ממוחשבת היא לאגור ולנהל **נתונים** ולסייע בתהליך הפיכתם ל**מידע** הנדרש לתמיכה בהחלטות. לצורך הדיון, **נתונים** יוגדרו כעובדות גולמיות ואילו **המידע** יוגדר כעיבוד או אינטרפרטציה מסוימת (פיענוח והבנה) של הנתונים לצורך קבלת החלטות. להלן מספר דוגמאות לנתונים: מספר הסטודנטים שכבר נרשמו לקבוצת לימוד מסוימת, מספר הסטודנטים שנרשמו ללימודים במחלקה מסוימת, מספר המרצים במחלקה וההתמחות שלהם, מספר הכיתות והמעבדות השייכות למחלקה מסוימת וכד'. נתונים אלה ואחרים דרושים לקבלת החלטה על פתיחת קבוצת לימוד חדשה. מכיון שהנתונים הם הבסיס לתהליך קבלת ההחלטות, וטיב הנתונים העומדים לרשות הארגון הינו אחד המרכיבים העיקריים בטיב ההחלטות שהארגון מקבל, מקובל **להתייחס אל הנתונים כאל אחד המשאבים העיקריים של הארגון המודרני**.

הבה נתבונן ב**מעגל המידע** – תהליך ההופך נתונים להחלטות ולפעולות:



תרשים 1.1: מעגל המידע.

נסקור את השלבים השונים של מעגל המידע:

- **שלב 1 – נתונים:** הנתונים נוצרים כתוצאה מדיווח על אירועים ועובדות. לדיווחים המעדכנים את הנתונים מקובל לקרוא **תנועות עסקיות** (Business Transactions). הנתונים מנוהלים בדרך כלל במערכת מידע ממוחשבת המנהלת ואוגרת אותם לאורך זמן.
- **שלב 2 – מידע:** הנתונים המנוהלים במערכת המידע עוברים תהליכי עיבוד שונים, כגון מיון, ביצוע חישובים שונים והפיכה להצגה גרפית ומוצגים למשתמשים.

הנתונים מוצגים למשתמשים בצורות שונות: דוחות, שאילתות, גרפים, טבלאות, מפות ועוד.

- **שלב 3 – החלטות:** בני האדם, **המשתמשים** (Users) כפי שמקובל לקרוא להם, משתמשים במידע כדי להבין, לנתח את המצב ולקבל החלטות עסקיות.
- **שלב 4 – פעולות:** החלטות עסקיות הופכות, בסופו של דבר, **לפעולות** (Actions) המשפיעות על המציאות ולכן הן מדווחות למערכת המידע וחוזר חלילה.

נתבונן כעת כיצד מעגל המידע פועל בסביבה של המכללה שלנו. נניח שמערכת המידע מנהלת נתונים על מספר הסטודנטים שנרשמו לקורסים השונים במשך השנים. מדי שנה, המזכירות האקדמית של כל מחלקה מקצה אולמות הרצאה וכיתות לקורסים השונים. מכיון שכיתות שונות יכולות לאכלס מספר שונה של סטודנטים, יש חשיבות לקבוע איזה קורס יועבר באיזו כיתה. לדוגמה, חבל להקצות כיתה שבה 100 מקומות ישיבה לקורס שבממוצע יש בו 30 סטודנטים. לקראת הקצאת הכיתות, מופק דוח המציג את מספר הסטודנטים שנרשמו לקורסים השונים במשך שלוש השנים האחרונות, איזה כיתות הוקצו לקורס בכל שנה ומה אחוז הגידול או הצימצום במספר הנרשמים. על סמך דוח זה ועל סמך מידע נוסף, מתקבלת ההחלטה איזו כיתה תוקצה לכל קורס. החלטה זו מדווחת חזרה למערכת המידע באמצעות מסך עדכון מיוחד – מסך הקצאת כיתה לקורס. הבסיס של כל המעגל הזה הם הנתונים.

כדי להקים מערכת מידע שתשרת את הארגון, יש להבין את המציאות בה הארגון פועל; להגדיר עבור איזה חלק מהמציאות יש לבנות מערכת מידע ממוחשבת; לעצב מודל המתאר את המציאות הרלוונטית; לפתח את התוכנה המתאימה שתבצע את התנועות העסקיות באמצעותן מדווחים האירועים השונים המתרחשים במציאות; לנהל את הנתונים הרלוונטיים לאורך זמן; ולבסוף – לבנות את מערכת התוכנה המעבדת ומציגה את הנתונים על פי הצורך למשתמשים השונים.

בספר זה נתמקד בתהליך ניהול הנתונים באמצעות **מערכות לניהול בסיסי נתונים**. אך לפני שנסביר מהו בסיס הנתונים ומהי מערכת לניהול בסיסי נתונים, נסקור מספר מושגי יסוד, ונשתמש בתיאור שהצגנו על המכללה וגם במספר דוגמאות נוספות. על רוב המושגים שנציג כאן נרחיב את הדיון בפרקים הבאים.

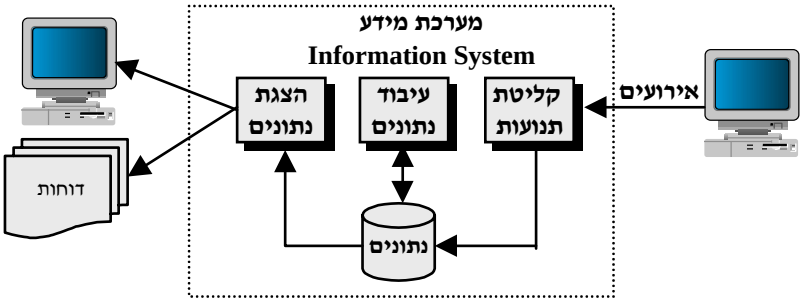
מערכת מידע

מערכת מידע **Information System**	***מערכת מידע*** **היא מערכת תוכנה המאפשרת את ניהול הנתונים, תומכת בביצוע פעולות עדכון הנתונים ומאפשרת את המרתם למידע בעל משמעות. לעיתים מקובל לקרוא למערכת מידע גם בשם 'יישום' (Application).**

מערכות המידע הן מערכות תוכנה שנבנו על ידי מהנדסי תוכנה ומבצעות פונקציה מסוימת. ניתן להתבונן במערכת מידע בשני אופנים: מנקודת המבט של **נתונים** ומנקודת המבט של ה**תוכנה**. עם זאת, זהו צמד שאיננו בר-הפרדה. אין כל משמעות לעסוק במערכת

מידע מבלי להבין מהן **התנועות** והפעולות שהתוכנה מאפשרת לבצע על הנתונים ומהם **תהליכי העיבוד** להפיכת הנתונים למידע, כפי שאין כל משמעות לדבר על מערכת מידע מבלי להבין מהם **הנתונים** שהיא מנהלת.

ניתן לחלק את מערכת המידע לשלושה רכיבי תוכנה עיקריים: רכיב העוסק ב**קליטת התנועות** העסקיות, רכיב העוסק **בעיבוד הנתונים** ורכיב העוסק ב**הצגת הנתונים**.



תרשים 1.2: רכיבי מערכת המידע.

כל שלושת הרכיבים מתבססים על רכיב נוסף: רכיב **ניהול הנתונים**. רכיב זה אחראי לניהול הנתונים לאורך זמן ולאספקת הנתונים הדרושים לכל אחד מהרכיבים האחרים.

רכיבי מערכת המידע

רכיב קליטת תנועות (Transaction Processing)

רכיב זה עוסק בקליטת התנועות העסקיות השונות. נגדיר תחילה מהי תנועה עסקית.

תנועות עסקיות **Business Transaction**	***תנועה עסקית*** **מוגדרת כפעולה בעלת משמעות עסקית, המעדכנת את הנתונים; כלומר, היא משנה את התוכן שלהם.**

מערכת מידע יכולה לתמוך בעשרות ומאות סוגים שונים של תנועות עסקיות. לדוגמה תנועה לשינוי כתובת המגורים של סטודנט המעדכנת את כתובת המגורים הנוכחית שלו לכתובת החדשה אליה הוא עבר. דוגמה אחרת היא פעולה של מכירת כרטיס טיסה המעדכנת את הסטטוס של מושב מסוים בטיסה בין תל אביב לניו-יורק מפנוי לתפוס ורושמת את פרטי הלקוח שהזמין את הכרטיס, את המחיר בו נקנה הכרטיס ואת צורת התשלום.

קיימות שתי שיטות לביצוע התנועות במערכות המידע – בצורה מקוונת או באצווה:

- **עיבוד תנועות מקוון – OLTP (On Line Transaction Processing)**: הצורה הנפוצה והמודרנית היא לאפשר את ביצוע התנועות העסקיות בצורה מקוונת (On Line), כלומר, מאפשרת למשתמש להזין את נתוני התנועה בתחנת העבודה שלו (מחשב אישי או מסוף). מערכת המידע קולטת את הנתונים שהוזנו על ידי המשתמש, בודקת אותם על פי הכללים שהוגדרו ומעדכנת באופן מיידי את הנתונים הרלוונטיים.

- **עיבוד באצווה (Batch Transaction Processing)**: בשיטת עבודה זו מתבצע תהליך של צבירת אירועי העדכון השונים והעדכון של התנועות מתבצע במרוכז אחת לתקופה. העדכון יכול להתבצע פעם בשעה, פעם ביום, או כל תקופה מוגדרת אחרת, או למשל, לאחר שהצטברה כמות מסוימת של אירועים. מערכות המידע הראשונות תמכו בעיקר בשיטת עיבוד זו של תנועות. עם הופעת מערכות התקשורת, המסופים ולאחר מכן המחשבים האישיים עם הממשקים הגרפיים המתוחכמים שלהם, שיטה זו הוחלפה כמעט כולה על ידי מערכות OLTP. למרות שרוב מערכות המידע המודרניות הן מקוונות, יש לא מעט מקרים שבהם שיטה זו של עיבוד באצווה היא המתאימה ביותר. לדוגמה אם נתייחס לחברת טלקומוניקציה המחייבת את לקוחותיה בגין ביצוע שיחות טלפון, הדרך המקובלת לקליטת האירועים היא על ידי רישום השיחות באופן אוטומטי על ידי המרכזיה והעברתן במנות לצבירה וקליטה במערכות החיוב והגביה (Billing).

שתי שיטות קליטת תנועות אלו, אינן מוציאות זו את זו אלא מהוות שתי שיטות משלימות. בחלק גדול ממערכות המידע המודרניות נמצא את שתי השיטות גם יחד. לדוגמה, בחברות הטלקומוניקציה ליד מערכת החיוב והגביה הפועלת בעיקר בצורה של קליטת אירועים באצווה נמצא את מערכת ניהול הלקוחות (Customer Care) הפועלת בצורה מקוונת, כך שכל הזמנה חדשה, תלונה של לקוח או שינוי באחד מנתוני הלקוח מעדכנים בצורה מקוונת ומיידית את הנתונים.

בדרך כלל, רכיב זה מבצע אוסף נרחב של בדיקות תקינות של הנתונים המופיעים בתנועה, לפני פעולת העדכון עצמה. רק תנועות המכילות נתונים תקינים מורשות לעדכן את הנתונים.

רכיב עיבוד הנתונים (Data Processing)

רכיב זה עוסק בביצוע עיבודים שונים של הנתונים, כפי שנדרש על ידי התהליכים העסקיים. לדוגמה, במערכת שכר מתבצע עיבוד חודשי להפקת תלושי המשכורת, במערכת פיננסית מתבצע עיבוד רבעוני להפקת מאזן ודוח רווח והפסד רבעוני, במערכת החיוב והגביה של חברת טלקומוניקציה מתבצע עיבוד חודשי או דו-חודשי להפקת חשבון הטלפון של הלקוח, במערכת בנקאית מתבצע תהליך חודשי של חישוב הריבית המגיעה ללקוח בחשבון חיסכון וכד'.

רכיב הצגת הנתונים (Data Presentation)

בנוסף לרכיב העוסק בקליטת תנועות עסקיות המעדכנות את הנתונים ולרכיב העוסק בעיבוד הנתונים, תומכת כל מערכת מידע במיגוון רחב של שיטות לעיבוד והצגת הנתונים, כלומר בהמרת הנתונים למידע. לדוגמה, במערכת המנהל האקדמי של המכללה ניתן להציג גרף של הגידול במספר הסטודנטים הנרשמים למחלקה למדעי המחשב, לבנות דוח המציג את הזכאות לתואר של סטודנט על פי המחלקה בה הוא למד, הקורסים בהם הוא למד, מספר נקודות הזכות של כל קורס והציונים שהשיג בכל קורס.

דוגמה הלקוחה ממערכת מלאי יכולה להיות שאילתה מקוונת בתחנת העבודה של המשתמש המציגה את רשימת כל הפריטים שהיתרה שלהם במלאי היא אפס, או את רשימת ערך הפריט במלאי (מכפלה של מחיר הפריט ביתרת המלאי) בסדר יורד; או גרף המציג את מספר הפריטים שהתקבלו בכל חודש במשך השנה האחרונה. כל שאילתה או דוח מסוגים אלה הינם בעלי משמעות ומאפשרים לעובדים לקבל החלטות. למשל, על סמך השאילתה של כל הפריטים ביתרה אפס ניתן להחליט איזה פריטים יש להזמין מהספקים; ועל סמך התפלגות הכניסות למחסן בכל חודש ניתן להחליט מהו מספר העובדים שיש להעסיק במחסן הקבלה.

רכיב ניהול הנתונים (Data Management)

רכיב זה עוסק בניהול הנתונים לאורך זמן (Persistent Data). הנתונים מנוהלים בדרך כלל ביחידות אחסון, כגון דיסקים מגנטיים, תקליטורים וקלטות. רכיב זה מאפשר לכל שאר הרכיבים לאתר את הנתונים במהירות, לעדכן אותם ולהציג אותם בעת הצורך. מערכות המידע המודרניות מסוגלות כיום לקלוט ולעבד כמויות עצומות של נתונים ולשרת מספר גדול מאוד של משתמשים בו-זמנית. במשך השנים התפתחו מערכות שונות לניהול נתונים, **מערכות לניהול קבצים** (File Management Systems) ומערכות מודרניות יותר – **מערכות לניהול בסיסי נתונים** (Database Management Systems). הדור הנוכחי של מערכות לניהול בסיסי נתונים נקרא **מערכות לניהול בסיסי נתונים טבלאיים** (Relational Database Management System), מכיון שמערכות אלו מנהלות את הנתונים באמצעות מבנה נתונים פשוט יחסית – **טבלה**. מערכות ניהול בסיסי נתונים אלו הן נשוא ספר זה, ועליהן נרחיב את הדיון בהמשך.

יישום (Application)

'**יישום**' הינו השם הכולל לכל ארבעת הרכיבים שתוארו לעיל, המשרתים תכלית מסוימת. '**מערכת**' ו'**יישום**' הם שמות חליפיים. הבה נתבונן במספר דוגמאות ליישומים שונים:

- **מערכת מינהל אקדמי** – מערכת זו עוסקת בכל ההיבטים של ניהול מוסד אקדמי ומחזיקה נתונים אודות הסטודנטים, המרצים, המחלקות האקדמיות, הקורסים המוצעים על ידי המחלקות השונות, הבניינים וחדרי הכיתות ועוד. מערכת זו תומכת בפעולות מגוונות של ניהול הנתונים, כגון קבלת סטודנט חדש לפקולטה מסוימת, רישום סטודנט לקורס, רישום ציוני הבחינות, קבלת מרצה חדש לעבודה, קידום מרצה במסלול הקידום האקדמי ופתיחת קורס חדש.

- **מערכת מלאי** – מערכת זו עוסקת בניהול כל הנתונים הרלוונטיים לניהול מלאי, וביניהם: מחסנים, אתרים, פריטים, ספקים, לקוחות ועוד. היא תומכת בביצוע כל הפעולות הדרושות לניהול מלאי בארגון: פתיחת פריט חדש, פתיחת אתר חדש במחסן, ביטול אתר קיים, רישום כניסת פריט למלאי, הנפקת פריט מהמחסן ללקוח, ספירת מלאי וכד'.
- **מערכת לניהול הזמנות בחברת תעופה** – מערכת מידע זו מנהלת נתונים העוסקים בנמלי תעופה, מטוסים, טייסים, טיסות, מושבים במטוס, כרטיסי טיסה, לקוחות, מזון לטיסות ועוד. היא תומכת בביצוע פעולות הקשורות לנתונים אלה, וביניהן: הגדרת יעד טיסה חדש, הגדרת לוח טיסות לעונה חדשה, ציוות טייסים לטיסות, מכירת כרטיסי טיסה, קליטת מטוס חדש והזמנת מנות מזון לטיסות.
- **מערכות לניהול ותכנון משאבי הארגון – ERM** (Enterprise Resource Management). מערכות אלו נקראות גם **מערכות ERP** (Enterprise Resource Planning). ייחודן בכך שהן מורכבות מאלפי טבלאות נתונים וממיליוני שורות קוד וייעודן לעסוק במיגוון פעילויות של הארגון, וביניהם: ניהול רכש, ניהול מלאי, ניהול משאבי אנוש, ניהול ייצור, ניהול אחזקה, ניהול פרויקטים, ניהול פיננסי, תמחיר, ניהול הזמנות לקוח ומכירות ועוד. מערכות אלו מאופיינות במידת האינטגרציה הגבוהה בין המודולים השונים המרכיבים אותן וכתוצאה – יכולת אינטגרציה של נתונים בתחומים שונים. לדוגמה, רישום כניסה למלאי מעדכן מיידית את מצב ההזמנה של הספק, נותן התראה למנהל הייצור שהגיע פריט שהיה בחוסר ונרשם במערכת הפיננסית.

היישומים יכולים להיות פשוטים יחסית, כמו מערכת מלאי, או מורכבים מאוד כמו מערכת לניהול טיסות או מערכות ERP. ככל שהשימוש בטכנולוגיית המחשוב הלכה והשתפרה, הפכו היישומים לגדולים ומורכבים מאוד.

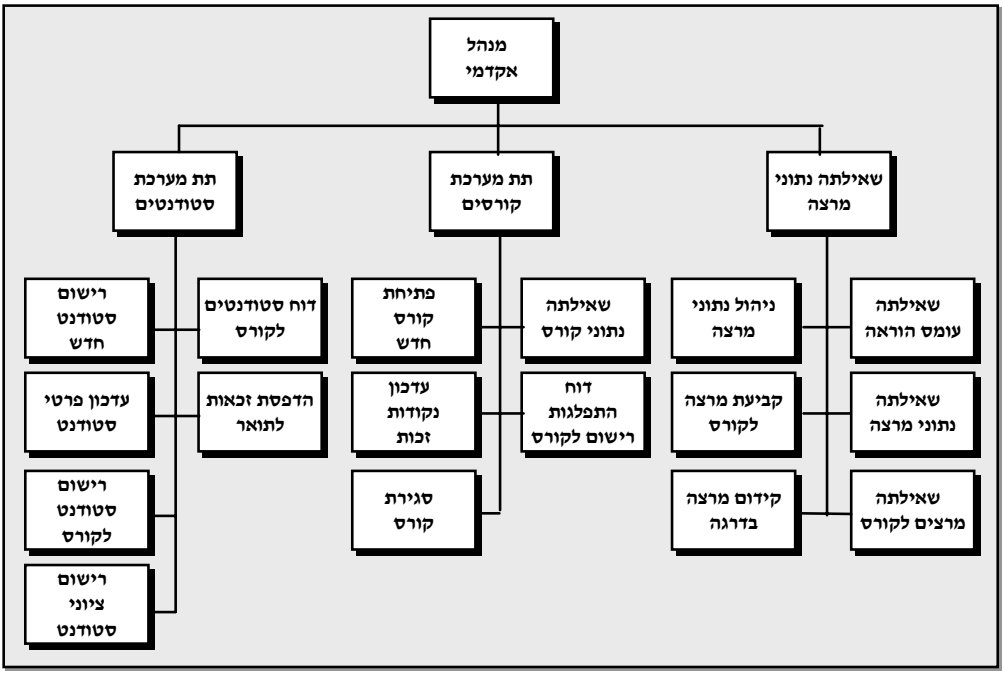
תוכניות יישום (Application Programs)

יישום יכול להיות מורכב מאוד ולכן מקובל לבנות אותו על ידי אוסף של תוכניות מחשב, שכל אחת מהן נקראת **תוכנית יישום**. יישום מורכב אחד יכול להיות מורכב ממאות או מאלפי תוכניות יישום.

תוכנית יישום **Application Program**	***תוכנית יישום*** **מוגדרת כיחידת תוכנה בסיסית המבצעת פעולה כלשהי (תנועה, עיבוד, הצגת נתונים, או שילוב כלשהו ביניהם).**

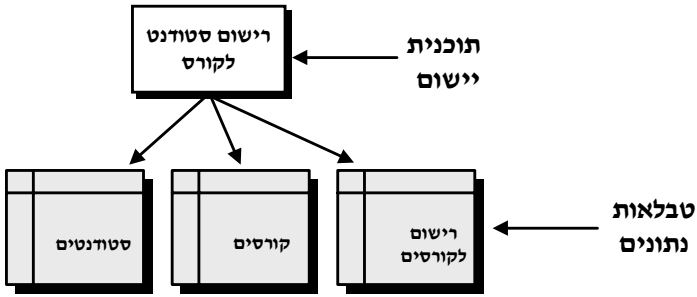
כל תוכנית יישום יכולה לבצע תנועה עסקית אחת או יותר, לבצע עיבוד כלשהו של הנתונים, לעסוק בהצגת הנתונים או לבצע שילוב כלשהו של שלוש הפעולות האלו. למעשה, החלוקה של יישום לתוכניות יישום הממלאות משימות ומטלות פשוטות ומוגדרות מאוד נובעות מהחלטת מעצב המערכת לחלק את המכלול ששמו 'יישום' ליחידות קטנות יותר שתהיינה **ניתנות לשליטה ולתחזוקה** על ידינו, בני האדם! לחלוקה זו יש יתרונות רבים שלא כאן המקום לפרטם.

תרשים 1.3 מציג יישום אחד המורכב משלוש תת-מערכות. בכל תת-מערכת ניתן להבחין במספר תוכניות יישום העוסקות בביצוע תנועות ובמספר תוכניות יישום העוסקות בעיבוד הנתונים ובהצגת מידע. לדוגמה, בתת-מערכת סטודנטים יש שש תוכניות יישום שונות. ארבע מתוכן מיועדות לבצע תנועות עסקיות המעדכנות את הנתונים הקשורים לסטודנטים, ושתי תוכניות יישום המציגות את הנתונים.



תרשים 1.3: דוגמה ליישום מנהל אקדמי.

הנתונים הדרושים לתוכניות היישום מנוהלים בבסיס הנתונים במבנה של טבלאות. תוכנית יישום אחת יכולה לגשת לטבלה אחת או יותר. לדוגמה, תוכנית היישום המבצעת את רישום הסטודנטים לקורס יכולה לגשת ולעדכן את טבלת הסטודנטים, את טבלת הקורסים ואת טבלת הרישום לקורסים.



תרשים 1.4: תוכנית יישום הניגשת לשלוש טבלאות נתונים שונות.

כל תוכנית יישום היא למעשה תוכנית מחשב הכתובה בשפת תכנות כלשהי (++C, Cobol, Visual Basic, Java, Magic, Developer/2000 ו- PowerBuilder). תוכנית היישום מבצעת את הלוגיקה העסקית, הכוללת בדיקות תקינות, חישובים ועיבודים וגם את עדכון הנתונים בטבלאות השונות. לדוגמה, תוכנית הרישום לקורס תבדוק אם הסטודנט למד בכל הקורסים המוגדרים כתנאי-קדם לקורס, תבדוק אם עדיין נותר מקום פנוי בקורס ותעדכן את הנתונים הרלוונטיים וביניהם: הוספת שורה בטבלת הרישום לקורסים, הגדלת מספר הסטודנטים הלומדים בקורס מסוים ועוד.

כפי שנראה בהמשך הספר, לעובדה שתנועה אחת יכולה לעדכן מספר טבלאות שונות יש משמעות רבה. טכנולוגיית בסיסי הנתונים צריכה להבטיח שתנועה אכן סיימה בהצלחה את עדכון כל הטבלאות או שהיא לא עדכנה אף טבלה. לתכונה מיוחדת זו של תנועה עסקית מקובל לקרוא **הכל או לא כלום (All or Nothing)**. בפרק 14 ניתן הסבר כיצד המערכות לניהול בסיסי נתונים תומכות בתנועות עדכון.

המשתמש מפעיל את תוכניות היישום השונות על ידי בחירת הפעולות מתוך תפריט (Menu) או על ידי פתיחת חלונות שונים. לתנועה של המשתמש בתוך מערכת המידע, בין תת-המערכות השונות, בין תוכניות היישום השונות ובין הפונקציות השונות שכל תוכנית יישום מסוגלת לבצע, מקובל לקרוא בשם **ניווט** (Navigation).

מערכות תפעוליות ומערכות מחסן נתונים

לכלל היישומים העוסקים בתפעול השוטף של הארגון מקובל לקרוא **יישומים תפעוליים של הארגון** (Operational Information Systems או Transactional Systems). אלה הם בדרך כלל היישומים הראשונים שכל ארגון מפתח או רוכש, ומיישם. יישומים אלה הם לעיתים מורכבים ביותר, אך למעשה עוסקים במחשוב תהליכי עבודה מובנים וידועים מראש, התומכים בתפעול השוטף של הארגון. המשתמשים העיקריים ביישומים אלה הם הדרג התפעולי בארגון: עובדים המשרתים את הקהל כמו היחידה למרשם הסטודנטים ומזכירויות הפקולטות ביישום אקדמי; או מחסנאים, קניינים, עובדים בקו הייצור, אנשי שירות, מנהלי חשבונות ותמחירנים ביישום תעשייתי. הנתונים הנאגרים בבסיסי הנתונים של יישומים אלה הם בעלי ערך רב, מכיון שבלעדיהם התפעול השוטף של הארגון בלתי אפשרי. בנוסף לדרג התפעולי משמשים הנתונים במערכות התפעוליות גם את דרג המנהלים, לקבלת החלטות תפעוליות שוטפות.

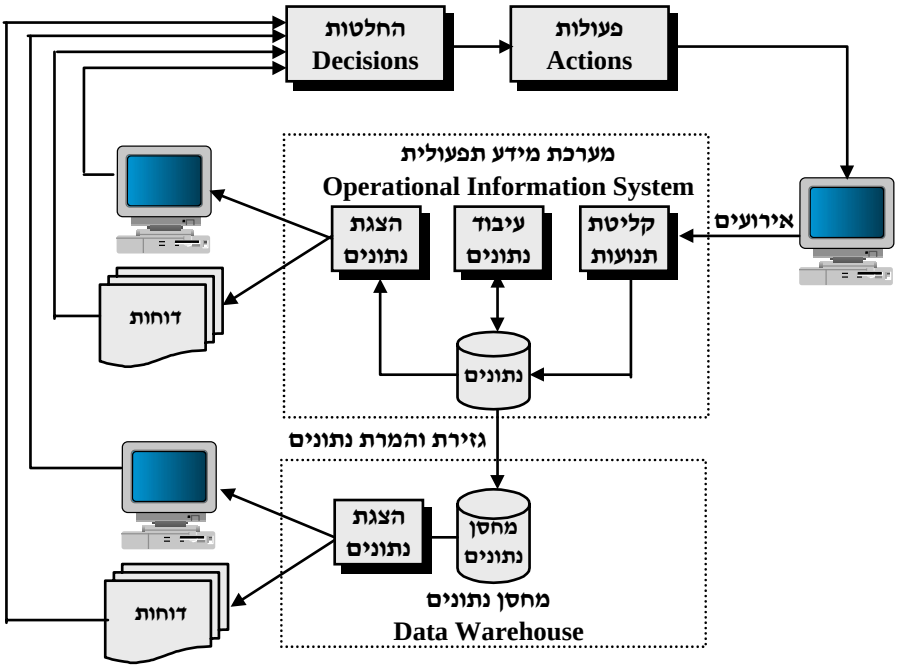
במקביל למערכות התפעוליות של הארגון, וכתוצאה מהצורך הרב במידע, החלו להתפתח סביבות מחשוב ייעודיות ל**ניתוח נתונים** ול**תמיכה בהחלטות**. מערכות אלו נקראות **מחסני נתונים** (Data Warehouses), או **מערכות תומכות החלטה** (Decision Support Systems) והן מיועדות בעיקר לדרג מקבלי ההחלטות בארגון. יישומים אלה אינם עוסקים באופן ישיר בתפעול הארגון, אולם חשיבותם אינה פחותה. בזכותם יכולים מקבלי ההחלטות השונים בארגון לראות את התמונה העסקית העדכנית בכל עת ולקבל החלטות באופן שוטף ומיידי. אוכלוסיית המשתמשים ביישומים אלה שונה ובעלת צרכים ודרישות

שונים: מנהלים ברמות שונות בארגון, אנשי תכנון ובקרה, כלכלנים שתפקידם לנתח מגמות, מנהלי שיווק, ראש מחלקה במכללה, ראש מנהל הסגל במכללה ואחרים.

היישומים התומכים בקבלת החלטות עוסקים בניתוח מצב הארגון. הם יכולים להציג את מה שקרה (העבר) ואת משמעויות השינויים המתחוללים, ומאפשרים למקבלי ההחלטות לצפות במידה מסוימת את אשר צפוי לקרות אם ינקטו פעולות כאלו או אחרות. הפעלת יישומים מסוג זה תומכת בפעילות הארגון לטווח הארוך יותר מאשר בתהליכים קצרי טווח העוסקים בפעילות היומיומית השוטפת. להלן מספר דוגמאות ליישומים השייכים לקבוצה זו:

- ניתוח רישום סטודנטים לקורסים על פי נתונים דמוגרפיים.
- דיווחים חודשיים לגבי מגמות ברמות המלאי במחסנים.
- ניתוח הצטרפות לקוחות חדשים על ציר הזמן.
- תזרים המזומנים הצפוי.
- ניתוח השפעת שינוי מחיר המוצר על נפח פעילות החברה.
- ניתוח השינויים במכירת המוצרים לפי סוגי המוצר והאזור בארץ.

בדרך כלל, יישומים אלה מתבססים על נתונים הנגזרים מתוך היישומים התפעוליים ומנוהלים במחשב ייעודי ובבסיס נתונים נפרד.



תרשים 1.5: מערכות תפעוליות ומחסני נתונים.

כפי שניתן לראות מתרשים 1.5, המערכות התפעוליות מזינות את מחסני הנתונים. הנתונים נגזרים מתוך המערכות התפעוליות, עוברים תהליכי המרה שונים, כמו גם תהליכי טיוב ופישוט, ומועברים אל מחסן הנתונים. כאן מנוהלים על פי רוב הנתונים בבסיס נתונים טבלאי, ומוצגים למשתמשי מחסן הנתונים השונים. כלי הצגת הנתונים בסביבת מחסן הנתונים הם בדרך כלל כלים מתקדמים כגון מחוללי שאילתות גרפיים, מערכות להצגה וניתוח נתונים רב-מימדיים (Multi-Dimensional), כלים סטטיסטיים וכלים מתקדמים לכריית נתונים (Data Mining). המידע המופק ממחסני הנתונים משמש לקבלת החלטות על ידי דרגים שונים. דרג מקבלי ההחלטות הזוטר יותר בארגון יכול להסתפק במידע המופק ישירות מהמערכות התפעוליות, בעוד שדרגי קבלת ההחלטות הבכירים יותר זקוקים למידע המופק ממחסן הנתונים.

סיכום

בפרק זה הצגנו את הרקע של **מערכות ניהול הנתונים** ואת מקומן בהקשר הרחב יותר של **מערכות המידע**. מערכות המידע המודרניות מהוות כיום את אחד הכלים המרכזיים של כל ארגון מודרני. ככל שהשוק שבו פועל הארגון הופך לתחרותי יותר, הזמינות של מערכת מידע המאפשרת קבלת החלטות המבוססות על נתונים אמינים ועדכניים הולכת ונעשית חיונית עוד יותר. קיים צורך בלתי פוסק להבין את צרכי הלקוחות, לייעל את תהליכי הייצור והשירות, לפתח מוצרים חדשים במהירות, לשפר את איכות השרות ללקוח ועוד. כל אלה הביאו לכך שכל הארגונים, קטנים כגדולים, מתבססים במידה רבה על מערכות מידע לתפקוד השוטף שלהם.

במסגרת מערכת המידע, יש לרכיב **ניהול הנתונים** חשיבות מרובה, כי הוא זה אשר מבטיח את שמירת הנתונים לאורך זמן, את זמינותם בעת הצורך, את היכולת לעדכן אותם ולהבטיח את אמינותם. על רקע זה קיבלו **המערכות לניהול בסיסי נתונים** תנופה אדירה והפכו לרכיב הכרחי בכל מערכת מידע מודרנית. מנתחי שוק שונים מעריכים כי שוק המערכות לניהול בסיסי נתונים בשנת 1999 הוא בהיקף של כ-13 מיליארד דולר. מתוך שוק זה, כ-11 מיליארד דולר הם חלקן של המערכות לניהול בסיסי נתונים טבלאיים.

ללא קשר לסוג מערכת המידע – מערכת תפעולית או מערכת תומכת החלטות, הצורך בניהול יעיל של נפחי נתונים גדולים קיים, והקושי לעמוד בדרישה זו הולך ומחמיר. הופעת האינטרנט והתפוצצות המידע הנלווית לתופעה זו, רק הגבירו את הצורך בניהול היקפים עצומים של נתונים במבנים שונים: מלל, תמונות, קול, מידע גיאוגרפי ואחרים. על רקע זה, ברורה מרכזיותה של טכנולוגיית בסיסי הנתונים בסביבת מערכות המידע. לא ניתן כיום לתאר מערכת מידע כלשהי, בין אם היא תפעולית, תומכת החלטות או מיועדת להפצת מידע, ללא תשתית מוצקה לניהול נתונים, או במילים אחרות – ללא **מערכת לניהול בסיסי נתונים**. כאן נכנסת לתמונה טכנולוגיית בסיסי הנתונים, ועל כך נרחיב בהמשך.

שאלות חזרה ותרגילים

שאלות חזרה

1. מהו מעגל המידע והשלבים השונים שלו?
2. מה ההבדל בין עיבוד תנועות מקוון לבין עיבוד אצווה?
3. הסבר את ההבדל בין מערכת מידע תפעולית לבין מחסן נתונים. מה הם הקשרים בין שני סוגי מערכות אלו?

תרגילים

1. התבונן במערכות המידע בארגון בו הינך עובד או לומד. נסה לסווג אותן על פי אחד מסיווגים אלה: מערכות אצווה, מערכות לעיבוד תנועות מקוונות, מערכות תפעוליות ומערכות תמיכה בקבלת החלטות.
2. בחר אחת מהמערכות שציינת בשאלה 1 ובחן את המבנה שלה מבחינת החלוקה למודולים. ציין את תפקידי המודולים ואת הקשרים ביניהם.

פרק 2

עקרונות מערכות לניהול בסיסי נתונים

1. מבוא
2. ציוני דרך עיקריים בהתפתחות מערכות לניהול נתונים
3. מערכות ניהול קבצים
4. מערכות לניהול בסיס הנתונים
5. מודל העבודה של מערכת RDBMS
6. תפקידים שונים בסביבת בסיס נתונים
7. עיקרון הפעולה של מערכות RDBMS
8. שירותים במערכת לניהול בסיסי נתונים
9. יתרונות וחסרונות של טכנולוגיית בסיסי הנתונים
10. סיכום
11. שאלות חזרה ותרגילים

מבוא

טכנולוגיית **בסיסי נתונים והמערכות לניהול בסיסי נתונים** – **DBMS** (DataBase Management Systems) הינה טכנולוגיה ותיקה שראשיתה בתחילת שנות ה-60. היא התפתחה על רקע הקשיים ההולכים וגוברים בניהול הנתונים באמצעות הטכנולוגיה שקדמה לה, טכנולוגיית **מערכות ניהול הקבצים** – **FMS** (File Management Systems). טכנולוגיית בסיסי הנתונים עצמה עברה מספר שינויים והתפתחויות, והחל ממחצית שנות ה-80 הטכנולוגיה הנפוצה ביותר היא זו של **המערכות לניהול בסיסי נתונים טבלאיים** – **RDBMS** (Relational DataBase Management Systems). כיום, מערכות מסוג RDBMS נפוצות מאוד ומנהלות את הנתונים במערכות המידע שפועלות במרבית הארגונים: במחשבים מרכזיים, בשרתים מחלקתיים או במחשבים אישיים, ללא תלות אם ניהול הנתונים הוא עבור מערכת תפעולית או עבור מחסן נתונים. טכנולוגיה זו חדרה גם למשתמש הביתי העובד עם מחשב אישי ובונה לעצמו מאגרים וכרטסות שונות בביתו. ספר זה מתמקד בעיקר בטכנולוגיית בסיסי הנתונים הטבלאיים, אשר מהווה את תשתית ניהול הנתונים במיגוון יישומי מחשב.

בפרק זה

- נסקור בקצרה את ההתפתחות שחלה במערכות ניהול נתונים מאז הופעת מערכות ניהול הקבצים, דרך הופעת טכנולוגיית בסיסי הנתונים ועד להפיכת מודל הנתונים הטבלאי ומערכות RDBMS המבוססות עליו, למערכות נפוצות.
- נסקור את טכנולוגיית המערכות לניהול קבצים ואת הבעיות העיקריות בטכנולוגיה זו. בעיות אלו היו הרקע לפיתוח טכנולוגיה חדשה שקרויה טכנולוגיית בסיסי הנתונים. נדון בצורך להתאים את מבנה הנתונים לדרישות משתנות ונסקור את הבעיות שנגרמות מהיעדר גישה אינטגרטיבית בבניית מאגר הנתונים.
- נציג את ההגדרה של בסיס הנתונים ושל המערכת לניהול בסיסי הנתונים.
- נציג את ארכיטקטורת ANSI/SPARC, שהגדירה את המבנה העקרוני של מערכת ניהול בסיסי נתונים. נשתמש בהגדרה זו כמצע לדיון במושגים המרכזיים של טכנולוגיית בסיסי הנתונים – סכימה, תת-סכימה וסכימה פנימית – מנגנוני הפשטה, שתפקידם להסתיר את מבנה הנתונים מהמשתמשים בו.
- נסקור את עיקרון הפעולה של מערכת RDBMS, ונבחין בין שלב הגדרת הנתונים לבין שלב השימוש בהם. נסביר את אפשרויות הגישה השונות לבסיס הנתונים: דרך תוכניות יישום הכתובות בשפת תכנות מדור שלישי או רביעי, דרך שפות שאילתות ודרך מחוללי דוחות.
- נסקור בקצרה את תפקידי מנהל בסיס הנתונים, פונקציה ארגונית חדשה שהתפתחה עם הופעת טכנולוגיית בסיסי הנתונים.

- נסקור את מיגוון השירותים שמערכת RDBMS מודרנית צריכה לספק. נתייחס למושגים חשובים, כמו שירותי אי-תלות לוגית ופיסית, ניהול קטלוג המערכת המכיל את כל הגדרת מבנה הנתונים באופן חיצוני לתוכניות היישום, שירותי תמיכה בעיבוד תנועות מקוון.
- נסכם בסקירת היתרונות והחסרונות העיקריים של טכנולוגיית בסיסי הנתונים.

ציוני דרך עיקריים בהתפתחות מערכות לניהול נתונים

לפני שנפרט את המבנה ואת עקרונות הפעולה של המערכות לניהול בסיסי נתונים, נסקור בקצרה את ציוני הדרך העיקריים בהתפתחות מערכות אלו. ניהול הנתונים באמצעי אחסון היה מאז ומתמיד נושא בעל חשיבות רבה, לאור העובדה שהנתונים הם מרכיב חשוב בכל מערכת מידע. סקירה קצרה זו תיתן לקורא מבט מקיף על השינויים שניהול הנתונים עבר ועל מיקומה של טכנולוגיית בסיסי הנתונים.

מערכות בסיסיות לניהול נתונים

מערכות בסיסיות לניהול נתונים היו קיימות כבר בתחילת שנות ה-60. הן דמו יותר לאוסף של הוראות קלט ופלט, מאשר למערכת ניהול נתונים. כל תוכנית יישום טיפלה בקובץ אחד או יותר, בהתאם לדרישות היישום. כל גישה לנתונים, הן הלוגית והן הפיסית, היוו חלק בלתי נפרד מהתוכנית עצמה ובוצעה באמצעות שגרות קלט/פלט שהיו משולבות בה. השגרות נכתבו על ידי המתכנתים בעמל רב, כיון שטיפלו ברמה הבסיסית של התכנות ובקשר שבין התוכנה לחומרה.

תרשים 2.1: מערכת בסיסית לניהול נתונים.

אמצעי האחסון הנפוץ באותה תקופה היה הסרט המגנטי. רק בשלב מאוחר יותר החלו להופיע דיסקים מגנטיים. לתוכניות היישום היתה תלות מוחלטת בצורת האחסון **הפיסית** של הנתונים. שיתוף נתונים בין יישומים שונים היה כמעט בלתי אפשרי. מגבלה זו גרמה לכך שקבצים שונים הכילו נתונים זהים בהתאם לצרכי היישום ולעיתים קרובות, אותם הנתונים הופיעו במבנים שונים (שם משפחה בקובץ אחד הכיל 40 תווים ובשני – 35 תווים). ככל שמספר היישומים הלך וגדל, כן גדלה גם כפילות הנתונים ונפגעה אמינותם. תוכניות שונות עדכנו בנפרד את אותם הנתונים שהופיעו בקבצים שונים וגרמו על כן לאי התאמה. המאפיין המרכזי של מערכות אלו היה הקושי הרב בגישה לנתונים ובניהולם. הקושי והאתגר בניהול הנתונים הפך לאחד החסמים העיקריים בהתפתחות מערכות המידע.

מערכות לניהול קבצים – FMS

מערכות לניהול קבצים – FMS (File Management Systems) ייעודיות החלו להופיע מתחילת שנות ה-60 והן נמצאות עדיין בשימוש נרחב. הן מציעות למפתח היישומים מיגוון מצומצם של מבנים לוגיים, אך משחררות אותו מטיפול במבנה הפיסי של הנתונים. מערכות אלו מטפלות בהצלחה בקבצים בעלי תבנית רשומה זהה, או משתנה, ומספקות כלים תקניים ליצירת הקובץ ואחזקתו. תוכניות היישום אינן מתייחסות למבנה הפיסי של הנתונים, אלא בעיקר למבנה ה**לוגי** שלהם. הן משיגות זאת באמצעות שגרות סטנדרטיות לגישה לנתונים אשר מהוות חלק מהמערכת לניהול קבצים. דוגמאות אופייניות למערכות אלו הן VSAM של יבמ, File System של מערכות Unix שונות, מערכת הקבצים במחשבים אישיים, Btrieve ברשתות נובל, ועוד.

תרשים 2.2: מערכת לניהול קבצים.

במערכות לניהול קבצים קיים קשר הדוק בין המבנה הפיסי לבין המבנה הלוגי. תוכניות היישום "מכירות" את התכונות הפיסיות של הקובץ כמו למשל מבנה הרשומה, מפתחות הקובץ, שיטת הגישה לקובץ, מספר רשומות לגוש ועוד. הקשר הקיים בין רשומות שונות בא לידי ביטוי בתוכניות היישום ועל כן, שינויים בארגון הקבצים מחייבים שינוי מקביל בתוכניות היישום.

תיאור הנתונים שהתוכנית השתמשה בהם והמבנה הפנימי של כל הרשומה היו לחלק ממנה, ללא אפשרות להתעלם מנתונים אלה, אפילו אם אינם דרושים. עם זאת, שחררו מערכות אלו את המתכנת מבעיות ההצגה הפיסית של הנתונים ביחידת האחסון. שיתוף נתונים בין יישומים שונים התאפשר על ידי השימוש במערכת לניהול קבצים, והושג צמצום מסוים בכפילותם.

בסעיף "מערכות ניהול קבצים" בהמשך נסקור מערכות אלו בהרחבה.

מערכות לניהול בסיסי נתונים – דור ראשון

מערכות לניהול בסיסי נתונים – **DBMS** (DataBase Management Systems) הופיעו לראשונה לקראת סוף שנות ה-60. היו אלו מערכת IMS של חברת יבמ ומערכת IDS של חברת הניוול. מערכת IMS התבססה על מודל נתונים מיוחד, שבשלב מאוחר יותר זכה לשם **מודל נתונים היררכי,** מכיון שהנתונים נוהלו כאוסף של היררכיות (מבנה עץ ששורשיו למעלה והעלים שלו למטה). מערכת נפוצה מאותה תקופה היתה מערכת Total של חברת Cincom, שהתבססה על מבנה נתונים גמיש יותר דמוי רשת בעלת מבנה מוגבל. מערכות אלו הקלו במעט על מפתחי היישומים, אם כי עדיין לא הציגו רמה גבוהה של אי-תלות בנתונים. מאמץ רב הושקע על ידי מוסדות מחקר, בתי תוכנה וארגונים שונים לקידום הידע ולהגדרת הדרישות ממערכות לניהול נתונים. בתחילת שנות ה-70 פורסמו ההצעות וההמלצות הראשונות למבנה רצוי של מערכת DBMS, מבנה שפות להגדרת נתונים ודרישות נוספות הקשורות בנושא בסיס הנתונים.

בשנת 1970 גובשה הצעה משותפת של שני ארגוני המשתמשים GUIDE ו-SHARE. לאחר מכן הוגשו ההמלצות של CODASYL (Conference On Data And System Language), ארגון שהיה אחראי לפיתוח וקביעת תקנים של COBOL והחליט לעסוק גם בבעיות של ניהול נתונים. CODASYL הקים צוות משימה מיוחד בשם DBTG (DataBase Task Group) לשם הגשת הצעה למודל נתונים רצוי במערכת לניהול בסיסי נתונים. הצוות חקר מספר מערכות ניהול קבצים וניהול בסיסי נתונים שהיו קיימות, ובשנת 1971 פרסם דוח שבו הציג את המלצותיו.

הדוח של CODASYL כלל ניסוח פורמלי של **מודל הנתונים הרשתי** (Network Data Model). מודל זה הניח שבין הנתונים קיימים קשרים מורכבים שניתן לתאר אותם במבנה רשת, שהוא מבנה משוכלל וגמיש יותר מהיררכיה. הדוח גם הגדיר לראשונה את החלוקה בין ה**סכימה** המתארת את **המודל הכללי** לבין **תת-הסכימה** המתארת את **המודל החיצוני**; הוא גם הציג את התפישה של **שפה להגדרת נתונים** בנפרד **משפה לטיפול בנתונים.** מודל הנתונים של CODASYL זכה להערכה רבה, כי הוא הציג לראשונה כיוון כללי לתכנון בסיסי נתונים וטיפל במספר בעיות עקרוניות חשובות.

בשנת 1975 הוצגה העבודה של ארגון התקינה הלאומי של ארה"ב – ANSI (American National Standards Institute). במסגרת הארגון פעלה קבוצת מחקר למערכות מחשב ומידע, צוות SPARC (Standards Planning And Requirements Committee on Computers and Information Processing), שעסק בגיבוש מסגרת לארכיטקטורה של מערכת DBMS. עבודה זו הציגה באופן פורמלי מערכת DBMS המורכבת משלוש רמות שונות.

אין ספק שדוח CODASYL והארכיטקטורה העקרונית שהוצעה על ידי ANSI היוו ציון דרך חשוב בהתפתחות מערכות DBMS. בהתבסס על המלצות אלו החלו מספר יצרנים לפתח מערכות ניהול נתונים, ביניהן IDMS של חברת Computer Associates (שבמקור פותחה על ידי Culinet), מערכת DBMS-11 של חברת דיגיטל ואחרות. חלק ממערכות אלו נמצאות בשימוש עד עצם היום הזה ומנהלות בהצלחה בסיסי נתונים גדולים ומורכבים מאוד בארגונים גדולים.

במקביל למערכות המסחריות המבוססות על המלצות CODASYL התפתחה מערכת ADABAS של חברת Software AG, אשר פועלת בעיקר על מחשבי יבמ מרכזיים. מערכת זו מבוססת על מודל ייחודי, **מודל הקבצים ההפוכים** (Inverted Files), שהוכיח את עצמו כמודל יעיל ביותר מבחינת הביצועים. המערכת פועלת בהצלחה מרובה במספר גדול של ארגונים, גם בארץ.

למרות הצלחת מערכות DBMS של הדור הראשון בניהול מבני נתונים מורכבים, בניהול היקפי נתונים גדולים מאוד וביצירת אי-תלות מסוימת בין תוכניות היישום לבין מבנה הנתונים, הן כשלו בסופו של דבר משתי סיבות: מורכבות ואי-גמישות לשינויים. העבודה עם מערכות אלו דרשה רמת מיומנות גבוהה של מפתחי היישומים. הם היו חשופים למבנה הנתונים – רשתות, היררכיות, קשרים בין סוגי רשומות – והיו צריכים ללמוד טכניקות ניווט (Navigation) מורכבות בתוך בסיס הנתונים. שינוי במבנה הרשת או ההיררכיה גרר אחריו שינויים משמעותיים בתוכניות היישום. על רקע זה החל להתפתח הדור החדש של מערכות ניהול בסיסי הנתונים.

מערכות לניהול בסיסי נתונים טבלאיים

בתחילת שנות ה-70 הציג Dr. Edgar Codd, שעבד באותה תקופה במעבדות המחקר של חברת יבמ, את **מודל הנתונים הטבלאי** (The Relational Data Model). הוא הציע גישה חדשנית לניהול נתונים, אשר מבוססת על מבנה נתונים פשוט ועל שפת ניהול נתונים מתוחכמת. שנות ה-70 היו מוקדשות למחקר המודל הזה ומאז תחילת שנות ה-80 החלו להופיע מערכות מסחריות מסוג **מערכות ניהול בסיסי נתונים טבלאיים - RDBMS** (Relational DataBase Management Systems). בתחילת הדרך מערכות אלו הפגינו חולשה בכל הקשור לביצועים, אולם במשך השנים שופרו הביצועים באופן משמעותי, וכיום הן מספקות ביצועים טובים ביותר. מערכות אלו כבשו את לב מפתחי היישומים והארגונים בגלל המבנה הפשוט, שפת נתונים בעלת עוצמה רבה והפרדה ברורה בין המבנה הפיסי לבין המבנה הלוגי, כפי שתוכניות היישום רואות אותו. המערכות לניהול בסיסי נתונים טבלאיים מהוות כיום את תשתית ניהול הנתונים העיקרית בכל מערכות המידע המודרניות, הן במערכות תפעוליות והן במערכות מחסני נתונים לתמיכה בהחלטות.

המערכות הטבלאיות נפוצות מאוד כיום וביניהן נוכל למנות את Oracle של חברת Oracle Corporation, Informix Dynamic Server של חברת Informix Software, Sybase ASE (Sybase Adaptive Server Enterprise) של חברת Sybase, SQL Server של חברת Microsoft, DB2 Universal Database של חברת יבמ, Ingres של חברת Computer Assiciates ועוד מספר רב מאוד. מודל העבודה של המערכות הטבלאיות יוסבר באופן מלא בפרק נפרד, ויוצג הקשר לשפת הגישה לנתונים – שפת SQL. שפה זו תומכת במיגוון רחב של פקודות המאפשרות את הגדרת בסיס הנתונים, את הגישה והעדכון של הנתונים המנוהלים בטבלאות, את הגדרת הרשאות הגישה לנתונים ועוד. שפה זו מוסברת בהרחבה בפרקים נפרדים. ספר זה מתמקד בטכנולוגיה זו, ולכן כל ההגדרות שתובאנה בהמשך תתייחסנה אליה, גם אם חלק מהן נוצרו לפני הופעתה ונועדו במקור עבור טכנולוגיות קודמות של בסיסי נתונים.

מערכות לניהול בסיסי נתונים מוכווני אובייקטים

ההצלחה העצומה של מערכות RDBMS ויישומים הפועלים בסביבה זו הציגה בצידה גם חולשות מסוימות של המודל הטבלאי, בעיקר בכל הקשור לניהול מבני נתונים מורכבים מאוד. ההתפתחות העצומה בטכנולוגיית המחשוב הביאה לחדירת המחשב כמעט לכל סוג יישום שהוא. היא גם איפשרה לנהל מבני נתונים לא-שגרתיים – תמונות, צלילים, וידאו, מפות, תרשימים סכמתיים (של מעגלים אלקטרוניים, למשל) וכד׳. התברר שמבנה הנתונים הפשוט של מערכות RDBMS אינו מסוגל לתמוך באופן ישיר ביישומים אלה. על רקע זה החלו להופיע בשנים האחרונות דור חדש של מערכות, **מערכות לניהול בסיסי נתונים מוכווני אובייקטים** – **ODBMS** (Object Oriented DBMS). למרות שמערכות אלו כבר נמצאות מספר שנים בשוק, הן לא הצליחו להגיע לפופולריות של מערכות RDBMS והן מיועדות יותר ליישומים ממוקדים, בעלי דרישות לניהול מיגוון רחב מאוד של אובייקטים שונים ומורכבים. במקביל למערכות אלו, נעשה כיום מאמץ פיתוח גדול גם על ידי יצרני מערכות RDBMS להעשיר ולגוון את מבני הנתונים הנתמכים על ידם, וכיום גם מערכות RDBMS מסוגלות לנהל מיגוון אובייקטים. מערכות אלו מקובל לכנות בשם **RODBMS** (Relational Object DBMS). לנושא RODBMS נקדיש את פרק 18.

מערכות ניהול קבצים

במקביל להתפתחות במערכות המידע ולצורך לנהל נפחי נתונים הולכים וגדלים וברמות מורכבות הולכות וגדלות, החלו להתפתח מערכות תוכנה ייעודיות שמטרתן ניהול קבצי נתונים באמצעי אחסון שונים. למערכות ייעודיות אלו מקובל לקרוא **מערכות לניהול קבצים** – **FMS** (File Management Systems). מערכות תוכנה אלו אפשרו לתוכניות היישום ניהול נוח יחסית של נתונים בקבצים. נסקור בקצרה את טכנולוגיית ניהול הקבצים, שקדמה לטכנולוגיית בסיסי הנתונים. מטרת הסקירה להציג כיצד המערכות פועלות ובעיקר – להבין את הבעיות העיקריות שהביאו לפיתוח טכנולוגיה חדשה יותר.

ניהול הקבצים במערכות ניהול קבצים מושתתת על שלושה מרכיבים בסיסיים: שדה, רשומה וקובץ. נתייחס גם ל״תו״, כדי להשלים את התמונה.

- **תו** (Character) – יחידת ייצוג הנתונים הבסיסית ביותר. תו יכול להכיל ספרה, אות, או סימן מיוחד.
- **שדה** (Field) – אוסף תווים בעלי משמעות לוגית כלשהי. לדוגמה, אוסף של תשע ספרות יכול לייצג מספר תעודת זהות; אוסף של 40 תווים יכול לייצג שם של סטודנט למשל. לכל שדה יש שם, טיפוס נתונים ואורך. טיפוס הנתונים (Data Type) מגדיר את סוג התווים שהשדה יכול להכיל. למשל, טיפוס Integer קובע שהשדה יכול להכיל רק מספרים שלמים בלבד.

- **רשומה** (Record) – אוסף שדות בעלי משמעות לוגית כלשהי. לדוגמה, רשומת הסטודנט יכולה להיות מורכבת מהשדות מספר סטודנט, שם סטודנט, כתובת, תאריך לידה וכד'; רשומת קורס יכולה להיות מורכבת מהשדות מספר קורס, שם קורס, מספר נקודות הזכות שהוא מעניק, סוג הקורס וכד'. הרשומה היא המנגנון שבאמצעותו ניתן לייצג ישות במערכת הקבצים.
- **קובץ** (File) – אוסף רשומות בעלות מבנה זהה שמתייחסות לישות לוגית כלשהי. לדוגמה, קובץ הסטודנטים יכיל את כל רשומות הסטודנטים; קובץ הקורסים יכיל את כל רשומות הקורסים. לכל קובץ יש שיטת ארגון (File Organization) המגדירה את צורת ארגון הרשומות בתוך הקובץ וכיצד ניתן לאתר אותן. שיטות הארגון הנפוצות הן ארגון **סדרתי** (Sequential), שבו הרשומות מופיעות זו אחר זו על פי סדר הכנסתן לקובץ; ארגון **אינדקס סדרתי** (Index Sequential), שבו הרשומות מסודרות על פי מפתח (Key) כלשהו והגישה אליהן היא באמצעות אינדקס; וארגון **אקראי** (Random) שבו המיקום היחסי של הרשומה בקובץ נקבע על פי נוסחה מתמטית כלשהי המופעלת על המפתח. הקובץ הוא המנגנון שבאמצעותו ניתן לייצג סוג ישות במערכת קבצים.

תוכנית היישום יכלה לגשת ולעדכן את הנתונים תוך שימוש בפקודות פשוטות מאוד – כגון: קרא רשומה, עדכן רשומה וכד' – מבלי שמהנדס התוכנה יהיה חשוף לכל המורכבות של ניהול הנתונים על אמצעי האחסון. נסקור את עקרונות העבודה והשלבים בעבודה של תוכנית יישום עם מערכת ניהול קבצים.

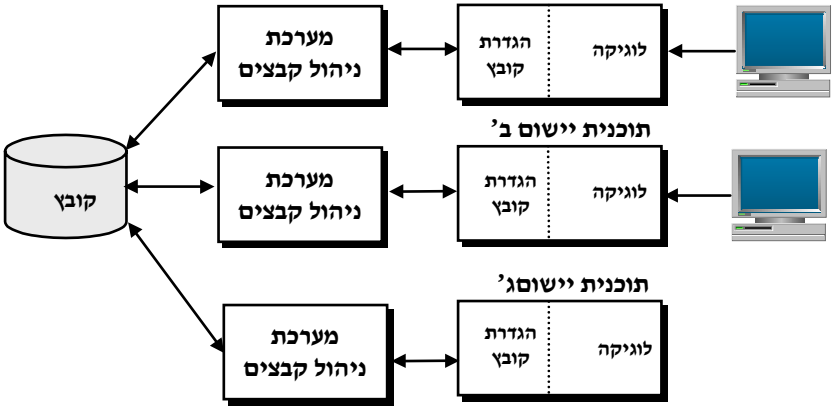
- **שלב א' – הגדרת מבנה הקובץ** (File Definition): תוכנית יישום המבקשת לגשת לקובץ כלשהו חייבת תחילה להגדיר את מבנה הקובץ, כמו השדות שיהיו ברשומה, אורך של כל שדה, טיפוס הנתונים של כל שדה ושיטת הארגון של הקובץ. אם תוכנית היישום מבקשת לגשת למספר קבצים, עליה להגדיר כל קובץ בנפרד ולתת לו שם לוגי שונה.
- **שלב ב' – כתיבת הלוגיקה העסקית** (Business Logic): מהנדס התוכנה כותב את תוכנית היישום המכילה את הלוגיקה העסקית. בכל מקום בו עליו לקבל נתונים, לעדכן נתונים וכד', הוא משתמש בפקודות היסוד של ניהול קבצים: פתח קובץ, קרא רשומה, עדכן רשומה, הוסף רשומה, בטל רשומה, סגור קובץ. התוכנית יכולה לטפל במספר קבצים שונים ולפנות לכל אחד מהם על פי השם הלוגי החד-ערכי שלו. חלק ניכר מהלוגיקה של התוכנית עוסק בגישה אל הנתונים ובטיפול בהם.
- **שלב ג' – הידור** (Compilation): תוכנית היישום עוברת תהליך הידור על ידי מהדר שפת התכנות (Compiler). המהדרים החדשים הותאמו להכיר את הגדרות מבני הקבצים ואת הפקודות המיוחדות לגישה לנתונים. המהדר מייצר את ההגדרות של שטחי זיכרון מיוחדים המשמשים לתקשורת בין תוכנית היישום לבין מערכת ניהול הקבצים.

❖ **שלב ד' – ריצה**: התוכנית מתחילה להתבצע פקודה אחר פקודה. ברגע שהתוכנית מגיעה לפקודה הקשורה לניהול הנתונים, הפיקוח מועבר למערכת ניהול הקבצים המבצעת את אוסף הפעולות הדרושות. היא מחזירה לתוכנית היישום את הרשומה המבוקשת, או לחילופין מודיעה לה שהפעולה התבצעה בהצלחה (למשל, בוטלה רשומה מהקובץ, עודכנו נתוני רשומה), או בכישלון (הרשומה המבוקשת לא נמצאה).

תרשים 2.3 מציג את מודל העבודה של תוכנית יישום הפועלת עם מערכת ניהול קבצים. תוכנית היישום אינה פונה באופן ישיר אל הקובץ אלא מפנה את בקשותיה אל המערכת לניהול קבצים. מערכת זו מבצעת את הפעולה המבוקשת ומחזירה את התוצאה. בצורה זו, מהנדס התוכנה המפתח את היישום אינו חשוף למורכבות ניהול הנתונים ביחידות האחסנה, הכולל הקצאת בלוקים על הדיסק, שליטה על מאגרי קלט/פלט בזיכרון, בניית אינדקסים ועדכונם, איתור שטחים חדשים בדיסק וכד'.

תרשים 2.3: שיטת העבודה עם מערכת ניהול קבצים.

הבעיה העיקרית במודל העבודה הזה נובעת מהעובדה שהגדרת מבנה הנתונים היא חלק בלתי נפרד מתוכנית היישום עצמה. קובץ אחד משרת מספר רב של תוכניות יישום שונות, ולכן הגדרת הקובץ נמצאת בעשרות, ולפעמים גם במאות תוכניות יישום. תרשים 2.4 מציג מצב שבו שלוש תוכניות יישום שונות (שתיים מהן מקוונות ואחת תוכנית אצווה) ניגשות לאותו קובץ ולכן – כל אחת מהן מכילה את הגדרת הקובץ. נשים לב לעובדה שכל תוכנית יישום מקבלת עותק נפרד של תוכנת המערכת לניהול קבצים.



תרשים 2.4: הגדרת קובץ מופיעה במספר תוכניות יישום.

המערכות לניהול קבצים אינן מטפלות בקשרים בין קבצים, אלא בקובץ הבודד בלבד. כדי להבהיר את הבעייתיות, נתייחס לדוגמה למערכת המינהל האקדמי שצריכה לנהל טבלאות, כגון סטודנטים, מרצים, מחלקות, קורסים וציונים. בין טבלאות אלו קיימים

קשרים שונים. לדוגמה, סטודנט לומד בקורס, קורס מוצע על ידי מחלקה, מרצה מלמד בקורס. מימוש מערכת מידע מסוג זה באמצעות מערכת לניהול קבצים מחייב ניהול חמישה קבצים שונים, שאת הקשרים ביניהם יש לממש על ידי הכנסת נתוני הקשר בתוך כל רשומה. ניהול הקשרים בין הקבצים הוא באחריות תוכניות היישום. לדוגמה, כדי לממש את הקשר מלמד בין קובץ המרצים לבין קובץ הקורסים, יש להוסיף את המפתח של קובץ המרצים, מספר המרצה, בתור שדה נוסף ברשומת הקורס.

מערכת ניהול קבצים אינה מכירה את הקשרים האלה ולכן אינה "מתייחסת" לעובדה שקיימים קשרים בין קבצי הנתונים. מכיון שהמערכת אינה מכירה את הקשרים, מבחינתה ניתן לבטל רשומה של מרצה מבלי לעדכן בהתאם את כל רשומות הקורס בהן רשום שהוא מלמד. כמובן שביצוע פעולה מסוג זה יכולה לשבש לחלוטין את אמינות הנתונים. על מהנדס התוכנה להכיר את מבנה הרשומות והשדות בשני הקבצים ואת שיטת הגישה לכל אחד מהם. וזאת, כדי שיוכל לכתוב את הוראות העיבוד הדרושות.

היעדר תמיכה בניהול קשרים בין נתונים היא רק אחת הבעיות במבנה נתונים זה. הבעיה העיקרית של מערכות ניהול הקבצים היא ברמת התמיכה הנמוכה יחסית שהיא נותנת לשינויים במבנה הנתונים. מערכות המידע חייבות להתמודד עם שינויים מתמידים בדרישות הפונקציונליות ובתהליכי עיבוד המידע. במשך מחזור החיים שלהן מתעורר צורך בנתונים חדשים, בשינויים במבנה הנתונים ובעיבודים שונים או חדשים של נתונים קיימים. שינויים בלתי פוסקים אלה במבנה הנתונים הם אחת מנקודות התורפה העיקריות במודל העבודה של המערכות לניהול קבצים.

שינוי מבנה הקובץ והתאמתו לדרישות החדשות דורש את **הוספת הנתונים החדשים לקובץ.** בשיטה זו "מרחיבים" את הרשומה הקיימת, כדי שתכיל גם את הנתונים החדשים. מבנה הרשומה משתנה, ולכן צריך לשנות את ההגדרה של המבנה שלה **בכל** תוכניות היישום הניגשות אל הקובץ שבו חל השינוי. דבר זה מחייב הידור מחדש של תוכניות היישום, אך ייתכן שיש צורך גם בשינויים מורכבים יותר, כמו שינויים בשטחי עבודה בהם התוכנית משתמשת. הבעיה המרכזית היא שצריך לשנות אפילו את תוכניות היישום שאינן זקוקות לנתונים החדשים וגם אינן מתכוונות להשתמש בהם. למרות ששינוי כזה לא נראה מסובך, הוא גורר אחריו שינויים בתוכניות יישום רבות המשתמשות בקובץ וכתוצאה מכך, צריך להשקיע העבודה רבה בביצוע השינוי. בנוסף לשינוי בתוכניות היישום יש לבצע הסבה של מבנה הקובץ הקיים למבנה החדש. תהליך זה נקרא **הסבת נתונים** (Data Conversion).

בגלל חוסר הגמישות לשינויים נמנעו מלבצע את התאמת מבנה הנתונים הקיים וניסו למצוא פתרונות עוקפים לבעיה. אחד הפתרונות המקובלים היה הקמת קובץ חדש שמכיל את הנתונים החדשים ואת המפתח של הרשומה הדרוש לזיהויה. לדוגמה, אם צריך היה להוסיף לקובץ הסטודנטים נתון חדש על הציון שהסטודנט קיבל במבחן הפסיכומטרי, בנו קובץ חדש שהכיל את המפתח מספר הסטודנט ואת הנתון החדש. בחלק מהמקרים, מתוך כוונה לחסוך בגישות לקבצים אחרים, גם הוסיפו לקובץ החדש מספר נתונים, כמו שם הסטודנט, כתובת הסטודנט ועוד. בשיטה זו אין צורך לשנות את כל התוכניות הקיימות ולהסב את מבנה הקובץ. יש צורך לבצע שינוי רק בתוכניות שזקוקות לנתון החדש. החיסרון העיקרי בשיטה זו היא **כפילות נתונים** (Data Redundancy), שעלולה

לגרום להיווצרות **שתי גרסאות של נתונים** כתוצאה מתהליכי עדכון ועיבוד נפרדים. לדוגמה, אם כתובת הסטודנט מופיעה בשני קבצים שונים, סביר להניח שעם הזמן נמצא שתי כתובות שונות בשני הקבצים.

ככל שהשימוש במערכות המידע הלך וגבר, צריך היה להשתמש בנתונים השייכים ליישומים שונים, להתבסס על קשרים מורכבים יותר בין הנתונים ולבצע מספר גדול יותר של שינויים במבנה הנתונים. עיקרון הפעולה של מערכות ניהול הקבצים, הקושי שלהן בהתמודדות עם שינויים תכופים במבנה הנתונים והיעדר השירותים התומכים הדרושים למערכות מידע גדולות ומורכבות שימשו רקע לפיתוח הדור הבא של מערכות ניהול הנתונים – טכנולוגיית בסיסי הנתונים.

הקורא המעוניין להרחיב ולהעמיק את ידיעותיו במערכות לניהול קבצים יפנה לספרו של המחבר **ארגון נתונים וקבצים** בהוצאת **הוד-עמי**.

מערכות לניהול בסיסי נתונים

ככל שדרישות המידע הפכו למורכבות יותר וצריך היה לשלב נתונים מיישומים שונים, התברר שהמערכות לניהול קבצים אינן מספקות תשובה הולמת. על הרקע הזה החלה להתפתח טכנולוגיה חדשה, **מערכות לניהול בסיסי נתונים** – **DBMS** (DataBase Management Systems).

לפני שנגדיר מהי מערכת לניהול בסיסי נתונים, נגדיר תחילה את המושג **בסיס נתונים**.

בסיס נתונים **Database**	***בסיס נתונים*** **מוגדר כאוסף של טבלאות הקשורות ביניהן בקשרים לוגיים כלשהם, המאפשר את שיתוף הנתונים בין היישומים השונים. בנוסף לטבלאות המכילות את הנתונים, מכיל בסיס הנתונים גם אוסף טבלאות המתארות את מבנה בסיס הנתונים עצמו.**

מהגדרה זו ניתן לראות שבסיס הנתונים הינו מבנה מתקדם יותר, כי הוא מתייחס לאוסף של טבלאות שונות ולקשרים ביניהן. מערכת ניהול הקבצים ממוקדת בניהול הקובץ הבודד. היא אמנם מסוגלת לנהל מספר גדול מאוד של קבצים, אולם היא מתייחסת אל כל קובץ כאל יחידה עצמאית. לעומת גישה זו, בסיס הנתונים מנהל אוסף של **טבלאות** ושל **הקשרים** ביניהן שעשויים להופיע בצורות שונות.

בסיס הנתונים הינו רמה גבוהה יותר של מבנה נתונים המתייחס לכל הטבלאות ולכל הקשרים ביניהן. בסיס נתונים אחד, אין פירושו שכל הטבלאות מאוחסנות **פיסית** בקובץ אחד. בדרך כלל, כל טבלה מאוחסנת בקובץ פיסי נפרד גם במסגרת בסיס הנתונים, אולם ניתן לאחסן מספר טבלאות שונות באותו קובץ ואפילו את כל הטבלאות ניתן לאחסן בקובץ פיסי אחד. כלומר, קיים נתק בין המושג הלוגי "טבלה" – סטודנטים, מרצים, קורסים – לבין המושג "קובץ" שהינו מושג פיסי.

כדי להשיג את אי התלות בין הנתונים המנוהלים בבסיס הנתונים לבין תוכניות היישום המשתמשות בהם, מכיל בסיס הנתונים בנוסף לנתונים עצמם גם את התיאור שלהם. מסיבה זו מקובל לומר שבסיס הנתונים הוא אוסף נתונים **המתאר את עצמו** (Self Describing). כפי שראינו, תיאור הנתונים במערכות לניהול קבצים היה חלק מתוכניות היישום ואילו כאן – בסיס הנתונים כולל גם את תיאור הנתונים.

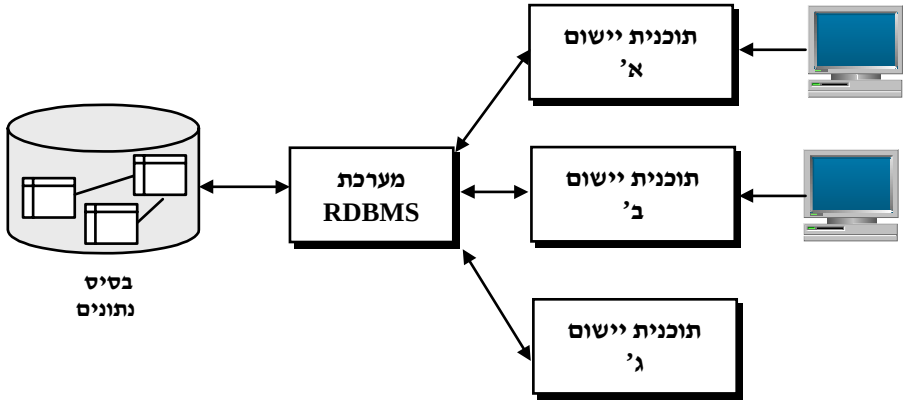
מערכת לניהול בסיסי נתונים טבלאיים **Relational DataBase Management System - RDBMS**	***מערכת לניהול בסיסי נתונים טבלאיים* היא מערכת תוכנה ייעודית המאפשרת את ניהול בסיס הנתונים תוך תמיכה במיגוון רחב של שירותים, כגון אפשרות גישה בו-זמנית למספר רב של יישומים שונים ושירותי בקרת גישה נוספים.**

מערכת RDBMS היא תוכנה ייעודית המאפשרת לתוכניות יישום לגשת בנוחות וביעילות אל הטבלאות ולבצע את התנועות הדרושות. מערכת זו תומכת בשלוש פונקציות בסיסיות:

- **הגדרת הנתונים** (Data Definition): הגדרת מבנה הנתונים הן ברמה הלוגית והן ברמה הפיסית. הגדרת הנתונים מתבצעת מחוץ לתוכניות היישום ומאפשרת את הקבצת הנתונים למבנה לוגי בעל משמעות – **הטבלה**.
- **טיפול בנתונים** (Data Manipulation): גישה אל בסיס הנתונים מתוך תוכניות היישום או באופן ישיר על ידי המשתמשים השונים, ובכלל זה גם שליפת נתונים, עדכון טבלאות, הוספת שורות חדשות וביטול שורות קיימות.
- **אילוצי אמינות** (Integrity Constraints): הגדרת אוסף של אילוצים שמטרתם להבטיח את אמינות בסיס הנתונים בכל נקודת זמן. חלק מהאילוצים הם ברמת הטבלה הבודדת, חלק מהם מתייחסים לקשרים בין טבלאות וחלקם עוסקים ברמת הארגון.

ההבדל העיקרי בין אוסף קבצים המנוהלים על ידי מערכת קבצים לבין אוסף הטבלאות המנוהלות על ידי מערכת לניהול בסיסי נתונים, טמון בעובדה שמערכת RDBMS "רואה" את **כל הטבלאות כאוסף אינטגרטיבי עם קשרים**, ולכן היא יכולה לטפל בניהול הקשרים ואמינות הנתונים.

מערכת RDBMS מאפשרת את שילוב הנתונים בבסיס נתונים אינטגרטיבי, תוך מתן אפשרות לגישה מהירה לנתונים במיגוון רחב של חתכים. מכיון שבסיס הנתונים יכול לשרת מספר רב של יישומים שונים, ניתן לצמצם מאוד את כפילות הנתונים, להגדיל את אמינות הנתונים ולשפר את זמינותם, כדי להיענות לצרכים המשתנים של דרישות המידע.



תרשים 2.5: מערכת RDBMS אחת משרתת את כל תוכניות היישום.

לעומת מודל העבודה של מערכת לניהול קבצים, מודל העבודה של מערכת RDBMS שונה ומתוחכם בהרבה. מערכת RDBMS היא מרכזית ומנהלת את כל הפניות אל בסיס הנתונים. בנוסף לניהול הנתונים בבסיס הנתונים, המערכת מכירה ומנהלת את כל הקשרים בין הטבלאות השונות. בנוסף לניהול בסיס הנתונים, מספקת מערכת RDBMS מיגוון רחב של שירותים, הדרושים לבניית מערכות מידע מורכבות המנהלות מספר גדול של טבלאות הקשורות ביניהן בקשרים שונים ומשרתות בו-זמנית מספר רב של משתמשים המחוברים באמצעות רשת תקשורת כלשהי אל המחשב.

להבדיל ממערכות ניהול הקבצים, מערכת RDBMS מנהלת את כל ההגדרות של מבנה הנתונים באופן מרכזי ובלתי תלוי בתוכניות היישום. שיטה זו מספקת רמת אי-תלות גבוהה יותר בין תוכניות היישום לבין מבנה בסיס הנתונים. על כן ניתן לבצע שינויים מסוימים בבסיס הנתונים ללא כל השפעה על תוכניות היישום.

בסיס הנתונים הוא מאגר נתונים אינטגרטיבי שמנהל את אחד המשאבים החשובים של הארגון ומכאן גם החשיבות הרבה בבנייתו. ניתן לומר שהקמת מאגר מרכזי תורמת לכך שהנתונים יוצאים מ'בעלות' תוכניות היישום ועוברים לפונקציה מרכזית האחראית על מאגר הנתונים, אל **מנהל בסיס הנתונים** – **DBA** (**DataBase Administrator**). בהמשך נפרט את תפקידיו של מנהל בסיס הנתונים. תהליך הקמת המאגר המרכזי יכול להיות מורכב מאוד, ועל כן הוא מחייב מיומנות ושיתוף פעולה בין קבוצות שונות בארגון. כישלון ביצירת שיתוף זה עלולה לגרום לכישלון בהקמת יישומים הפועלים בסביבת מערכת ניהול בסיס הנתונים, או לגרום לכך שהפוטנציאל הטמון במאגר מרכזי לא ימומש במלואו. הצלחת ההקמה של בסיס נתונים מרכזי מחייבת תיאום מתמיד בין משתמשים שונים ובקרה מרכזית על הנתונים.

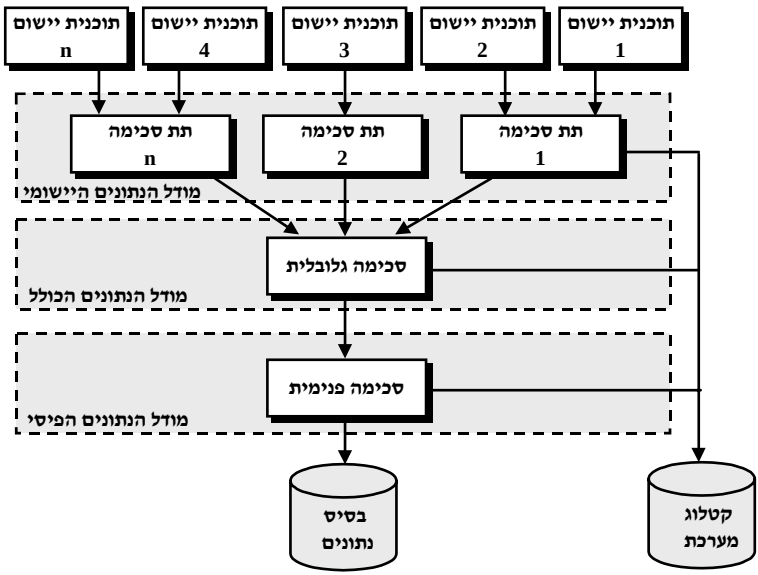
מודל העבודה של מערכת RDBMS

בשנת 1975 פרסם מכון התקנים האמריקאי, **ANSI** (American National Standards Institute), במסגרת אחת הוועדות המקצועיות, ארכיטקטורה כללית של מערכת לניהול בסיס נתונים. מודל זה נודע בשלב מאוחר יותר כמודל ANSI/SPARC/X3 על שם קבוצת העבודה שעיצבה אותו, ובקיצור – מודל ANSI/SPARC.

למרות הזמן הרב שעבר מאז פרסום המודל, העקרונות והמרכיבים של מערכת לניהול בסיסי נתונים נשארו תקפים. למרות שהמודל לא מיושם במלואו ברוב המערכות המסחריות, הוא מהווה מסגרת התייחסות נוחה. המודל הוצג בתקופה שבה מערכות RDBMS היו במעבדות הפיתוח, ולכן רוב המושגים שלו תאמו יותר את המערכות הוותיקות ובעיקר – את המערכות המבוססות על המודל הרשתי. למרות זאת, בחרנו להציג את הארכיטקטורה, תוך התאמת המושגים למערכות RDBMS.

מודל ANSI/SPARC הציג לראשונה ובאופן פורמלי את **רמות ההפשטה** (Abstraction Levels) כפתרון לבעיית תלות תוכניות היישום והמשתמשים במבני הנתונים המשמשים אותם.

בתרשים 2.6 ניתן לראות מודל המכיל חמש שכבות שונות. בשכבה הנמוכה ביותר נמצא בסיס הנתונים וקטלוג המערכת. הקטלוג מכיל את כל ההגדרות ובסיס הנתונים מכיל את אוסף כל טבלאות הנתונים ואת הקשרים ביניהן. הקטלוג ובסיס הנתונים מאוחסנים ביחידות אחסנה כלשהן (בדרך כלל דיסקים מגנטיים). בשכבה העליונה ביותר נמצאות תוכניות היישום או המשתמשים, הפועלים במקוון או באצווה עם בסיס הנתונים. שלוש שכבות ה"בידוד", או רמות ההפשטה, שבין תוכנית היישום לבין בסיס הנתונים, הן המייחדות מערכות RDBMS לעומת מערכות ניהול קבצים.



תרשים 2.6: מודל כללי של מערכת RDBMS.

המטרה העיקרית של רמות ההפשטה היא **לבודד** את תוכנית היישום ממבנה הנתונים, כדי לאפשר גמישות רבה יותר בביצוע שינויים במבנה הנתונים, ללא צורך בשינוי תוכניות היישום. נדגיש, שההשקעה הגדולה של הארגון בפיתוח מערכות המידע מתבטאת בתוכניות היישום. מערכת מידע בינונית יכולה להכיל עשרות ומאות תוכניות כאלו. אנו מכירים בכך ששינויים חלים כל הזמן במבנה הנתונים כמו לדוגמה, העברת טבלה לדיסק מהיר וחדש, הוספת אינדקס חדש לטבלה, הוספת עמודה חדשה לטבלה קיימת, שינוי ההגדרה של עמודה מסוימת או הוספת טבלה חדשה. אם כל שינוי היה מחייב שינוי מקביל בתוכניות היישום, הדבר היה מאט באופן ניכר את יכולת מערכות המידע להתאים את עצמן לצרכים המשתנים של הארגון.

נרחיב במקצת את הדיון ברמות ההפשטה המרכיבות את מודל ANSI/SPARC ונסקור את השכבות השונות.

סכימה גלובלית (Global Schema)

הסכימה הגלובלית **Global Schema**	***הסכימה הגלובלית*, שלפעמים מקובל לכנות אותה גם בשם Conceptual Schema או בקיצור Schema, הינה התיאור הפורמלי והמלא של כל הנתונים המנוהלים בבסיס הנתונים.**

הסכימה מתארת את כל האובייקטים המנוהלים בבסיס הנתונים: את כל הטבלאות, את העמודות בכל טבלה, את טיפוס הנתונים של כל עמודה, את הקשרים הלוגיים בין הטבלאות, את האילוצים שעל הערכים בטבלה לקיים ועוד. לכל הטבלאות והעמודות ניתנים במסגרת הסכימה שמות סימבוליים המאפשרים למשתמשים לפנות אליהם תוך שימוש בשם בעל משמעות. למשל, לטבלה המכילה את נתוני הסטודנטים ניתן שם סימבולי STUDENTS, ולעמודה המכילה את מספר הסטודנט ניתן שם סימבולי STUDENT_ID (אין חשיבות לכתיבה באותיות רישיות או רגילות, אך יש להיות עקביים).

הסכימה הינה אחד המאפיינים העיקריים של כל מערכת לניהול בסיסי נתונים, כי היא מאפשרת להגדיר את בסיס הנתונים באופן **חיצוני** לכל תוכניות היישום שמשתמשות בנתונים. נזכיר כי מודל העבודה של מערכות ניהול הקבצים מתבסס על הגדרת הנתונים באופן **פנימי** בכל תוכנית יישום. במצב זה, כל שינוי במבנה הקובץ, גם אם אינו נוגע לתוכנית היישום, מחייב שינוי מתאים ומיידי בהגדרת הנתונים הנמצאת בתוך תוכנית היישום. לעומת מודל זה, מודל העבודה במערכות RDBMS מתבסס על הגדרת הנתונים באופן חיצוני לתוכניות היישום, ונמצא בסכימה הגלובלית. במצב זה, לא כל שינוי מחייב שינוי מקביל בתוכניות היישום.

שפת הגדרת נתונים **Data Definition Language - DDL**	***שפת הגדרת נתונים* הינה שפה מיוחדת המאפשרת למנהל בסיס הנתונים להגדיר את המרכיבים השונים של בסיס הנתונים - טבלאות, עמודות, מפתחות, אילוצים וכד'.**

כל מערכת RDBMS מספקת אוסף של פקודות מיוחדות להגדרת הסכימה הגלובלית. בין השאר, השפה מאפשרת למנהל בסיס הנתונים:

- להגדיר את כל **הטבלאות** המנוהלות בבסיס הנתונים. לכל טבלה יש לתת **שם חד-משמעי**. לדוגמה, ניתן להגדיר שבסיס הנתונים יהיה מורכב משלוש הטבלאות האלו: טבלת הסטודנטים, טבלת הקורסים וטבלת הציונים.
- להגדיר לכל טבלה את כל **העמודות** שלה. לדוגמה, ניתן להגדיר שטבלת הסטודנטים תכיל את העמודות מספר הסטודנט, שם הסטודנט ו- עיר מגורים.
- להגדיר את **טיפוס נתונים** (Data Type) לכל עמודה. כלומר, האם היא יכולה להכיל רק ערכים נומריים או גם אלפביתיים, מה אורך עמודת כתובת הסטודנט (למשל, 30 תווים) וכד'.
- להגדיר את **המפתח העיקרי** (Primary Key) של כל טבלה. לדוגמה ניתן להגדיר שהמפתח של טבלת הסטודנטים הוא מספר סטודנט. ומכיון שכך, המערכת צריכה לוודא שלא יהיו בטבלת הסטודנטים שתי שורות המכילות את אותו ערך בעמודה מספר סטודנט.
- להגדיר **תחומי ערכים מותרים** של הערכים בעמודות הטבלה. הגדרת תחומי ערכים אלה בסכימה הגלובלית משחררת את תוכניות היישום מן הבדיקה הזו, כי המערכת תבצע אותה באופן מרכזי ואוטומטי. כאשר צריך לשנות את תחום הערכים המותר, ניתן לבצע את השינוי באופן מרכזי ובמקום אחד בלבד.
- להגדיר את **הקשרים הלוגיים** בין הטבלאות. לדוגמה, ניתן להגדיר קשר בין טבלת הסטודנטים וטבלת הציונים שיבטיח שלא ניתן יהיה להוסיף שורה לטבלת הציונים עבור סטודנט שלא קיים בטבלת הסטודנטים. הגדרת קשר זה בין טבלת הסטודנטים וטבלת הציונים יבטיח גם שאם מבטלים שורת סטודנט, המערכת תבטל גם את כל השורות של ציוניו.
- להגדיר את **כללי ההגנה** על הטבלאות והנתונים שבבסיס הנתונים. למשל משתמש ששמו "דן" יוכל לקרוא את כל שורות הציונים שקיבל בלבד, אך הוא לא יוכל לקרוא נתונים אחרים ובוודאי שלא יוכל לעדכן דבר, גם לא את ציוניו.
- הגדרת **פרוצדורות בסיס** נתונים (Database Procedures) **ומזניקים** (Triggers) המופעלים בתנאים מסוימים ומבצעים פעולות מסוימות.

כל ההגדרות האלו עוברות בדיקות תקינות ונרשמות בקטלוג המערכת. נציג כאן דוגמה קצרה לבניית בסיס נתונים המכיל שתי טבלאות. הסבר מלא ומדויק של כל פקודות DDL של מערכות RDBMS ניתן בפרק 9.

```
1.  CREATE TABLE STUDENTS
2.              (STUDENT_ID  CHAR ( 5) NOT NULL,
3.               NAME            CHAR ( 30),
4.               CITY            CHAR ( 40),
5.  PRIMARY KEY (STUDENT_ID))
6.
7.  CREATE TABLE GRADES
8.              (STUDENT_ID CHAR ( 5)  NOT NULL,
9.               COURSE_ID    CHAR ( 5) NOT NULL,
10.              SEMESTER     CHAR ( 6) NOT NULL,
11.              TERM         CHAR ( 1) DEFAULT 'A'  CHECK (VALUE IN ('A', 'B', 'C'),
12.              GRADE        SMALLINT CHECK (VALUE BETWEEN  0 AND 100),
13.              GRADE_SEM  SMALLINT CHECK (VALUE BETWEEN  0 AND 100)
14. PRIMARY KEY  (STUDENT_ID, COURSE_ID, SEMESTER, TERM)
15. FOREIGN KEY   (STUDENT_ID) REFERENCES STUDENTS
16.                                ON DELETE NO ACTION
```

הסבר:

- שורות 1 עד 5 מגדירות את טבלת **סטודנטים** ושורות 7 עד 16 מגדירות את טבלת **ציונים**.
- שורות 2 עד 4 מגדירות את שמות העמודות של טבלת **סטודנטים** ושורות 8 עד 13 מגדירות את שמות העמודות של טבלת **ציונים**. נשים לב שלכל עמודה קובעים את טיפוס הנתון, את אורכו ובמידת הצורך גם את האילוצים. למשל, בשורה 8 האילוץ NOT NULL קובע שעמודה זו חייבת להכיל ערך כלשהו. בשורה 12 מופיע אילוץ הקובע שהציון חייב להיות מספר בין 0 ל-100.
- שורה 5 מגדירה את המפתח העיקרי של טבלת **סטודנטים** ושורה 14 מגדירה את המפתח העיקרי של טבלת **ציונים**.
- שורות 15 ו-16 מגדירות את הקשר בין שתי הטבלאות, כך שלא ניתן להוסיף ציון לסטודנט שאינו מופיע בטבלת **סטודנטים** או לבטל שורת סטודנט שיש לו ציונים.

תת-סכימה (View)

תת-סכימה **View/Sub Schema**	***תת-סכימה*** **מגדירה חלק מבסיס הנתונים על פי הצרכים המיוחדים של משתמש מסוים, או של תוכנית יישום מסוימת.**

בדרך כלל, משתמש או תוכנית יישום זקוקים רק לחלק קטן מכלל הנתונים המנוהלים בבסיס הנתונים. אם לצורך כל פנייה היינו חושפים את המשתמש לכל ההגדרות שבסכימה, היינו מסתכנים בבלבול, בקושי מיותר ובבעיות אבטחת הנתונים.

מנגנון **תת-הסכימה** (Sub-Schema) מאפשר ליצור לכל משתמש **נקודת מבט** (View) יחודית המותאמת לצרכיו. בגלל היכולת ליצור אשליה של טבלה שאינה קיימת בפועל בבסיס הנתונים, מקובל לקרוא לתת-הסכימות גם בשם **טבלאות מדומות**. קיים הבדל משמעותי בין הטבלאות המדומות במערכות RDBMS לבין מנגנון תת-הסכימה במודל ANSI/SPARC. במודל ANSI/SPARC כל תוכנית משתמשת בתת-סכימה אחת המכילה את כל הנתונים שהתוכנית צריכה. הגדרות תת-הסכימה פשוטות יחסית, ולמעשה מכילות תת-קבוצה של כל הגדרות הסכימה. לעומת זאת, מנגנון הטבלה המדומה מתוחכם בהרבה ותוכנית יישום אחת יכולה לפנות למספר בלתי מוגבל של טבלאות ממשיות ומדומות. טבלה מדומה יכולה להכיל בעצמה פנייה לטבלה מדומה אחרת ועוד.

לדוגמה, ניתן להשתמש במנגנון הטבלה המדומה כאשר שמות העמודות שנקבעו בסכימה אינם נוחים למשתמש, כאשר הוא אינו זקוק לכל העמודות או השורות המופיעות בטבלה מסוימת, או כשרוצים להציג טבלה המורכבת משתי טבלאות ממשיות וכד'.

מערכת RDBMS מאפשרת ניהול מספר בלתי מוגבל של טבלאות מדומות, שכולן מבוססות על אותה סכימה, או על טבלאות מדומות אחרות.

בדומה לשפה להגדרת הסכימה, שפת DDL מכילה פקודה מיוחדת לבניית טבלאות מדומות מתוך טבלאות ממשיות. הפקודה CREATE VIEW מאפשרת למנהל בסיס הנתונים:

- לבנות טבלה מדומה מתוך טבלה ממשית המוגדרת בסכימה הגלובלית, כך שהטבלה המדומה תכיל רק חלק מהעמודות או מהשורות. לדוגמה, תוכנית יישום שצריכה להדפיס את שמות כל הסטודנטים אינה זקוקה, ואולי גם אינה מורשית, לראות את נתוני הציונים. במצב זה, תוכנית היישום תורשה לגשת רק אל הטבלה המדומה החדשה שנבנתה עבורה.
- לשנות את השמות הסימבוליים של הטבלאות והעמודות על פי נוחות תוכנית היישום.
- להגדיר כללי הרשאות, כך שמשתמש מסוים יכול לראות רק חלק מהשורות המופיעות בטבלה. לדוגמה, רק את הציונים שהסטודנט קיבל בקורס מסוים.
- הגדרת **עמודות מדומות** (Virtual Columns) או **עמודות מחושבות** (Computed Columns), שאינן מופיעות באופן ממשי בבסיס הנתונים. לדוגמה, נוכל לבנות טבלה מדומה המכילה עבור כל סטודנט את הציון הממוצע של כל ציוניו. עמודה כזו אינה קיימת בבסיס הנתונים, אלא מחושבת רק לצורך הצגה או הדפסה.

לדוגמה, ניצור תת-סכימה המכילה רק את מספר הסטודנט ואת שם הסטודנט, ובאותה עת נשנה את השמות הסימבוליים.

```
1. CREATE VIEW STUDENT_NAMES (STUDENT_NUMBER, STUDENT_NAME) AS
2.        SELECT   STUDENT_ID, NAME
3.        FROM     STUDENTS
```

הסבר:

- שורה 1 מגדירה שם סימבולי לטבלה המדומה החדשה STUDENT_NAMES ואת השמות החדשים של העמודות.
- שורות 2 ו-3 מגדירות שאילתה הבונה את הטבלה המדומה מתוך טבלה ממשית ששמה STUDENTS המוגדרת בסכימה הגלובלית.

נשים לב שהשאילתה שולפת רק שתי עמודות מתוך שלוש העמודות של הטבלה ומגדירה להן שמות סימבוליים חדשים. לדוגמה, שם העמודה STUDENT_ID יוחלף בשם STUDENT_NUMBER.

טבלאות מדומות הן נושא בעל חשיבות רבה במערכות RDBMS ולכן נדון בו בהרחבה בפרק 10.

סכימה פנימית (Physical Schema)

סכימה פיסית **Physical Schema**	***סכימה פיסית*, שלפעמים מקובל לקרוא לה גם בשם או *סכימה פנימית* (Internal Schema), היא תיאור המבנה הפיסי של הטבלאות כפי שהן מאוחסנות ביחידות האחסנה השונות.**

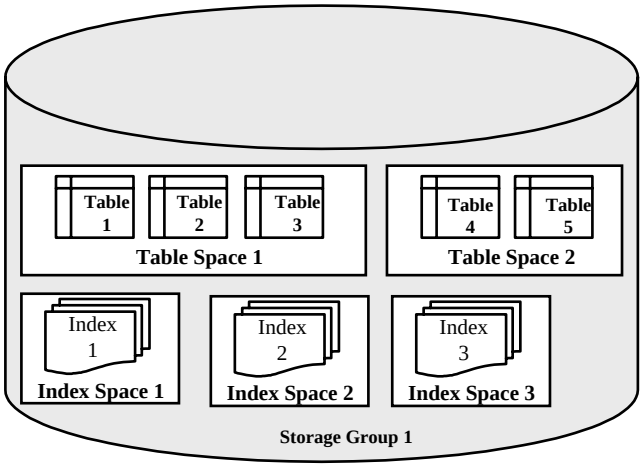
הסכימה הפנימית מכילה מיגוון פרטים הנוגעים לצורת האחסון של הטבלאות, לשיטות הגישה לשורות של הטבלה (על ידי אינדקסים, או על ידי מנגנון Hashing למשל), מספר השורות בגוש (Block) אחד, השטח הפיסי שבו הטבלה מאוחסנת, סוג יחידת האחסנה של הטבלה וכד'.

מטרת שכבה זו לאפשר שינויים הקשורים למבנה הפיסי של הנתונים, מבלי להשפיע על השכבות שמעליה. לדוגמה, החלטה של מנהל בסיס הנתונים להוסיף אינדקס חדש לטבלה מסוימת, אינה משנה את התכולה הלוגית של בסיס הנתונים, אלא יכולה להשפיע רק על ביצועי תוכניות היישום. באופן דומה, העברת טבלה מסוימת משטח פיסי מסוים לשטח פיסי אחר הדורש גודל גוש שונה, אינה משפיעה על התכולה הלוגית של הטבלה, אלא רק על ביצועי המערכת.

בכדי להדגים את מושגי הסכימה הפנימית, נתאר בקצרה את המבנה הפיסי של מערכת DB2 של חברת יבמ. מבנה זה מורכב מהאובייקטים הבאים:

- **Table Space** – **אזור הטבלה** הוא שטח דיסק המכיל טבלה אחת או יותר. השטח מורכב מאוסף דפים (pages) בדיסק שניתן להגדילו או לצמצמו באופן דינמי. בדרך כלל דף אחד הוא בגודל 4KB או 32KB. כל הדפים בתוך אותו שטח זהים בגודלם.
- **Index Space** – **אזור האינדקס** הוא שטח דיסק, המכיל אינדקס אחד. בדומה לשטח הטבלאות, גם שטח זה מורכב מאוסף דינמי של דפים. כל דף הוא בגודל 4KB.

❖ **Storage Group** – **קבוצת אחסנה** היא אוסף של כל יחידות האחסנה מאותו סוג (למשל דיסקים מגנטיים או דיסקים אופטיים). כל אזור, הן זה שמוקצה לטבלאות והן זה שמוקצה לאינדקסים, קשור לקבוצת אחסנה אחת בלבד. כאשר צריך להגדיל את אזור הטבלאות או את אזור האינדקסים, המערכת תבקש להקצות דפים נוספים מאותה קבוצת אחסנה בלבד.



תרשים 2.7: מבנה פיסי של בסיס נתונים.

מערכות RDBMS מכילות אוסף פקודות מיוחד להגדרת המבנה הפיסי של בסיס הנתונים. לרשות מנהל בסיס הנתונים עומד אוסף פקודות המאפשרות לו להקצות שטחים חדשים, להגדיר איזו טבלה תאוחסן באיזה אזור, להגדיר אזורי אינדקסים וכד׳. הפקודות הן לשימושו הבלעדי של מנהל בסיס הנתונים. מכיון שפקודות אלו שונות ממערכת למערכת ואפילו באותה מערכת קיים שוני בהתאם למחשב שבסיס הנתונים מופעל בו, לא נתאר פקודות אלו.

פקודה אחת שדומה בכל המערכות היא פקודת DDL להגדרת אינדקס חדש לטבלה, ולכן נציג דוגמה שלה:

CREATE UNIQUE INDEX ON TABLE STUDENT (STUDENT_ID)

המערכת תבנה אינדקס לטבלת סטודנטים על פי העמודה STUDENT_ID ותוודא שלא יהיו שני סטודנטים בעלי אותו מספר סטודנט. מנהל בסיס הנתונים צריך להגדיר גם באיזה אזור אינדקסים יש לבנות את האינדקס החדש.

בשנים הראשונות של טכנולוגיית בסיסי הנתונים, התפתחו מערכות מסחריות שונות שלא הכילו הפרדה ברורה בין המבנה הפיסי לבין המבנה הלוגי. טענת המפתחים היתה שהפרדה כזו תשפיע לרעה על ביצועי המערכת. כיום, כל מערכות RDBMS המסחריות מקיימות הפרדה ברורה מאוד בין המבנה הפיסי לבין המבנה הלוגי. הן אינן מאפשרות לתוכניות היישום לראות את המבנה הפיסי ולהתייחס אליו כלל, וכך הן מאפשרות **אי-תלות** בין תוכניות היישום לבין מבנה הנתונים. לדוגמה, הוספת עמודה חדשה לטבלה קיימת, העברה של טבלה מאזור אחד למשנהו, הוספת דפים חדשים לטבלה בקבוצת

אחסנה אחרת, הוספת אינדקס באזור אינדקס חדש – פעולות אלו מבוצעות על ידי מנהל בסיס הנתונים, מבלי שתהיה לכך השפעה כלשהי על תוכניות היישום.

בסיס הנתונים (Database)

הנתונים בכל מערכת ממוחשבת מנוהלים בקבצים ומאוחסנים ביחידות אחסנה (Storage Devices) כלשהן כמו דיסק מגנטי, קלטת, או דיסק אופטי. מבנה הנתונים, כפי שהוא מנוהל ביחידות האחסנה, הינו **המבנה הפיסי** של הנתונים. בסופו של דבר, הנתונים מיוצגים על ידי אוסף של סיביות שמאוחסנות ביחידות האחסנה. מבנה זה מוכתב על ידי תכונות אמצעי האחסון, ולאו דווקא על ידי הדרישות הלוגיות של תוכניות היישום. לדוגמה, הניהול של נתוני סטודנט בדיסק קשיח, דיסקט, דיסק אופטי או סרט מגנטי שונה ברמה הפיסית, למרות שמבחינה לוגית זו בדיוק אותה טבלה.

ברמה הפיסית מתייחסים למושגים כגון סיביות, מסילות הדיסק (Tracks), גושים (Blocks) של סיביות, אינדקסים, מצביעים (Pointers) ועוד. מי שהתנסה פעם בכתיבת תוכניות הפונות ישירות אל יחידות האחסנה יודע שזהו נושא מורכב ומייגע מאוד. כתיבת תוכניות יישום היתה הופכת לתהליך מורכב ואיטי ביותר, לו מהנדס התוכנה היה צריך להכיר עולם מורכב זה ולטפל באופן ישיר במושגים אלה. מכאן נבע הצורך הברור לבודד את תוכניות היישום מהמורכבות של מבנה הנתונים הפיסי, ולאפשר להן להשתמש **במבנה לוגי** כלשהו, שיסתיר את המורכבות הזאת. המבנה הלוגי המוצג לתוכניות היישום הוא **הטבלה**. תוכנית היישום תוכל לקרוא את שורות הטבלה, להוסיף לה שורות, או לבטל שורות – וכל זאת, מבלי להיות חשופה לצורת ניהול הטבלה ביחידות האחסנה.

קטלוג המערכת (System Catalog)

קטלוג המערכת **System Catalog**	***קטלוג המערכת* הוא קובץ מערכת (System File) מיוחד המנוהל על ידי מערכת RDBMS ומכיל את כל ההגדרות של הסכימות השונות.**

הסכימות השונות המנוהלות בתוך קטלוג המערכת כוללות את ההגדרה הכוללת של כל הנתונים המנוהלים בבסיס הנתונים, ולכן הקטלוג עצמו הוא מקור מידע חשוב. על כן, מקובל לקרוא למידע המנוהל בקטלוג המערכת בשם Meta Data, כלומר "מידע על הנתונים". שאלות כמו אלו יכולות לקבל מענה מתוך הקטלוג: אילו טבלאות מנוהלות בבסיס הנתונים? אילו עמודות מנוהלות בטבלה מסוימת? מה טיפוס הנתון של סוג הקורס בבסיס הנתונים? איזה אילוצים הוכתבו על ידי מעצבי המערכת (לדוגמה, סוג הקורס יכול להיות קוד בין 1 ל-4)? אילו טבלאות מדומות מנוהלות במערכת? מה השאילתה שמגדירה טבלה מדומה? איזה קשרים קיימים בין הטבלאות השונות? ועוד.

ניתן לומר שמעבר לחשיבות הגדולה של הגדרת מבנה הנתונים באופן חיצוני לתוכניות היישום, קטלוג המערכת הייחודי למערכות ניהול בסיסי נתונים, משמש מאגר מידע בעל חשיבות רבה מאוד לארגון. עד להופעת טכנולוגיה זו, כל הגדרות הנתונים היו בתוך תוכניות היישום והיה קושי רב לקבל מידע מסודר על מבנה הנתונים.

פרק 14 מוקדש לדיון מעמיק יותר ב**קטלוג המערכת**.

תוכניות יישום (Application Programs)

כל המבנה שתארנו עד כאן בא לשרת בסופו של דבר את תוכניות היישום. תוכניות אלו ניגשות ומעדכנות את הטבלאות בהתאם לדרישות הפונקציונליות של היישום. למעשה, הן מהוות את מימוש המודל הפונקציונלי של המציאות. כלומר, הן מייצגות את האירועים ואת זרימת הנתונים בארגון. לדוגמה, הן אלו המבצעות תנועה כגון קבלת סטודנט חדש למכללה שתתבטא בסופו של דבר בהוספת שורה חדשה לטבלת סטודנטים. תנועה של הודעה על שינוי כתובתו תתבטא בסופו של דבר בעדכון העמודה המתאימה בטבלת סטודנטים.

תוכניות היישום המבקשות לגשת אל הנתונים המנוהלים בטבלאות, חייבות להשתמש באוסף מיוחד של פקודות לניהול וטיפול בנתונים.

שפת טיפול בנתונים **Data Manipulation Language - DML**	***שפת טיפול בנתונים* היא שפה מיוחדת המאפשרת למהנדסי תוכנה ולמשתמשים שונים לגשת אל בסיס הנתונים, לאחזר ממנו נתונים ולשנות את תוכן הנתונים על פי הצורך.**

השפה לטיפול בנתונים במערכות RDBMS נקראת SQL ובאמצעותה יכולה תוכנית היישום לאחזר שורות מתוך טבלאות, לעדכן שורות, לבטל שורות ולהוסיף שורות חדשות. SQL אינה שפת תכנות מלאה, כי אינה מאפשרת לבצע את כל הלוגיקה הדרושה בתוכנית יישום. יעודה לאפשר לתוכנית היישום לאחזר נתונים מבסיס הנתונים ולעדכן אותם בצורה הנוחה ביותר, מבלי להתייגע בכל פרטי הניווט בין הטבלאות השונות ובתוך מבנה הנתונים הפיסי כפי שבא לידי ביטוי ביחידות האחסנה. קיימות מספר שיטות להפעלת פקודות SQL מתוך שפת התכנות שבה נכתבת תוכנית היישום. שיטה אחת, המאפשרת שיבוץ פקודות SQL בתוך שפת התכנות המארחת, נקראת Embedded SQL (SQL מוטבע, או משובץ). שיטה אחרת מאפשרת פנייה לממשק תכנות (API) מיוחד המסופק על ידי יצרן מערכת RDBMS ודרכו ניתן להפעיל את כל שירותי המערכת.

להדגמה נשתמש בשיטת שיבוץ של פקודות SQL בתוך שפת C המשמשת כאן כשפת תכנות מארחת. פקודות שפת הטיפול בנתונים (DML) אינן מוכרות למהדר של שפת C ולכן דרוש **קדם-מהדר** שיודע לתרגם את פקודות SQL המיוחדות, לאוסף של פקודות CALL אשר מפעילות את הפרוצדורות המתאימות.

לדוגמה, כדי לאחזר מטבלת סטודנטים את שם הסטודנט שמספרו מוזן על ידי המשתמש במסוף, יש לכתוב את תוכנית היישום הזו:

```
1.      EXEC SQL BEGIN DECLARE SECTION;
2.      varchar  userid (20);
3.      varchar  password (20);
4.      varchar  student_number (5);
5.      varchar  student_name (20);
6.      EXEC SQL END DECLARE SECTION;
7.
8.   main()
9.   {
10.      printf ("\n Enter Student Number: ");
11.      scanf ("%s", student_number);
12.
13.      EXEC SQL SELECT NAME INTO :student_name
14.                   FROM     STUDENTS
15.                   WHERE  STUDENT_ID = :student_number;
16. }
```

שיטות שיבוץ פקודות שפת SQL בתוכניות יישום, מובאות בפרק 12 בהרחבה, ולכן לא נסביר בשלב זה את התוכנית המוצגת.

תפקידים שונים בסביבת בסיס נתונים

בסביבת בסיס הנתונים פועלים בעלי תפקידים שונים. הבה נתאר אותם.

מנהל בסיס הנתונים

בגישה המסורתית היתה האחריות לניהול נתונים נתונה בידי היישומים השונים. מנהל היישום שמר וניהל בהתאם לצרכיו את הנתונים שנדרשו לו בתהליכי העיבוד השונים. הגידול המתמיד במורכבות ובגודל היישומים הציב בעיות שליטה קשות למנהלי היישומים. הופעת טכנולוגיית בסיסי הנתונים הביאה לשינוי בגישה המקובלת לניהול הנתונים וכתוצאה – נדרשו בעלי תפקידים חדשים.

אפשרות ריכוז הנתונים של הארגון במאגר אחד הפכה אותם למשאב של הארגון, ולא של היישום. המורכבות הרבה של מבני הנתונים, הקשרים ביניהם והרגישות הנובעת מריכוזם למאגר משותף אחד הביאו ליצירת תפקיד חדש, **מנהל בסיס הנתונים** – **DBA** (Database Administrator), או **מינהלן בסיס הנתונים**. חשוב להדגיש שמנהל בסיס הנתונים **אינו** הבעלים של הנתונים שנמצאים בבסיס הנתונים, אלא רק אחראי לניהולם התקין והיעיל ולאבטחתם. תוכן הנתונים במאגר נשאר באחריות היישומים השונים המזינים ומעדכנים את הנתונים.

מנהל בסיס הנתונים מעורב בכל שלבי ההקמה, התכנון, היישום והבקרה של מערכת RDBMS. מעורבותו נובעת מריכוז כל הנתונים במאגר אחד ומהצורך במומחה מקצועי שיתפעל את מערכת RDBMS, יפקח עליה באופן שוטף ואף יסייע למפתחי היישומים המשתמשים במאגר הנתונים. חשיבות התפקיד עולה ככל שהמערכת מורכבת יותר. במערכות גדולות ומורכבות ממלא את התפקיד צוות אנשי מקצוע בתחום בסיסי נתונים, כל אחד בעל תחום התמחות שונה. למנהל בסיס הנתונים תפקיד חשוב בשלבי התכנון והבחירה, בשלב עיצוב בסיס הנתונים ובעת ההפעלה השוטפת של המערכת.

נסקור בקצרה את תפקידי מנהל בסיס הנתונים, ה-DBA, כפי שבא לידי ביטוי במשך מחזור החיים של בסיס הנתונים, שלב אחר שלב:

- **תכנון ראשוני ובחירה**: בשלב זה הארגון צריך לבחור מערכת RDBMS חדשה. ה-DBA משתתף בבחירת המערכת המתאימה ביותר לצרכי הארגון, זאת תוך הבנת ההבדלים הטכנולוגיים בין המערכות השונות והגדרת רמת השירות הנדרשת מהספקים, ניתוח נקודות החולשה והעוצמה של כל מערכת ואפיון סוגי היישומים השונים הדרושים לארגון. הוא משתתף בתהליכי התכנון לטווח הארוך של צרכי המידע בארגון ויכול לשלב צרכים אלה עם אפשרויות היישום במערכות ניהול בסיסי נתונים לטווח הקרוב ולטווח הרחוק.
- **עיצוב בסיס הנתונים**: בשלב זה מוגדר המודל הכללי של הנתונים בארגון. תהליך זה מבוסס על צרכי המידע ועל דרישות העיבוד של היישומים השונים. יחד עם מנתחי המערכות, אחראי ה-DBA על עיצוב והקמת מודל הנתונים הלוגי שמתאר את הנתונים של הארגון בצורה עקבית ושלמה, ומשמש בסיס לפיתוח כל היישומים העתידיים. תהליך זה הוא בדרך כלל הדרגתי ובו מתפתח מודל הנתונים של הארגון עם הזמן, תוך כדי פיתוח ושילוב יישומים נוספים.
- **הגדרת המודל הפיסי של הנתונים**: בשלב זה ה-DBA מגדיר את המבנים הפיסיים של בסיס הנתונים. לא ניתן לפתח מבנים פיסיים שיאפשרו עיבוד יעיל בכל היישומים, ולכן צריך לקבוע איזה יישומים עדיפים מבחינת דרישות העיבוד ובהתאם לכך לבנות את בסיס הנתונים: טבלאות ממשיות, אינדקסים, טבלאות מדומות ועוד; אילו טבלאות יאוחסנו באותם אמצעי אחסון ולאילו יוקדשו דיסקים נפרדים וכד'. בתיאום עם מנהלי היישומים, הוא מגדיר את כללי אבטחת הנתונים במערכות ואת נהלי הגישה לנתונים לקריאה ולעדכון.
- **תפעול שוטף של בסיס הנתונים:** בשלב הפעלת היישומים, מנהל בסיס הנתונים עוקב באופן מתמיד אחר השינויים בטבלאות ובמידת השימוש בהן כדי לשפר, לייעל ולהעלות את רמת האמינות והזמינות. עליו להציע שינויים שונים בנהלי הפעלה, בתוכנה, או בחומרה לאחר הערכת התועלת הצפויה לעומת עלות השינוי. לדוגמה, ייתכן שניתן להוסיף אינדקס כדי לייעל את הגישה לטבלאות מסוימות, אבל אינדקסים אלה משפיעים על זמן העדכון וצורכים שטחי אחסון. דוגמה נוספת היא החלטה על כפילות נתונים במטרה לייעל ולשפר את זמן הגישה. מנהל בסיס הנתונים מכיר את המבנה הפנימי ולכן הוא צריך לשקוד על שיפורו המתמיד. עליו לבצע שינויים שונים במודל הנתונים, בהתאם לצרכים המשתנים של הארגון ולתת תמיכה

לפיתוח יישומים חדשים. תפקיד נוסף הוא פיתוח כלים לשמירת אמינות בסיס הנתונים על ידי שימוש בחומרה ובתוכנה לגיבוי והתאוששות, ולמעקב אחר גישת המשתמשים לבסיס הנתונים.

מעצבי בסיסי נתונים (Database Designers)

המורכבות ההולכת וגדלה של בסיס הנתונים והצורך בידע בתחומים הקשורים לעיצוב לוגי של בסיסי נתונים, הביאה להתמחות מיוחדת של אנשי מקצוע בתחום עיצוב בסיס הנתונים. אנשים אלה הם בעלי הבנה מעמיקה בנתונים המנוהלים בבסיס הנתונים, בקשרים בין הנתונים ובאילוצים שעל הנתונים לקיים. מעצבי בסיסי הנתונים עובדים בדרך כלל במקביל למעצבי התוכנה.

מהנדסי יישום (Application Engineers)

מהנדסי היישום מפתחים את היישומים שמושתתים על בסיס הנתונים. הם מתרגמים את מפרטי הדרישות ואת אפיון המערכת לאוסף של תוכניות יישום הכתובות בשפת תכנות אחת או יותר בהתאם לצורך. מהנדסי תוכנה אלה חייבים להכיר את המבנה העקרוני של בסיס הנתונים ואת השפה לגישה לנתונים, שפת SQL. הם משבצים את הפקודות לגישה לנתונים בתוך שפת תכנות מארחת כלשהי, או משתמשים במחוללי יישומים ודוחות לבניית היישום.

משתמשי קצה (End Users)

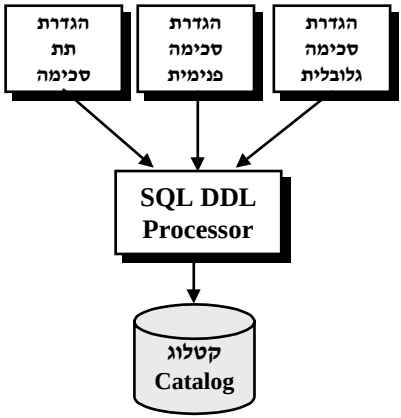
משתמשי הקצה הם האוכלוסיה שמשתמשת ביישומים השונים כדי לבצע את עבודתה. מקובל להבחין בין משתמשי קצה המפעילים רק יישומים מוכנים מראש, לבין משתמשי קצה מתוחכמים היודעים להפעיל מחוללי שאילתות ומחוללי יישומים כדי לאחזר נתונים על פי צרכים משתנים. בסביבה תפעולית, מרבית המשתמשים ניגשים אל הנתונים באמצעות היישומים השונים המוכנים מראש, ואילו בסביבת מחסן נתונים מקובל לאפשר למשתמשי קצה מסוימים לגשת ולתחקר את הנתונים באופן ישיר, תוך שימוש במחוללי שאילתות וכלי ניתוח מידע אחרים.

עיקרון הפעולה של מערכות RDBMS

לאחר שסקרנו את מרכיבי מערכות RDBMS – מערכות לניהול בסיסי נתונים טבלאיים – הבה נסקור את שיטת העבודה שלהן, החל משלב הגדרת בסיס הנתונים ועד שלב הגישה אליהם מתוך תוכנית יישום, או מתוך מחוללי שאילתות ודוחות. חשוב להדגיש שאלה הם עקרונות בלבד, וכל מערכת מסחרית מממשת חלק מעקרונות אלה בצורה שונה.

הגדרת בסיס הנתונים (Database Definition)

מערכת RDBMS מאפשרת הגדרה נוחה ומהירה של בסיס הנתונים. תהליך זה מתבצע באופן אינטראקטיבי על ידי דו-שיח בין מנהל בסיס הנתונים (DBA) לבין המערכת. ההגדרה מתבצעת בדרך כלל בהנחיה של תפריטים, המובילים את מנהל בסיס הנתונים דרך כל השלבים הנדרשים להגדרת בסיס הנתונים: הגדרת בסיס הנתונים, הגדרת הטבלאות, הגדרת העמודות, הגדרת האינדקסים וכד'.



תרשים 2.8: תהליך הגדרת בסיס הנתונים.

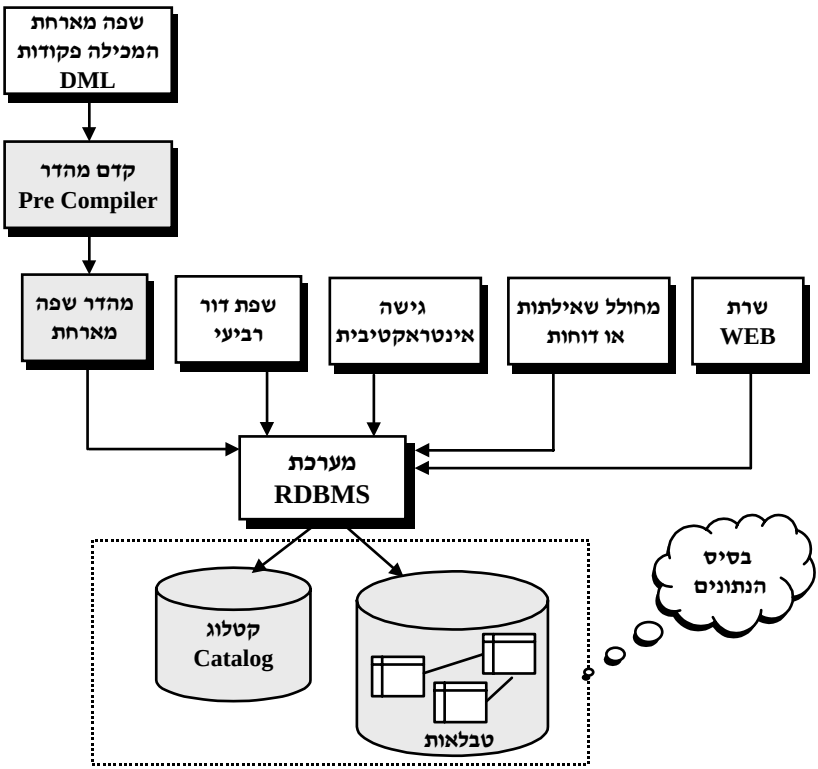
לצורך הגדרת בסיס הנתונים משתמש מנהל בסיס הנתונים בפקודות שונות וביניהן: CREATE TABLE להגדרת טבלה חדשה, CREATE VIEW להגדרת טבלה מדומה, ALTER TABLE לשינוי ההגדרות של טבלה, CREATE INDEX להגדרת אינדקס חדש לטבלה וכד'. פקודות אלו נמסרות לרכיב מיוחד של המערכת, מעבד פקודות SQL, והוא מפענח אותן, בודק את תקינותן ובונה מתוכן את ההגדרות המתאימות. לדוגמה, להגדרת טבלת סטודנטים יבצע מנהל בסיס הנתונים את הפקודה הבאה:

```
1. CREATE TABLE STUDENTS
2.              (STUDENT_ID  CHAR  (5) NOT NULL,
3.               NAME        CHAR (30),
4.               CITY        CHAR (40),
5. PRIMARY KEY (STUDENT_ID))
```

תהליך הגדרת בסיס הנתונים בונה אוסף של טבלאות המיועדות לשימוש המערכת לניהול בסיסי הנתונים, **קטלוג המערכת** (System Catalog). טבלאות אלו מכילות את כל המידע המתאר את מבנה בסיס הנתונים וכן מידע נוסף הדרוש למערכת כדי לבצע את פעולותיה.

גישה לבסיס הנתונים

לאחר שבסיס הנתונים הוגדר, מגיע שלב השימוש בבסיס הנתונים, שבו טוענים את הטבלאות בנתונים, מעדכנים אותם על ידי תנועות עסקיות ומאחזרים מהן נתונים למטרות דיווח וקבלת מידע. כפי שנראה מייד, קיימות שיטות גישה שונות לנתונים המאוחסנים בבסיס הנתונים: גישה אינטראקטיבית וישירה, גישה מתוך תוכניות יישום, גישה מתוך מחוללי שאילתות ודוחות וגישה מתוך שרת Web הפועל באינטרנט. נסקור בקצרה כל אחת מהן.



תרשים 2.9: שיטות גישה שונות לבסיס הנתונים.

גישה אינטראקטיבית (Interactive Access)

רוב מערכות RDBMS מספקות למשתמשי בסיס הנתונים (משתמשי קצה, מהנדסי תוכנה ומנהל בסיס הנתונים) כלים, המאפשרים להם גישה ישירה לבסיס הנתונים. הגישה לבסיס הנתונים מתבצעת על ידי שיגור פקודות SQL מתחנת עבודה (מחשב אישי או מסוף) אל השרת שבו נמצא בסיס הנתונים.

פקודת SQL הנפוצה ביותר בשיטת גישה זו מיועדת לשליפת שורות מתוך טבלה אחת או יותר של בסיס הנתונים. נשתמש בפקודה SELECT כדי להדגים את השלבים שמעבד SQL מבצע בתהליך העיבוד.

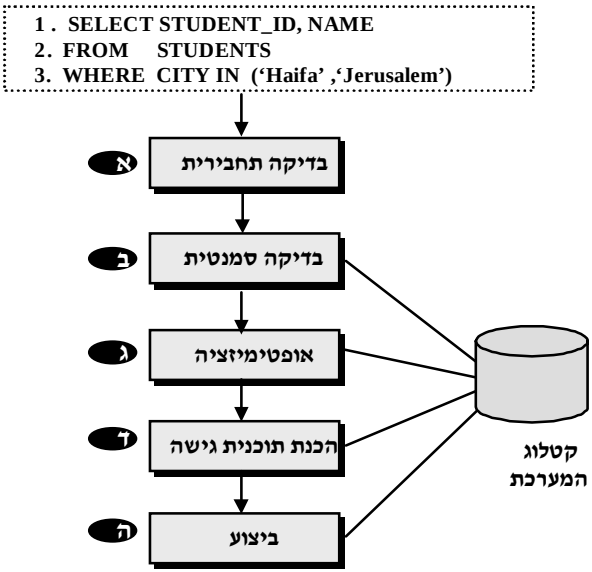
```
1. SELECT  STUDENT_ID, NAME
2. FROM    STUDENTS
3. WHERE  CITY IN  ('Haifa', 'Jerusalem')
```

הפקודה SELECT שולפת את מספר הסטודנט ואת שם הסטודנט לכל הסטודנטים הגרים בעיר חיפה או ירושלים. בשלב זה, נשים לב לשני מאפיינים חשובים של פקודת SQL:

- הפקודה יכולה לשלוף בבת אחד מספר גדול של שורות. זהו חידוש לעומת שיטת העבודה במערכות לניהול קבצים וגם לעומת הדור הקודם של מערכות לניהול בסיסי נתונים. במערכות אלו כל פקודה החזירה לתוכנית רשומה אחת בלבד.
- הפקודה מתארת בצורה פשוטה וברורה **מהם** הנתונים המבוקשים ואינה אומרת **כיצד** יש לאחזר אותם. במערכות קודמות, חלק ניכר מהלוגיקה של תוכנית היישום עסק בניווט מורכב למדי בתוך בסיס הנתונים. מערכות RDBMS מבצעות את החלטות הניווט באופן אוטומטי, ללא התערבות מהנדס התוכנה. מקובל לקרוא לתפישת העבודה הזו, שבאה לידי ביטוי באמצעות פקודות SQL, בשם תפישה לא-פרוצדורלית, לעומת התפישה הפרוצדורלית של מערכות ניהול הקבצים ומערכות DBMS הישנות יותר.

בפרקים הבאים נסביר בצורה מפורטת את מבנה פקודות SQL ונראה ששפה זו הינה רבת עוצמה ומאפשרת ביצוע מטלות מורכבות ביותר בצורה פשוטה ונוחה.

פקודת SQL משוגרת מתחנת העבודה אל מערכת RDBMS שבשרת. היא מועברת אל רכיב מיוחד של המערכת, **מעבד SQL** (SQL Processor). נציג בקצרה את השיטה שבה מעובדות פקודות SQL.



תרשים 2.10: שלבים בביצוע פקודת SQL.

כל פקודת SQL עוברת מספר שלבים עד לביצועה. הבה נסקור אותם.

- **שלב א' – סריקה ופיענוח (Parsing):** בשלב זה מעבד SQL סורק את פקודת SQL שקיבל, מפרק אותה למרכיביה הבסיסיים ובודק אם היא כתובה בצורה נכונה. פקודה "נכונה" בהקשר זה היא פקודה הנפתחת במילת המפתח, מכילה את כל המשפטים שהם משפטי החובה, מכילה משפטי רשות נכונים והתחביר שלה נכון. שלב זה יכול להתבצע ללא כל גישה לקטלוג מערכת RDBMS כי זו **בדיקה תחבירית** בלבד.

- **שלב ב' – בדיקת תקינות (Validation):** בשלב זה פקודת SQL נבדקת מול קטלוג המערכת. מתבצעת בדיקה שכל הטבלאות אכן קיימות, שכל עמודה אכן קיימת בטבלה, שלמשתמש יש הרשאות לגשת לטבלאות ולעמודות המבוקשות, ושמותר לו לבצע את הפעולה המבוקשת. בשלב זה מתקיימת **בדיקה סמנטית** לעומת הבדיקה התחבירית שבשלב הקודם.

- **שלב ג' – אופטימיזציה (Optimization), מיטוב:** שלב האופטימיזציה הוא אחד השלבים החשובים והייחודיים למערכות RDBMS. נבדקות בו חלופות שונות לביצוע הגישות הטובות ביותר לבסיס הנתונים למטרת אחזור על פי הדרישה. המעבד בודק אם קיימים אינדקסים או שיש צורך לבצע סריקות מלאות של טבלאות, האם למיין תחילה את הטבלה או לסרוק ללא מיון, ועוד. נתבונן בשאילתה שהצגנו ונראה שקיימות לפחות שתי חלופות לביצוע האחזור: (א) סריקה סדרתית של טבלת סטודנטים ושל כל שורה כדי לבדוק אם עיר המגורים היא חיפה או תל אביב; (ב) בדיקה אם קיים אינדקס לעמודה עיר מגורים, ואם קיים – לגשת אליו פעם אחת עם הערך "חיפה" ופעם שנייה עם הערך "ירושלים" ולשלוף ישירות את השורות המתאימות. רכיב המיטוב של המערכת יבחר באיזו משתי החלופות להשתמש. מעבר לשימוש באינדקסים, קיימים פרמטרים נוספים שרכיב המיטוב צריך לקחת בחשבון, כמו גודל הטבלאות וכיצד הן מאוחסנות בדיסק ועוד.

מקובלות בשימוש שתי שיטות מיטוב: **מיטוב מבוסס תחביר** (Syntax Based Optimization) **ומיטוב מבוסס עלות** (Cost Based Optimization). מיטוב מבוסס תחביר מתבסס על הסדר שבו מופיעות הטבלאות בפקודת SQL ועל הסדר שבו מופיעים התנאים הלוגיים. היתרון בשיטה זו הוא בפשטות וביכולת של המשתמש לשלוט בסדר ביצוע הפקודה. שני חסרונותיה הבולטים בכך שהיא מחייבת את הכרת מבנה בסיס הנתונים והעובדה שזו שיטה סטטית.

שיטת מיטוב זו פינתה את מקומה לשיטת מיטוב מבוססת עלות. על פי שיטה זו מעבד הפקודה אדיש למבנה פקודת SQL והוא בונה אסטרטגיה חכמה, שאינה תלויה בצורת הכתיבה של הפקודה. היא מבססת את אסטרטגיית החיפוש בנסיון להגיע לחלופה שבה עלות הביצוע (זמן חיפוש, למשל) תהיה הנמוכה ביותר. בשיטה זו המעבד מתבסס על נתונים סטטיסטיים שונים שנאספו באופן שוטף ונרשמו בקטלוג המערכת. שיטה זו מורכבת מאוד ודורשת שימוש בטכניקות מיטוב מתמטית מתוחכמות. כיום כל מערכות RDBMS המסחריות מבוססות על שיטה זו.

עם הופעת ארכיטקטורות חומרה מרובות-מעבדים כגון SMP, MPP, NUMA ואחרות, עודכן רכיב המיטוב של מעבד SQL. כך ניתן לנצל ארכיטקטורות חדשות אלו לפעולות מקבילות אשר תורמות לשיפור זמני העיבוד של פקודות SQL. כיום, רוב מערכות RDBMS המסחריות מסוגלות לפצל שאילתה למספר תת-שאילתות המתבצעות במקביל, כל אחת ביע"מ (יחידת עיבוד מרכזית) שונה באותו מחשב. לשיטה זו מקובל לקרוא Intraquery Parallelism (מקביל וּת תת-שאילתות). קיימות גם מערכות התומכות במקביל וּת תת-שאילתות בשיטה שונה: פיצול השאילתה למספר תת-שאילתות הפועלות על מחשבים נפרדים, שבכל אחד מהם מנוהלת מחיצה (Partition) נפרדת לטיפול באותה טבלה. שיטה זו מתאימה במיוחד לארכיטקטורות MPP.

דיון מעמיק בשיטות המיטוב של משפטי SQL היא מעבר למטרות ספר זה, ולכן לא נרחיב את הדיון בנושא.

- **שלב ד' – בניית תוכנית גישה (Access Plan):** לאחר ששלב המיטוב מצא את דרך הביצוע המיטבית, מתחיל תהליך הבנייה של תוכנית הגישה. תוכנית זו היא למעשה אוסף פקודות, לעיתים אוסף גדול מאוד, שיש לבצע כדי לממש את הבקשה. תוכנית הגישה מפותחת באופן אוטומטי על ידי מעבד SQL.
- **שלב ה' – ביצוע (Execution):** בשלב זה תוכנית הגישה מופעלת ומבוצעת. עם סיומה מוחזרת **התוצאה** למשתמש.

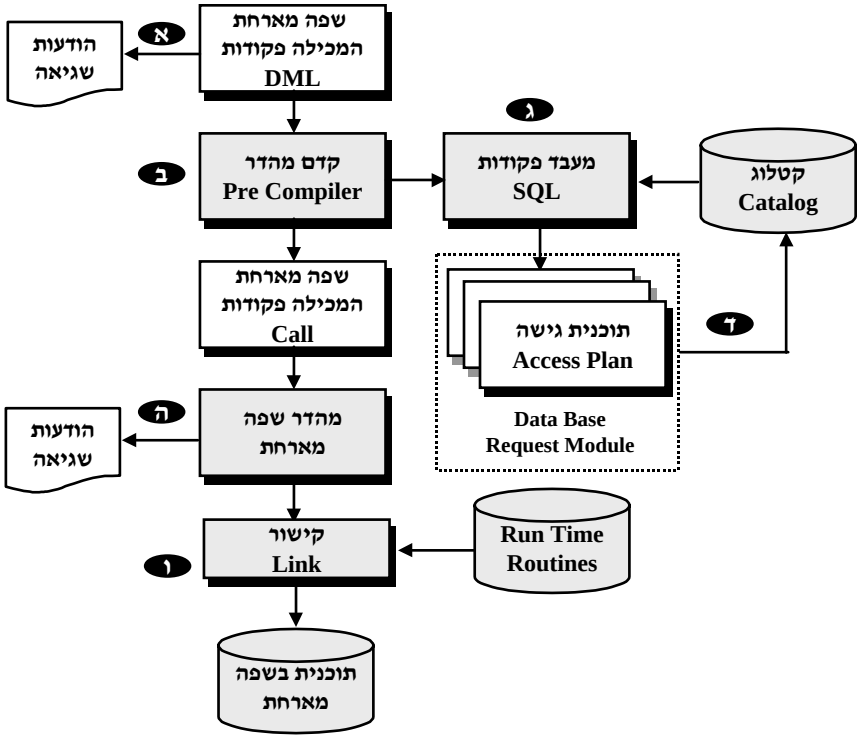
לסיכום, שיטת הגישה האינטראקטיבית מאפשרת גישה נוחה לבסיס הנתונים ללא צורך בכתיבת תוכניות יישום מורכבות. השימוש העיקרי של צורת גישה זו היא בעיקר לאחזור מהיר של נתונים, ולא לבניית לוגיקה מורכבת. אם יש צורך בלוגיקה מורכבת, הפתרון הוא כתיבת תוכניות יישום בשפות דור שלישי או רביעי.

גישה מתוכנית יישום הכתובה בשפת תכנות מדור שלישי

ניתן לגשת לבסיס הנתונים מתוך תוכנית יישום הכתובה בשפה מארחת כלשהי כגון C, COBOL, Java, Pascal, Ada וכו'. יצרני מערכות RDBMS פיתחו קדם-מהדר (Pre Compiler), לכל שפה מארחת. תפקיד קדם-המהדר הוא לאתר את כל פקודות SQL ששובצו בשפה המארחת. להלן שלבי העבודה בתהליך ההידור של תוכנית שמשובצות בה פקודות SQL:

- **שלב א' – כתיבת תוכנית יישום**: מהנדס התוכנה כותב תוכנית יישום בשפת תכנות כלשהי ומשבץ בתוכה פקודות SQL לגישה לבסיס הנתונים.
- **שלב ב' – קדם-הידור**: מכיון שפקודות SQL אינן חלק מאוסף הפקודות החוקי של השפה, קדם-המהדר מחליף אותן בפקודות Call, המוכרות על ידי כל מהדר. זיהוי פקודות SQL מתבצע על ידי סימון מיוחד של הפקודה בגוף התוכנית. בדרך כלל, פקודות אלו חייבות להתחיל בקידומת הקבועה EXEC SQL, ומסתיימות בצורה שונה בכל שפת תכנות. למשל, בשפת C הן מסתיימות בסימן ; ובשפת COBOL הן מסתיימות בסיומת END EXEC.

❖ **שלב ג' – הכנת מודול גישה לבסיס הנתונים**: קדם-המהדר מעביר כל פקודת SQL אל מעבד SQL (SQL Processor), שמעביר אותה דרך השלבים שתוארו בגישה האינטראקטיבית. להבדיל מהגישה האינטראקטיבית, שבה תוכנית הגישה לבסיס הנתונים הופעלה מיידית, תוכנית הגישה הזו אינה מופעלת באופן מיידי, אלא בעת ביצוע התוכנית המארחת. מקובל לקרוא לאוסף כל תוכניות הגישה השייכות לתוכנית יישום אחת בשם **מודול הגישה לבסיס הנתונים – DBRM** (DataBase Request Module). מודול זה מכיל את תוכנית הגישה עבור כל אחת מפקודות SQL שהופיעו בתוכנית היישום.



תרשים 2.11: הכנת תוכנית יישום לעבודה עם RDBMS.

❖ **שלב ד' – אחסון מודול הגישה**: מודול הגישה של תוכנית היישום מאוחסן בקטלוג המערכת. לכל תוכנית יישום נשמר מודול גישה נפרד, הכולל תוכנית גישה אחת עבור כל פקודת SQL שהופיעה בתוכנית היישום. תוכנית הגישה תופעל מתוך מודול הגישה בזמן הריצה של תוכנית היישום, בעת ביצוע פקודת Call שנשתלה בתוכנית במקום פקודת SQL.

❖ **שלב ה' – הידור השפה המארחת**: תוכנית היישום שבה הוחלפו כל פקודות SQL בפקודות Call מועברת למהדר של השפה המארחת.

❖ **שלב ו' – קישור**: לאחר שלב ההידור עוברת התוכנית תהליך קישור (Link) לשגרות זמן הריצה של מערכת RDBMS. בשלב זה תוכנית היישום מוכנה להפעלה.

גישה מתוך תוכנית הכתובה בשפת תכנות מדור רביעי

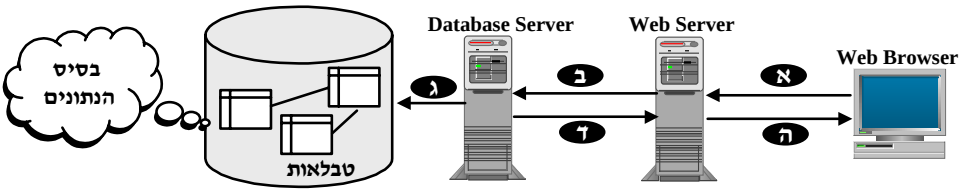
שפות התכנות העיליות מדרגה גבוהה יותר הן **שפות דור רביעי** – **4GL** (4th Generation Language). למרות שאין הסכמה על ההגדרה מהי שפת דור רביעי, הדרך הנכונה ביותר להגדירן היא דרך רמת התפוקה של מהנדסי התוכנה. הטענה העיקרית היא ששפות אלו מספקות רמת תפוקה גבוהה ביחס פי 10. חלק משפות הדור הרביעי פותחו על יד יצרני צד שלישי, כמו Magic של חברת משוב, סביבת העבודה Delphi של חברת Insight או Visual Basic של חברת מיקרוסופט. בנוסף, רוב יצרני מערכות RDBMS פיתחו שפות תכנות שתומכות בניהול בסיס הנתונים וביניהן Developer/2000 של Oracle או PowerBuilder של Sybase. שפות דור רביעי אלו מיועדות לפיתוח מהיר של יישומים הפועלים עם בסיס הנתונים ומספקות מיגוון רחב של פונקציות, שלא נמצאות בדרך כלל בשפות הדור השלישי (כגון טיפול בתפריטים, חלונות וכו'). הן מכילות גם אוסף שלם של פקודות התומכות בעבודה עם בסיסי נתונים טבלאיים. שפות אלו נבנו מראש לעבודה עם בסיסי הנתונים, ולכן המהדר או המפענח (Interpreter) של השפה מכיר את פקודות SQL ואין צורך בשירותי קדם-מהדר.

מחוללי שאילתות ודוחות (Query and Report Generators)

בדומה לשפות דור רביעי, פיתחו יצרני צד שלישי ויצרני מערכות RDBMS מחוללי שאילתות ומחוללי דוחות המותאמים לעבודה עם בסיסי נתונים. מחוללים אלה מנצלים את עוצמת שפת הגישה לבסיס הנתונים ומוסיפים פונקציונליות על ידי הגדרה בשיטת ציור על המסך, תפריטים מובנים, בחירת טבלאות ועמודות מתוך תפריטים, הגדרה נוחה של תנאי אחזור, הגדרת מבנה דוח (כותרות, סיכומים וכד') ועוד. בין סביבות אלו נציין את מחוללי השאילתות Discoverer/2000 של Oracle ו- Business Objects של Business Objects. בין מחוללי הדוחות נציין את SQR.

גישה משרת Web

התפוצה הרחבה של טכנולוגיית האינטרנט הביאה איתה מודל גישה נוסף לבסיסי נתונים: מודל הגישה מתוך שרת Web. נסקור את העקרונות של מודל זה באמצעות דוגמה. נקים מערכת המאפשרת לסטודנטים במכללה לבצע את הרישום לקורסים באמצעות רשת האינטרנט ישירות מביתם, או ממחשבים אישיים הפזורים ברחבי הקמפוס.



תרשים 2.12: גישה לבסיס נתונים משרת Web.

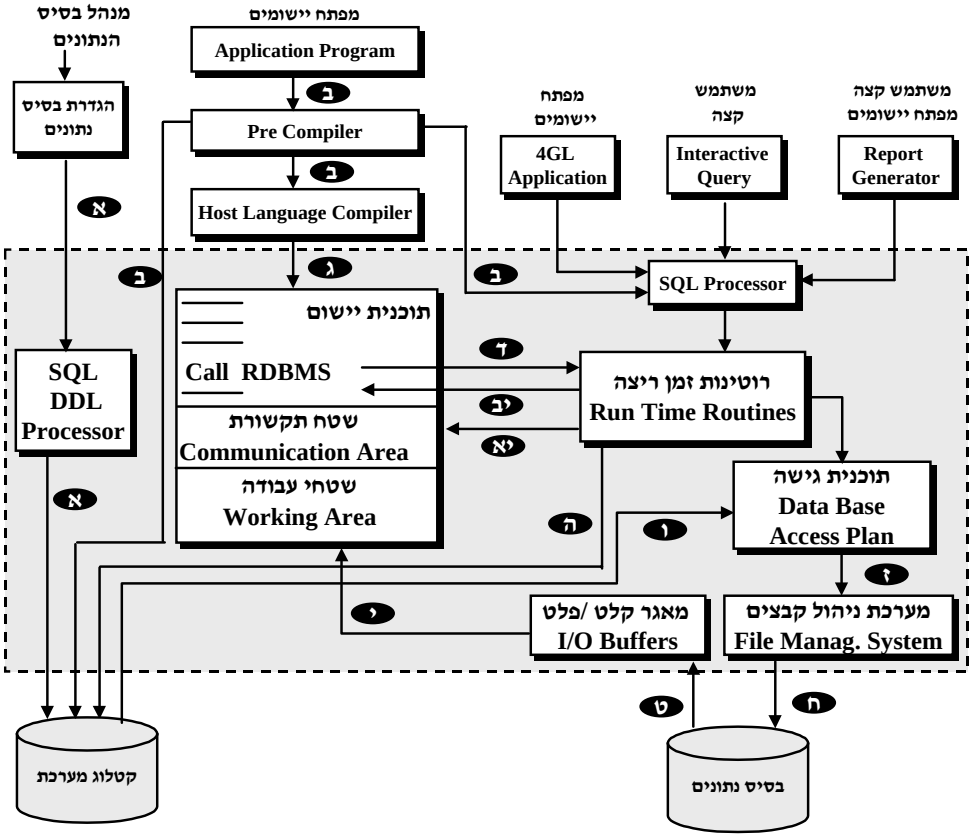
נסקור את שלבי העבודה במודל זה:

- **שלב א' – הפעלת היישום מתוך הדפדפן**: המשתמש פועל בתחנת העבודה שלו באמצעות דפדפן (Browser), נכנס לאתר המבוקש ומפעיל את היישום הרצוי לו. בודגמה שלנו, הסטודנט ניגש לאתר האינטרנט של המכללה, עובר תהליך הזדהות ובדיקת הרשאת גישה ומפעיל את יישום הרישום לקורס. היישום שולח אליו דף במבנה HTML, הכולל בין השאר טופס רישום מתאים. הסטודנט ממלא את טופס הרישום לקורס ומזין בו את מספר הקורס המבוקש, הסמסטר, יום בשבוע וקבוצת הלימוד הרצויה. נתונים אלה נארזים על ידי הדפדפן ונשלחים אל שרת Web, באמצעות פרוטוקול HTTP.
- **שלב ב' – בניית השאילתה בשרת Web**: שרת Web מקבל את הפנייה ומעורר פרוצדורה מיוחדת להפעלת יישום זה. היישום מקבל את הפרמטרים הרלוונטיים, בונה שאילתת SQL באופן דינמי ומשתמש בממשק מיוחד כדי לשלוח את השאילתה לשרת בסיס הנתונים.
- **שלב ג' – ביצוע השאילתה בשרת בסיס הנתונים**: שרת בסיס הנתונים, שבו פועלת מערכת RDBMS, מקבל את שאילתת SQL משרת Web. השאילתה נמסרת למעבד SQL המבצע את שלבים הפעולה הרגילים לכל בקשה: סריקה ופיענוח, בדיקת תקינות, מיטוב, בניית תוכנית גישה וביצוע השליפה. מנקודת מבט של שרת בסיס הנתונים, אין כל הבדל במתן השירות ליישום המופעל מתוך שרת Web או ליישום המופעל ישירות מתחנת העבודה של המשתמש.
- **שלב ד' – החזרת תוצאות השאילתה לשרת Web:** תוצאות השאילתה מוחזרות משרת בסיס הנתונים אל שרת Web ואל היישום של רישום סטודנט לקורס. היישום אורז את התוצאה במבנה מיוחד של דף HTML ומשגר אותו.
- **שלב ה' – שיגור התוצאה אל הדפדפן**: דף HTML משוגר ברשת האינטרנט בפרוטוקול HTTP אל הדפדפן של המשתמש. על מסך המשתמש מופיע המידע בהתאם ליישום.

מודל העבודה של מערכת RDBMS

כפי שראינו, מערכת לניהול בסיסי נתונים מורכבת ממספר חלקים שכל אחד מהם ממלא פונקציה מסוימת שהוא אחראי לביצועה. להלן תיאור של המודל הכללי של מערכת RDBMS ושל הפעולה המשולבת של מרכיבי המערכת.

תרשים 2.13 מציג באופן סכמתי את עיקרון פעולת המערכת במהלך העבודה של תוכנית יישום הכתובה בשפת תכנות כלשהי, אשר מבקשת לגשת לבסיס הנתונים ולאחזר ממנו נתונים מסוימים על סטודנט מסוים. לצורך הדגמה נניח שהתוכנית מבקשת לאחזר את נתוני הסטודנט באמצעות טבלה מדומה.



תרשים 2.13: מודל להצגת עיקרון הפעולה של מערכת RDBMS.

- **שלב א' – הגדרת בסיס הנתונים**: מנהל בסיס הנתונים מגדיר את בסיס הנתונים (טבלאות, עמודות, אינדקסים, קשרים וכד') ומשתמש בפקודות DDL של SQL. פקודות אלו נבדקות ומעובדות על ידי רכיב מיוחד המטפל בפקודות הגדרת הנתונים. כתוצאה מכך, נוצר קטלוג המערכת, שהוא אוסף טבלאות **המתארות** את בסיס הנתונים. בדוגמה שלנו הוא מגדיר תחילה את טבלת הסטודנטים STUDENTS, ואחר כך הוא מגדיר את הטבלה המדומה STUDENTS_NAME המבוססת עליה.
- **שלב ב' – כתיבת תוכנית יישום והידורה**: מהנדס התוכנה כותב את תוכנית היישום ומשבץ בתוך שפת התכנות המארחת פקודות SQL לטיפול בנתונים. תוכנית היישום עוברת תהליכי קדם-הידור והידור של השפה המארחת. קדם-המהדר מאתר את כל פקודות SQL המשובצות בתוכנית, מפעיל את מעבד SQL שבודק את הפקודה מבחינה תחבירית, מבצע מיטוב ובונה את תוכנית הגישה לבסיס הנתונים. תוכנית הגישה מאוחסנת בקטלוג המערכת. בתום ההידור מתקבלת תוכנית היעד (Object) שעוברת קישור (Link) ונמצאת במצב מוכן להפעלה.
- **שלב ג' – הפעלת התוכנית**: תוכנית היישום נטענת לזיכרון ומתחילה לפעול.

- **שלב ד' – בקשה לאחזור נתוני סטודנט**: תוכנית היישום מבקשת ממערכת RDBMS לאחזר את השורה של סטודנט מסוים. התוכנית מבקשת שורה מהטבלה המדומה STUDENTS_NAMES שהמבנה שלה כבר הוגדר בקטלוג המערכת. שגרות זמן הריצה של מערכת RDBMS מופעלות באמצעות הוראת CALL שבגוף תוכנית היישום. הוראה זו הוכנסה על ידי קדם-המעבד והיא מחליפה את פקודת SQL המקורית.
- **שלב ה' – בדיקת הבקשה מול קטלוג המערכת**: שגרות זמן הריצה של מערכת RDBMS מנתחות את הבקשה ומשתמשות לשם כך בקטלוג המערכת.
- **שלב ו' – שליפת מודול גישה לבסיס הנתונים**: אם התוכנית מורשית לבצע את הפקודה, שגרות זמן הריצה שולפות את תוכנית הגישה לנתונים מתוך מודול הגישה שהוכן בשלב ב' ומאוחסן בקטלוג המערכת. להזכיר, לכל פקודת SQL בתוכנית היישום הוכנה תוכנית גישה נפרדת, וכולן יחד נשמרו במודול הגישה. בשלב זה נשלפת תוכנית הגישה המתאימה לפקודת SQL שצריכה להתבצע בשלב הזה.
- **שלב ז' – הפעלת תוכנית הגישה**: תוכנית הגישה מתחילה לפעול ופונה למערכת ניהול הקבצים לקבלת הנתונים מטבלת הסטודנטים. נשים לב, שמערכת RDBMS משתמשת כאן בשירותי מערכת ניהול הקבצים. חלק מהמערכות עוקפות את מערכת ניהול הקבצים ומנהלות את שטח הדיסק העומד לרשותה, כלומר מתייחסות אל שטח האחסון כאל שטח גולמי (Raw Device) ומנהלות אותו באופן ישיר. לצורך התיאור המובא כאן, נניח שמערכת RDBMS שלנו משתמשת בשירותי FMS. הבקשה כוללת את כל הפרמטרים הדרושים למערכת ניהול הקבצים כדי לבצע את הגישה ליחידת האחסון. רוב הפרמטרים מתקבלים מהסכימה הפנימית (המציינים היכן הטבלה נמצאת על הדיסק, כמה שורות יש בגוש אחד, היכן האינדקסים וכד').
- **שלב ח' – פנייה לקבלת הנתונים מיחידת האחסנה**: מערכת FMS מפעילה את שגרות מערכת ההפעלה לגישה לנתונים המאוחסנים ביחידת האחסון. השגרות מטפלות בבת-אחת בגושים שלמים של נתונים.
- **שלב ט' – העברת הגוש למאגר קלט /פלט**: גוש שמכיל את השורה המבוקשת מועבר אל שטח מאגר קלט/פלט (I/O Buffer). עם סיום העברת הגוש אל המאגר, מערכת ניהול הקבצים מחזירה את הפיקוח למודול הגישה. מודול הגישה מוצא במאגר את השורה המבוקשת.
- **שלב י' – מיפוי מבנה ממשי למבנה לוגי**: תוכנית הגישה משתמשת בהגדרת הטבלה המדומה המופיעה בקטלוג כדי לבצע את המיפוי הדרוש בין עמודות הטבלה כפי שהן מופיעות בבסיס הנתונים, לבין עמודות הטבלה המדומה. השורה המבוקשת מועברת לשטח העבודה בתוכנית היישום. נשים לב לכך שלשטח העבודה מועברת שורה לוגית שאינה קיימת באופן ממשי בבסיס הנתונים.

- **שלב י"א – דיווח סטטוס הפקודה**: תוכנית הגישה מעבירה מספר נתונים אל שטח התקשורת שבין מערכת RDBMS לבין תוכנית היישום, וביניהם גם קוד החזר (Return Code). קוד ההחזר מציין את אופן סיום הבקשה. קוד החזר "0" מציין שהשורה המבוקשת נמצאה בבסיס הנתונים והועברה לשטח העבודה. קוד החזר שלילי מציין שהשורה המבוקשת לא נמצאה בטבלה.
- **שלב י"ב – החזרת הפיקוח לתוכנית היישום**: שגרות זמן הריצה של מערכת RDBMS מחזירות את הפיקוח לתוכנית היישום. לאחר בדיקת קוד ההחזר שבשטח התקשורת, תוכנית היישום מתחילה את עיבוד השורה שבשטח העבודה.

כפי שניתן לראות מתיאור זה, אי התלות בין מבנה הנתונים שבו משתמשת תוכנית היישום, לבין מבנה הנתונים המנוהל בבסיס הנתונים נובעת משני מיפויים: המיפוי שבין הגדרת הטבלה המדומה לבין הגדרת הטבלה, והמיפוי שבין ההגדרה הלוגית של הטבלה לבין ההגדרה הפיסית של הטבלה כפי שהיא מופיעה בסכימה הפנימית. לדוגמה, הוספת עמודה חדשה בטבלת סטודנטים לא תשפיע על תוכנית היישום, כי עמודה זו אינה מופיעה בטבלה המדומה שבה היא משתמשת. נרחיב בהמשך את הדיון בנושא אי-התלות בנתונים.

שירותים במערכת לניהול בסיסי נתונים

בסעיף זה נעסוק באוסף השירותים שמערכת RDBMS אמורה לספק, מעבר לשירותים הבסיסיים והמובנים מאליהם של אחסון, שליפה ועדכון של הנתונים המנוהלים בבסיס הנתונים. באחד ממאמריו הציג Codd, אבי המודל הטבלאי, רשימת שירותים שמערכת RDBMS אמורה לספק:

א. **אחסון, עדכון ושליפה**: מערכת RDBMS צריכה לאפשר אחסון, עדכון ושליפת נתונים המאוחסנים בבסיס הנתונים, מבלי לערב את המשתמש בטיפול במבנים פיסיים.

ב. **ניהול קטלוג המערכת**: מערכת RDBMS צריכה לאפשר ניהול קטלוג מרכזי, אשר יכיל את כל ההגדרות של בסיס הנתונים. מידע זה צריך להיות זמין למשתמשים המורשים.

ג. **תמיכה בעיבוד תנועות ועדכון מרובה משתמשים**: מערכת RDBMS מרובת משתמשים צריכה להכיל אוסף מנגנונים, שיאפשרו למספר רב של משתמשים לגשת ולעדכן בו-זמנית את בסיס הנתונים.

ד. **אבטחת נתונים**: מערכת RDBMS צריכה לספק מנגנונים שיבטיחו שרק משתמשים מורשים יוכלו לגשת לבסיס הנתונים. משתמשים מורשים אלה יוכלו לבצע רק את הפעולות המותרות להם על חלקים של בסיס הנתונים.

אחסון, עדכון ושליפה

פעולות אחסון, עדכון ושליפה של נתונים הן השירות הבסיסי ביותר של כל מערכת RDBMS. קיים הבדל בסיסי בין מערכת RDBMS לבין מערכת ניהול קבצים, FMS, שגם היא מספקת שירות בסיסי זה. ההבדל מתבטא ברמת **אי-התלות בנתונים** המסופקת על ידי מערכת DBMS.

אי-תלות בנתונים **Data Independence**	***אי-תלות בנתונים*** **מוגדרת כיכולת להשתמש בנתונים המאוחסנים בבסיס הנתונים מבלי להכיר את פרטי שיטת אחסונם, ולבצע שינויים מסוימים בבסיס הנתונים מבלי שתוכניות היישום יושפעו מכך.**

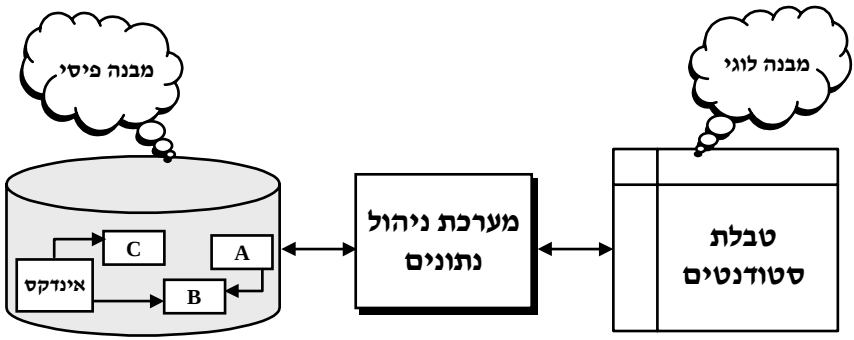
בגלל החשיבות הרבה של שירותי אי-תלות בנתונים שמערכת RDBMS אמורה לספק, נרחיב במקצת את הדיון בנושא זה.

ניתן להתייחס לנתונים בבסיס הנתונים משתי נקודות מבט שונות: על פי המבנה הלוגי ועל פי המבנה הפיסי.

מבנה לוגי ופיסי של נתונים

המבנה הלוגי של הנתונים (Logical Data Structure) הינו המבנה מ**נקודת המבט** של המשתמש או של תוכנית היישום. זהו למעשה מבנה מופשט הקיים רק בדמיונם של המשתמש או של מהנדס התוכנה. בדרך כלל המשתמש "רואה" את הנתונים במבנה מסודר כלשהו, למשל מבנה של טבלה בעלת שורות ועמודות. מנקודת מבט זו, אחזור הנתונים מתבצע על ידי פקודות SELECT של SQL.

את המבנה הלוגי של הנתונים ניתן לתאר בצורתו המופשטת באמצעות כלים שונים, כגון **מודל הנתונים**, אשר יפורט בפרק הבא.



תרשים 2.14: תרגום המבנה הפיסי למבנה לוגי.

המבנה הפיסי של הנתונים (Physical Data Structure) מתייחס לאופן האחסון של הנתונים ביחידות האחסנה (דיסק מגנטי, דיסק אופטי, דיסקט וכד'). מבנה זה מכיל את **הייצוג הבינארי של הנתונים**, ורק הוא קיים **באמת** במערכת הממוחשבת.

התבוננות במבנה הפיסי של הנתונים מחייבת התייחסות לנושאים מורכבים, כגון כתובות פיסיות, אורך השורה, מספר שורות לגוש, מצביעים, אינדקסים, שטחי אחסון לא רציפים ועוד. הטיפול במבנה זה טכני ומורכב מאוד והוא מעל יכולתו או עניינו של מהנדס התוכנה או המשתמש הסביר. הם מעדיפים לחשוב במונחי המבנה הלוגי (טבלאות, שורות, עמודות) ולא במונחי המבנה הפיסי (סיביות, כתובות על דיסק, גושים וכד').

אי-תלות לוגית ופיסית

התפקיד המרכזי של מערכת ניהול נתונים הוא לבצע המרה בין המבנה הלוגי לבין המבנה הפיסי. ככל שהמבנה הלוגי מופשט יותר ומכיל פחות התייחסויות למבנה הפיסי, נוכל לומר ש**אי-התלות בנתונים** גבוהה יותר. מקובל להבחין בין שתי רמות של אי-תלות בנתונים: אי-תלות פיסית ואי-תלות לוגית.

אי-תלות פיסית בנתונים **Physical Data Independence**	***אי-תלות פיסית בנתונים*** **מוגדרת כיכולת לבצע שינויים במבנה הפיסי של הנתונים מבלי שיהיה צורך לשנות את תוכניות היישום.**

דוגמה לשינוי פיסי היא העברת טבלה משטח אחסון אחד לאחר. העברה כזו צריכה להיות שקופה לחלוטין לתוכניות היישום, למרות שהמבנה הפיסי של הטבלה עלול להיות שונה בגלל תכונות הדיסק שמבנהו עשוי להיות שונה. דוגמה אחרת לצורך באי-תלות פיסית היא שינוי גודל הגוש כדי לשפר את זמני הקריאה/כתיבה ועקב כך את זמני התגובה. דוגמה נוספת היא הוספת אינדקס לטבלה כדי שגישות מסוימות תהיינה יעילות יותר. לכל הפעולות האלו אין נגיעה למבנה הלוגי של הנתונים ולכן הן צריכות להיות שקופות ולא צריכות להשפיע על תוכניות היישום. ניתן לומר שכל השינויים ברמה הפיסית מיועדים לשפר בצורה כלשהי את ביצועי המערכת ולכן הם אינם צריכים להשפיע על המבנה הלוגי, אשר למעשה נשאר ללא שינוי.

אי-תלות לוגית בנתונים **Logical Data Independence**	***אי-תלות לוגית בנתונים*** **מוגדרת כיכולת לבצע שינויים במודל הלוגי של הנתונים מבלי שיהיה צורך לשנות את תוכניות היישום.**

דוגמאות לשינוי לוגי הן הוספת עמודה חדשה לטבלה קיימת, יצירת קשר חדש בין שתי טבלאות, הוספת טבלה חדשה ועוד. לא ניתן לדרוש אי-תלות לוגית לכל סוג שינוי שיבוצע. לדוגמה, סילוק עמודה מטבלה ישפיע על כל תוכניות היישום המשתמשות בעמודה זו, אולם אין כל סיבה שהשינוי ישפיע על תוכניות יישום המשתמשות בטבלה אך אינן פונות לעמודה המבוטלת.

מערכות ניהול קבצים מספקות אי-תלות פיסית באופן מוגבל, אם הן מאפשרות לבצע שינויים מסוימים במבנה הפיסי ללא צורך בשינוי תוכניות היישום. יש שינויים במבנה הפיסי שמשפיעים על תוכניות היישום, כמו למשל שינוי מספר הרשומות לגוש שמחייב הידור של כל תוכניות היישום הניגשות אל הקובץ, למרות שלא חל כל שינוי במבנה הלוגי שלו. הסיבה לכך היא, שבמערכות לניהול קבצים מספר הרשומות לגוש מוגדר בכל תוכנית יישום שניגשת אל הקובץ. דוגמה אחרת, הוספת אינדקס חדש לקובץ. זהו שינוי פיסי במהותו, אולם נדרש לבצע שינויים בתוכניות היישום כדי להשתמש באינדקס החדש.

לעומת מצב זה, מערכות RDBMS מספקות רמה גבוהה מאוד של אי-תלות פיסית ולוגית. למשל, על ידי שימוש במנגנון הטבלה המדומה ניתן להסתיר מתוכנית היישום את העובדה שעמודה חדשה התווספה לטבלה קיימת, או שעמודה קיימת בוטלה. הוספת אינדקס לגישה מהירה יותר לרשומה או לחילופין ביטול אינדקס, אינם משפיעים על תוכניות היישום, אלא רק על זמני הביצוע. בחירת גישה לטבלה דרך האינדקס החדש תיעשה באופן אוטומטי על ידי מעבד SQL בשלב המיטוב.

מערכת RDBMS אשר מספקת שירותי אי-תלות פיסית ואי-תלות לוגית, היא בעלת היתרונות הבאים:

- נוחות שימוש מנקודת מבט המשתמש, או מהנדס התוכנה, שאינו צריך לטפל בהיבטים פיסיים של אחסון הנתונים.
- שיפור משמעותי בתפוקה (פרודוקטיביות) של מהנדס התוכנה. הוא יכול לפנות אל הנתונים באופן לוגי ובצורה לא-פרוצדורלית באמצעות פקודות SQL, מבלי שיצטרך לדעת כיצד הנתונים מאורגנים ביחידות האחסון השונות.
- אפשרות לכוונון (Tuning) של המבנה הפיסי לשיפור ביצועי המערכת, ללא שינוי בתוכניות היישום.
- יכולת להתאים את המבנה הלוגי של הנתונים לצרכים המשתנים של הארגון, תוך מזעור ההשקעה הנדרשת להתאמת תוכניות היישום. עם זאת, שינויים מסוימים במבנה הלוגי עלולים לחייב שינוי מקביל בתוכניות היישום.

ניהול קטלוג המערכת

קטלוג המערכת הינו אחד המאפיינים הבולטים של מערכות RDBMS. הגדרת הנתונים, הקשרים והאילוצים מתבצעת באופן חיצוני לכל תוכניות היישום ומנוהלת באופן מרכזי על ידי המערכת. הקטלוג **פעיל** (Active) במובן זה שכל פנייה אל בסיס הנתונים צריכה להתבצע דרכו. כך הוא משמש לא רק כלי תיעוד, אלא מהווה חלק בלתי נפרד ממערכת RDBMS.

מערכת RDBMS מספקת שפה שמאפשרת למנהל בסיס הנתונים לבצע את כל ההגדרות הנדרשות; זהו מרכיב **DDL** בתוך שפת SQL. במערכות DBMS הישנות יותר, שפת DDL היתה שפה נפרדת ומיוחדת שיועדה לשימושו הבלעדי של מנהל בסיס הנתונים. במערכות RDBMS, שפת DDL מהווה חלק משפת SQL.

להלן מספר דוגמאות לסוגי הגדרות, שמערכת RDBMS מנהלת בקטלוג:

- ❖ רשימת הטבלאות המנוהלות בבסיס הנתונים.
- ❖ רשימת העמודות שיש בכל טבלה.
- ❖ המאפיינים של כל עמודה (אורך, טיפוס נתונים).
- ❖ המפתח העיקרי של כל טבלה.
- ❖ ציון האינדקס של הטבלה והעמודות שעבורן הוא מוגדר.
- ❖ הקשרים בין הטבלאות בבסיס הנתונים ונוהל הטיפול בקשרים אלה.
- ❖ הערכים המותרים לנתונים בעמודות השונות.
- ❖ טבלאות מדומות שהוגדרו והשאילתה שמגדירה כל אחת מהן.
- ❖ משתמשים או תוכניות יישום שמורשים לגשת לכל טבלה ומה מותר להם לבצע בכל טבלה (קריאת נתונים בלבד, עדכון, ביטול וכד').
- ❖ פרוצדורות מיוחדות שיופעלו בתנאים מסוימים לביצוע מטלות מוגדרות.

הקטלוג מכיל למעשה **מידע על הנתונים** (Meta Data): מידע על האובייקטים המאוחסנים בבסיס הנתונים, שמות העמודות, סוגי הקשרים בין האובייקטים, תוכניות היישום המשתמשות בנתונים, המשתמשים המפעילים המורשים להפעיל תוכניות יישום שונות, ועוד. מכיון שבסיס הנתונים מכיל מרכיב המתאר את תכולתו, מקובל לומר שבסיס הנתונים הוא **אובייקט המתאר את עצמו** (Self Describing Object). **קטלוג הנתונים מהווה מקור מידע אחיד ומרכזי על תכולת בסיס הנתונים**.

השימוש בקטלוג נעשה על ידי גורמים שונים, שכל אחד מהם יכול להפיק ממנו את היתרונות המתאימים לצרכיו. הבה נתבונן בתרומת הקטלוג למשתמשים השונים:

המשתמש במערכת המידע:

- ❖ הבנה טובה של הנתונים המאוחסנים בבסיס הנתונים ומשמעותם (לרוב ניתן גם תיאור מילולי).
- ❖ כלי תיעודי גם לנתונים לא-טכניים הקשורים במערכת המידע.
- ❖ בסיס לתקשורת עם אנשי יחידת מערכות המידע.
- ❖ מידע על הנתונים שהוא משתמש בהם ועל הנתונים שהוא מייצר.
- ❖ סיווג הנתונים לפי ה"בעלים" שלהם בארגון (פרטיים או מחלקות).

מנתח המערכות:

- ❖ בדיקת חלופות ליישום מערכת חדשה.
- ❖ בדיקת עלות השינוי הצפויה לפני ביצוע שינוי כלשהו במערכת.

- הגדרה נוחה של השימוש בנתונים ושימוש חוזר בהגדרות עבור תוכניות אחרות.
- כלי תיעוד לנתונים טכניים אודות בסיס הנתונים והתוכניות.

מהנדס תוכנה:

- חיסכון בזמן פיתוח תוכנה, תוך שימוש בהגדרות קיימות של נתונים ותוכניות,
- סיוע למהנדסי תוכנה חדשים להכיר את מבנה הנתונים והקשרים ביניהם,
- כלי תיעוד אוטומטי לתוכניות ונתונים.

מנהל בסיס הנתונים:

- בקרה ושליטה על הנתונים שבבסיס הנתונים.
- הקטנת כפילות הנתונים.
- אכיפת סטנדרטים לנתונים ולתוכנה.
- "חקירה" מהירה של רכיבי בסיס הנתונים.

תמיכה בעיבוד תנועות ועדכון מרובה משתמשים

אחד השירותים החשובים שמערכת RDBMS מספקת הוא תמיכה בעיבוד תנועות ובגישה בו-זמנית של מספר רב של משתמשים אל בסיס הנתונים. שירות זה מבטיח שמירה על **אמינות ושלמות בסיס הנתונים** בסביבה מרובת משתמשים, בעת תמיכה ב**עיבוד תנועות** (Transaction Processing).

בגלל מורכבות הנושא נדחה את הדיון בו לפרק 14.

אבטחת נתונים

ריכוז כל הנתונים של הארגון בבסיס נתונים מרכזי אחד והשימוש הנרחב במערכות תקשורת המאפשרות למשתמשים מרוחקים לגשת אל בסיס הנתונים, מחייבים שמערכת RDBMS תספק מספר שירותים העוסקים בנושא **אבטחת נתונים**.

אבטחת נתונים עוסקת בעיקר במניעת אפשרות הגישה לבסיס הנתונים לגורמים שאינם מורשים. היא גם עוסקת בהגבלת אפשרויות פעולה של משתמשים מורשים, כמו למשל חסימת אפשרות כתיבה למי שהותרה להם אפשרות קריאה בלבד. הגורמים יכולים להיות משתמשים בארגון ומחוצה לו, או תוכניות יישום ומתכנתים בארגון. לדוגמה, תוכנית הסורקת את טבלת כוח האדם לצורך פעילויות רווחה של העובדים (התפלגות לפי גיל, מצב משפחתי, ועוד) אינה צריכה "לראות" את נתוני השכר, או נתונים רפואיים של העובד.

אבטחת נתונים אפשר להשיג במספר דרכים נפרדות או משולבות:

- הרשאת גישה: מי רשאי לגשת לנתונים.
- סיווג הגישה: מהן הפעולות המותרות למשתמש.
- תחום גישה: איזה מהנתונים יכול המשתמש לראות.

מנגנוני אבטחת הנתונים מורכבים בדרך כלל מכללי הרשאות הנקבעים על ידי מנהל בסיס הנתונים ועל ידי היישום, על ידי טבלאות מדומות, הצפנה ופרוצדורות ייחודיות המאוחסנות בבסיס הנתונים ומופעלות במקרים מסוימים. נושא זה מוסבר בהרחבה בפרק 11.

יתרונות וחסרונות של טכנולוגיית בסיסי הנתונים

טכנולוגיית בסיסי הנתונים מחליפה במקרים רבים את טכנולוגיית הקבצים. לטכנולוגיה זו יש יתרונות בצד חסרונות. נציג את החשובים שבהם.

יתרונות

- **שיתוף נתונים (Data Sharing)** – מערכות RDBMS מאפשרות שיתוף נתונים אמיתי בין היחידות השונות בארגון, ללא צורך בבניית מבני נתונים יחודיים לכל פונקציה. שיתוף נתונים זה הוא הבסיס למערכות מידע אינטגרטיביות המסוגלות לשרת טוב יותר את הרמות ואת הפונקציות השונות בארגון.
- **אמינות נתונים (Data Integrity)** – מערכות RDBMS מכילות מספר מנגנונים שמטרתם הכוללת הינה שמירת אמינות בסיס הנתונים. בין מנגנונים אלה נציין את אלה: ניהול הקשרים בין הנתונים על ידי המערכת, יכולת הגדרת בדיקות תקינות באופן מרכזי, יכולת הפעלת פרוצדורות באופן אוטומטי בהתרחש אירוע כלשהו, רישום אוטומטי של יומן האירועים והפעלת מנגנון למחיקת העדכונים כאשר יש תקלה כלשהי.
- **זמינות נתונים (Data Availability)** – מערכת RDBMS יכולה לספק למשתמשים השונים כלים מתוחכמים לשליפה ועדכון. לפיכך, בסביבת עבודה המבוססת על בסיס נתונים טבלאי, ניתן להפוך במהירות ובפשטות נתונים גולמיים למידע הנדרש על ידי הארגון בתהליכי העבודה וקבלת ההחלטות. מערכות RDBMS מבוססות על שפת SQL שהיא שפת נתונים רבת-עוצמה, המשפרת באופן משמעותי את התפוקה של מהנדסי התוכנה ושל המשתמשים.

- **סטנדרטיזציה (Standartization)** – כל הנתונים מוגדרים באופן מרכזי בבסיס נתונים אחד ועל ידי פונקציה מרכזית שהיא מנהל בסיס הנתונים, ולא על ידי כל תוכנית יישום בנפרד. על כן ניתן בקלות יחסית לאכוף סטנדרטים של מבני נתונים, שמות נתונים, בדיקות תקינות וכד'. סטנדרטיזציה, או אחידות, מבטיחה יתר אמינות של בסיס הנתונים ומפשטת את השימוש בו.
- **גמישות לשינויים (Flexibility to Changes)** – קל יותר להתאים את בסיס הנתונים ואת תוכניות היישום המשתמשות בו לשינויים המתרחשים באופן שוטף בכל ארגון. העובדה שכל מבני הנתונים מוגדרים באופן חיצוני לתוכניות ושתוכניות אלו יכולות לגשת לבסיס הנתונים באמצעות תת-סכימה (View), מבטיחה רמה גבוהה יחסית של אי-תלות לוגית ואי-תלות פיסית.
- **אבטחת נתונים (Data Security)** – מערכות RDBMS מכילות מספר מנגנונים לאבטחת הנתונים בבסיס הנתונים. הן מאפשרות הגדרה ברורה למי מותר לעשות מה ועל איזה נתונים. מנגנון **הטבלה המדומה** יכול לתרום גם הוא לאבטחת הנתונים, כי הוא מאפשר הגדרה מאוד מתוחכמת של הנתונים שיוצגו למשתמשים השונים. המערכת גם כוללת מנגנון רישום של כל פעולה שנעשית על ידי מי ומתי, והדבר מאפשר לשחזר לצורך בירור את הפעולות שבוצעו בבסיס הנתונים.
- **תמיכה בעיבוד תנועות (Transaction Processing)** – מערכות RDBMS מבוססות על מנגנונים מתוחכמים התומכים במודל תנועות, כדי להבטיח שאוסף העדכונים המוגדר כתנועה יעדכן את בסיס הנתונים או שלא יעשה זאת. למנגנון זה חשיבות מרובה במערכות מידע מקוונות המעדכנות את בסיס הנתונים עם אוסף גדול מאוד של תנועות מקוונות. כשל במהלך ביצוע תנועה ואי השלמתה הראוי מחייב שחזור המצב כדי להבטיח את שלמות ואמינות בסיס הנתונים.
- **תמיכה בעדכון בו-זמני (Concurrent Update)** – מערכות DBMS מכילות מנגנונים מתוחכמים אשר תפקידם לוודא שעדכון בו-זמני על ידי מספר משתמשים לא יפגום באמינות בסיס הנתונים. מערכות אלו מכילות, בדרך כלל שירותי נעילה, איתור נעילה ללא מוצא וכד'.

חסרונות

כפי שניתן לצפות, למערכות RDBMS יש גם מספר חסרונות, ולא רק יתרונות. נסקור את העיקריות שבהם:

- **סיבוכיות** – מערכת RDBMS הינה מערכת תוכנה מרכזית השולטת על כל הנתונים ולכן היא מורכבת יחסית ודורשת מיומנות רבה בתפעולה. לתמיכה שוטפת בפעולות מערכת ניהול בסיסי נתונים גדולה אשר משרתת מצידה מערכות מידע מורכבות המטפלות במספר רב של משתמשים, דרושה בדרך כלל קבוצה של אנשי מקצוע, מנהלי בסיסי הנתונים.

- **עלות** – מערכת RDBMS עולה כסף רב. העלויות יכולות לנוע ממאות דולרים למערכות המיועדות לעבודה במחשבים אישיים, לעשרות אלפי דולרים בשרתים מחלקתיים ועד למאות אלפי דולרים במערכות מחשב גדולות.
- **משאבי חומרה** – מערכת RDBMS דורשת יותר משאבי חומרה (זיכרון, CPU ושטחי אחסנה בדיסק) מאשר מערכות לניהול קבצים.
- **רגישות לתקלות** – כל הנתונים של הארגון מנוהלים בבסיס הנתונים ועל כן תקלה יכולה לגרום להשבתה של פעילות הארגון, או לפחות של חלקים ניכרים ממנו.
- **מורכבות שחזור** – מערכת RDBMS מנהלת קשרים בין נתונים, בנוסף לנתונים עצמם. על כן, שיקום בסיס הנתונים לאחר תקלה יכול לגרור שחזור של נתונים פגומים בבסיס הנתונים עצמו ושחזור נתונים נוספים, בגלל הקשרים המורכבים בין הנתונים. מערכות RDBMS כוללות מנגנונים מתוחכמים לשחזור, אבל תיקון תקלה יכול בהחלט לגרור תיקון נתונים נוספים בגלל הקשרים המורכבים ביניהם.

סיכום

מערכות המידע מבטאות יחסי גומלין בין אנשים, נתונים, כללים לעיבוד הנתונים ומחשבים המעבדים את הנתונים. מטרת מערכת המידע היא לאחסן, לעבד ולאחזר נתונים, הן לצרכים תפעוליים של הארגון והן לתמיכה בקבלת החלטות. החלק הקריטי ביותר במערכת מידע הינו בדרך כלל החלק המטפל באחסון ובאחזור הנתונים, בעדכון ובהגנה על אמינותם ושלמותם.

בתחילת השימוש במחשבים ליישומים תפעוליים, כל תוכנית יישום טיפלה באופן ישיר בנתונים. כתוצאה מכך נוצרה כפילות נתונים רבה והמאמץ שהושקע בפיתוח יישומים היה רב. שיתוף נתונים בין יישומים שונים היה קשה מאוד וכמעט בלתי אפשרי. כתוצאה מכך התפתחו מערכות לניהול קבצים ששחררו את מהנדסי התוכנה מטיפול ישיר במבנה הפיסי של הנתונים. ואולם גם מערכות אלו חייבו ידע מסוים מצד המשתמש על צורת אחסון הנתונים ועל שיטות הגישה אליהם, על מבנה הרשומה וסדר השדות בה ועוד. עם זאת, הושגה התקדמות רבה בתחום אי-התלות הפיסית של תוכניות היישום בנתונים, אשר ביטאו במידה רבה יותר את התהליך הלוגי של הטיפול בנתונים.

המערכות לניהול בסיסי נתונים פותחו כדי לאפשר גישה קלה ונוחה יותר לנתונים. הן משמשות כממשק בין תוכנית היישום לבין המבנה הפיסי של בסיס הנתונים. המשתמש מציין באיזה נתונים הוא רוצה, מבלי שיצטרך לפרט היכן הם נמצאים וכיצד לגשת אליהם. חשוב להדגיש, שלמרות העובדה שמערכות DBMS אינן מערכות מידע, הן מהוות תשתית טכנולוגית הכרחית בדרך ליישום מערכות מידע.

יישום מוצלח של מערכות DBMS מחייב תכנון מפורט, תוך שימת דגש על עיצוב בסיס הנתונים, פיתוח מודולרי של היישומים ותמיכה טובה של אנשי תוכנה.

המעבר ממערכת ניהול קבצים למערכת DBMS, כלומר לבסיס נתונים משולב, מהווה שינוי משמעותי בתהליך עיבוד הנתונים בארגון. אין להתעלם גם מהמרכיב הכספי הכרוך בשינוי זה – צורך במומחים בנושא, עלות התוכנה וצריכת משאבים גדולה יותר. גורמי העלות ניתנים לזיהוי ולכימות בקלות יחסית לעומת גורמי התועלת. בין גורמי התועלת הישירים אפשר למנות את החיסכון בפיתוח יישומים וניהול משולב של נתונים, חיסכון באחזקה ובשינויים מקיפים, כאשר נדרש שינוי במערכת או בחומרה ועוד. תועלת נוספת היא העלאת רמת האמינות והזמינות של המערכת, וגמישות יחסית לשינויים בנתונים, ביישומים או בחומרה.

שאלות חזרה ותרגילים

שאלות חזרה

1. הסבר את ההבדלים העיקריים בין מערכת FMS לבין מערכת RDBMS.
2. הסבר את מרכיבי מודל ANSI/SPARC, את קשרי הגומלין בין מרכיבים אלה וכיצד הם תומכים בנושא אי-התלות הלוגית והפיסית.
3. הסבר מה תפקידי שפת DDL ושפת DML. חווה דעתך האם טוב שקיימת הפרדה בין שתי שפות אלו.
4. הצג סקירה קצרה של סוגי המשתמשים במערכת RDBMS, ומה הם הכלים הנדרשים לכל אחד מהם.
5. האם מערכות RDBMS מתאימות ליישומים גדולים ומורכבים בלבד? הסבר איזה שימושים ניתן לעשות במערכת RDBMS ביישומים שונים.
6. סקור בקצרה את מיגוון השירותים שמערכת RDBMS מספקת. השווה שירותים אלה מול השירותים של מערכת FMS.
7. נסה לחשוב על שפת התכנות המוכרת לך. איזה מהפקודות המוכרות לך היית משייך ל-DDL ואיזה ל-DML. האם אבחנה זו תקפה רק אם משתמשים במערכת RDBMS או שהיא מתאימה גם למערכות FMS?
8. הסבר מהי שפה מארחת וכיצד היא מתקשרת עם בסיס הנתונים. מה נדרש כדי ששפת תכנות כלשהי תוכל לשמש כשפה מארחת ושתוכל לגשת לבסיס נתונים כלשהו?
9. הסבר בקצרה את התפקידים הבאים ומה מטרתם בסביבת RDBMS: *מנהל בסיס הנתונים, משתמש קצה, מפתח יישומים, מנתח מערכות.*

10. קרא את המשפטים הבאים וענה בכן/לא:

א. רק במערכות DBMS הגדרת מבנה הנתונים הוא חיצוני לתוכנית היישום.

ב. שפת DDL משמשת את ה-DBA בלבד.

ג. מול כל תת-סכימה ניתן להגדיר מספר רב של סכימות.

ד. קטלוג המערכת מכיל את הנתונים של בסיס הנתונים.

11. הגדר בקצרה את המושגים הבאים: *בסיס נתונים, מערכת לניהול בסיס נתונים, אי-תלות בנתונים, סכימה, משתמש קצה.*

תרגילים

1. בחן את מערכות המידע השונות הפועלות בארגון בו הינך לומד או עובד. בדוק האם מערכות מידע אלו משתמשות במערכות ניהול קבצים או במערכות ניהול בסיסי נתונים. באיזה שפות תכנות משתמשים בארגון? האם מערכות אלו מנהלות מילון נתונים מרכזי, או שהגדרת הנתונים נמצאת בכל תוכנית יישום?

2. עליך לבנות יישום באינטרנט המאפשר לתושבי עירך לגשת לאתר של הספריה העירונית ולבדוק אם יש בספריה ספר מסוים, והאם הוא זמין להשאלה. תאר את מרכיבי היישום ואת שלבי העבודה שלו.

הערה: תוכל לבצע תרגיל זה באופן תיאורטי, אך רצוי שתבדוק היכן קיימת ספריה שברשותה אתר שניתן לפנות אליו באמצעות האינטרנט.

חלק ב'

עיצוב מודל הנתונים הטבלאי

חלק ב' של הספר עוסק בהצגת תהליך עיצוב מודל הנתונים הטבלאי עבור **מערכות מקוונות לעיבוד תנועות – OLTP**. כפי שראינו בחלק א' של הספר, כל מערכת RDBMS מנהלת אוסף טבלאות המכילות את הנתונים עצמם. בחלק זה של הספר נעסוק בשאלה כיצד מחליטים אילו טבלאות יכיל בסיס הנתונים, אילו עמודות תהיינה בכל טבלה, מה יהיו המפתחות של כל טבלה, איזה קשרים ננהל בין הטבלאות וכיצד נבטיח שאוסף הטבלאות הוא אכן אוסף יציב ונכון. נדגיש כאן שעבור סביבת מחסן הנתונים התפתחו שיטות עיצוב שונות. שיטות אלו הן מעבר למטרות ספר זה, והקורא המעוניין מופנה לספרו של המחבר **מחסני נתונים – עקרונות, ארכיטקטורה, עיצוב ויישום** שיצא לאור **בהוצאת הוד-עמי**.

חלק ב' מורכב משלושת הפרקים הבאים:

- **פרק 3 – ניתוח ועיצוב מודל הנתונים התפישתי**: פרק זה מציג את הנושא והמושג המרכזי של חלק ב' – מודל הנתונים. כפי שנראה, מודל הנתונים הוא אמצעי לתיאור המציאות שעבורה בונים את מערכת המידע. פיתוח מודל הנתונים מחולק לשלושה שלבים: פיתוח המודל התפישתי שמשוחרר מכל אילוצים של מערכת DBMS כלשהי; פיתוח המודל הלוגי שהוא ייחודי ומותאם למערכת DBMS, שתשמש לפיתוח המערכת ופיתוח המודל הפיסי העוסק בתרגום המושגים הלוגיים למבנה הפיסי הייחודי לכל מערכת. הפרק מתמקד בעיקר בפיתוח מודל הנתונים התפישתי ומציג את מיגוון השיקולים בעיצובו.

- **פרק 4 – פיתוח ועיצוב מודל הנתונים הטבלאי:** פרק זה סוקר את שלב עיצוב המודל הלוגי, שהינו השלב השני בעיצוב בסיס הנתונים. הפרק מציג את מודל הנתונים הטבלאי, הנפוץ ביותר כיום. מודל זה התפתח על רקע המורכבות הגדולה של מערכות DBMS ותיקות יותר שהתבססו על מודלים אחרים, כגון המודל ההיררכי והמודל הרשתי. הפרק סוקר את הרקע ההיסטורי של התפתחות המודל הטבלאי ומציג את מושגי היסוד על תורת הקבוצות, המהווה את הרקע התיאורטי שעליו מבוסס מודל הנתונים הטבלאי. לבסוף הוא סוקר את מרכיבי מודל הנתונים הטבלאי.

- **פרק 5 – נירמול נתונים:** פרק זה סוקר את תהליך נירמול הנתונים שמטרתו להבטיח שאוסף הטבלאות שהחלטנו לנהל בבסיס הנתונים הוא אוסף תקין וטוב ויש בו כפילות נתונים חיונית בלבד. הפרק מציג את המושג "תלות פונקציונלית" שבין עמודות שונות של אותה טבלה, ומסביר את השיטה שניתן להפעיל כדי לסלק את כפילות הנתונים מתוך הטבלאות.

- **פרק 6 – מתודולוגיה לעיצוב בסיס הנתונים:** פרק זה מציג מתודולוגיה לעיצוב בסיס הנתונים, אשר תחילתה בעיצוב מודל ישויות-קשרים ראשוני, והמשכה בתהליך איטרטיבי (חוזר על עצמו) של שילוב מודלים מקומיים עד לקבלת מודל ישויות-קשרים גלובלי. סיום התהליך הוא תרגום מודל ישויות-קשרים לאוסף הטבלאות של בסיס הנתונים.

פרק 3

ניתוח ועיצוב מודל הנתונים התפישתי

1. מבוא
2. מודל הנתונים (Data Model)
3. תהליך עיצוב מודל הנתונים
4. מודל ישויות-קשרים (Entity Relationship Data Model)
5. סיכום
6. שאלות חזרה ותרגילים

מבוא

כל מערכת מידע מנהלת אוסף כלשהו של נתונים. בפרק זה נעסוק בשאלה כיצד ניתן לתאר אוסף זה באופן שאינו תלוי בצורה שבה הם ינוהלו במערכת הממוחשבת – באמצעות מערכת ניהול קבצים, או באמצעות מערכת ניהול בסיסי נתונים כזו או אחרת. בפרק זה נציג את מודל הנתונים התפישתי ונראה כיצד מעצבים אותו. נושא זה הוא בעל חשיבות רבה בניתוח ועיצוב מערכות מידע בכלל ובעיצוב בסיסי נתונים בפרט.

למעשה, ניתן לנסות ולתאר את אוסף הנתונים הדרושים ליישום מסוים באמצעות מלל בשפה טבעית. אנשים אוספים ומציגים מידע באמצעות השפה הטבעית. הם כותבים ספרים, מכתבים, תזכירים וכד׳ כדי לצבור ולהעביר את המידע לאנשים אחרים. עם כל עוצמת הביטוי של השפה הטבעית והיכולת שלה לתעד ולשמש ככלי להעברת מידע, היא אינה בהכרח הכלי היעיל ביותר לתיאור כל סוגי הרעיונות ולייצוג המידע. כמו באימרה המפורסמת על ״תמונה אחת השווה אלף מילים״, אנו רואים שבני האדם מחפשים ללא הרף דרכים טובות ויעילות יותר לייצוג מידע. למשל, מידע גיאוגרפי עדיף לייצג באמצעות מפות, מידע מתמטי ניתן לייצוג מדויק ונוח בהרבה באמצעות אוסף נוסחאות מאשר במלל, ואילו מעגל חשמלי ניתן לתיאור נוח באמצעות סכימה חשמלית. במשך השנים פותחו כלים ורעיונות המאפשרים ייצוג נוח ואמין יותר של מידע בתחומים שונים. בפרק זה נציג שיטה נוספת ייעודית לייצוג מידע על מבנה נתונים – **תרשימי ישויות-קשרים**.

בעת הקמת מערכת מידע ממוחשבת אנו זקוקים לכלים מדויקים לתיאור, ייצוג וטיפול בנתונים ובמידע. אחד הכלים האלה הוא מודל הנתונים. **מודל הנתונים** (Data Model) מאפשר לנו לתאר את המציאות, לארגן ולייצג את המידע בצורה המובנת גם לבני אדם וגם למחשבים.

מטרת פרק זה להציג מספר מושגי יסוד בעיצוב מודלים של נתונים ובנייתם. נושא זה קיבל יתר תוקף עם הופעת מערכות לניהול בסיסי נתונים, אולם ניתן היום להתייחס לפעולה זו ללא תלות בבסיסי נתונים. הסיבה העיקרית לדיון המוקדש למושג זה היא שזהו הכלי שבאמצעותו מבינים ומנתחים את הבעיה שעומדת לפתרון, מבינים את המסגרת הכללית של הנתונים הדרושים לפתרון הבעיה ובסופו של דבר גם מבינים כיצד **לעצב** (Design) את בסיס הנתונים.

בפרק זה

- נסקור בקצרה סוגי מודלים שונים שאנו משתמשים בהם בחיי היום-יום, ונתמקד במודל הנתונים כאחד מכלי העבודה העיקריים של מנתח המערכות ומעצב בסיס הנתונים.
- נסקור את תהליך עיצוב מודל הנתונים. זהו תהליך תלת-שלבי המתחיל בעיצוב המודל התפישתי המשוחרר מהתייחסות לאילוצי תוכנה/חומרה/מערכת DBMS, ועובר לשלב העיצוב הלוגי, המתייחס למערכת DBMS שבה ימומש המודל. לסיום, אנו מגיעים לשלב העיצוב הפיסי העוסק בשיקולים העוסקים בהקצאת שטחי הדיסקים עבור הטבלאות השונות, האינדקסים וכד׳.

- נציג את מודל ישויות-קשרים שהינו אחד המודלים התפישתיים הנפוצים. המודל עושה שימוש במושגים כגון ישויות, קבוצות ישות, תכונות של קבוצה, מפתחות, היררכיות סמנטיות לבניית קבוצות וקשרים בין קבוצות. בסופו של דבר, נציג את תרשים ישויות-קשרים כתוצר העיקרי של תהליך העיצוב התפישתי.
- נסקור בקצרה את שלושת המודלים הלוגיים הידועים: המודל ההיררכי, המודל הרשתי והמודל הטבלאי.

מודל הנתונים (Data Model)

מערכת המידע מהווה תיאור מסוים של מציאות, או של עולם ממשי כלשהו. לתיאור זה מקובל לקרוא בשם "מודל של המציאות". על ידי בניית המודל של המציאות אנו מסוגלים לבנות מערכת מידע שבסופו של דבר תתמוך בניהול המציאות, תספק מידע על המציאות ותאפשר לארגון לתפקד באופן שוטף ולקבל החלטות על סמך מידע זה.

בחיי היום-יום אנו משתמשים במודלים רבים: מודלים אלגבריים, מודלים פיסיקליים, מודלים כלכליים, מודלים טופולוגיים, סכימות חשמליות, מפות של קווי אוטובוס ועוד. מכל הדוגמאות הללו אנו לומדים שהמודל מהווה **הפשטה** (Abstraction) מסוימת של הבעיה המתוארת ומתאים למטרות מסוימות. מפת קווי האוטובוס יכולה לסייע לנוסע בתחבורה ציבורית בתשובה לשאלה כגון "איזה אוטובוס עובר בתחנה A ומגיע לתחנה B". נוסחאות במקרו-כלכלה משמשות את מקבלי ההחלטות ככלי תכנון כדי לחזות לדוגמה, אם כי לא במדויק, מה יהיה שיעור החיסכון במשק כתוצאה מעליה או ירידה בשיעור הריבית. באופן דומה, מודל הנתונים יאפשר לנו לתאר את מבנה הנתונים הדרוש ליישום מסוים.

נחזור לדוגמה של המכללה שבה עסקנו ונראה שהמציאות שבה פועלת המכללה כוללת מספר גדול מאוד של **אובייקטים ואירועים**. בין האובייקטים האלה נוכל לציין סטודנטים, מרצים, בניינים, כיתות, קורסים ומסלולי לימודים. כפי שנראה בהמשך, לרוב האובייקטים האלה יהיה ביטוי כלשהו במערכת המידע. הם מהווים חלק מהעולם הממשי, המציאות, ולכן הם יהיו גם חלק מהמודל שנבנה במסגרת מערכת המידע. במציאות קורים כל הזמן אירועים, כמו לדוגמה: סטודנט משנה כתובת, מרצה עולה בדרגה, קורס חדש נפתח או סטודנט נבחן בקורס ומקבל ציון. כל האירועים האלה יביאו לידי שינוי כלשהו במערכת המידע, שיתבטא בדרך כלל על ידי שינוי בנתונים.

מודל הנתונים הוא כלי תפישתי (Conceptual), כלומר רעיון, המשמש לתיאור המציאות, לארגון וייצוג נתונים והקשרים ביניהם. זהו **כלי הפשטה** המאפשר לראות את "היער" (מה משמעות הנתונים) לעומת "העצים" ("ים" הערכים השונים שהנתונים יכולים לקבל).

מודל הנתונים הוא מסגרת לוגית המתארת את הישויות, התכונות והקשרים המייצגים את המציאות. המודל קובע את המבנה ואת השמות הלוגיים של המרכיבים השונים. לתוך מסגרת זו יוצקים בהמשך את התוכן, שהוא הערכים השונים שהנתונים מקבלים. שינוי ערך של נתון כלשהו מייצג אירוע שחל במציאות, אולם השינוי אינו משפיע על המודל,

אלא רק על תוכנו ברגע נתון. בהיותו מסגרת לוגית, משמש המודל ככלי תפישתי המאפשר קיום תקשורת בין מנתחי המערכת, המשתמשים והמחשב.

מודל הנתונים משמש אותנו בשלב עיצוב בסיס הנתונים כדי להגדיר במדויק את מבנה הנתונים הדרוש למערכת מידע מסוימת. **מודל הנתונים** הוא כלי לעיצוב בסיס הנתונים בדיוק כמו ש**תרשימי זרימת נתונים** – DFD (Data Flow Diagram) ו**פסידו-קוד** (Pseudo-code) הם כלים לעיצוב התוכנה. במובן מסוים ניתן להסתכל על מודל הנתונים כארגז כלי העבודה של מעצב בסיס הנתונים. כשם שארגז כלי העבודה מכיל את כלי העבודה השונים שבאמצעותם בעל המלאכה מבצע את עבודתו, כך גם מודל הנתונים מכיל את המרכיבים השונים שמעצב בסיס הנתונים משתמש בהם לניתוח המציאות, בונה את המודל של המציאות ובסופו של דבר – מממש את המודל במערכת לניהול בסיסי נתונים כלשהי.

עכשיו נוכל לבחון את ההגדרה הפורמלית של מודל הנתונים.

מודל הנתונים **Data Model**	***מודל הנתונים*** **מוגדר ככלי תפישתי המשמש לתיאור המציאות באמצעות אוסף של כללים המתארים את מבנה הנתונים, האילוצים שעל נתונים אלה לקיים, האופרטורים לשליפה ושינוי הנתונים.**

אם נקרא למודל הנתונים בשם M, אזי ניתן לתאר אותו על ידי הביטוי הבא:

$$M = \{S, C, O\}$$

המודל מורכב משלושה מרכיבים, המסומנים באותיות **S**, **C**, **O**:

S אוסף כללים המגדירים את ה**מבנה** (**S**tructure), את הישויות, את התכונות הרלוונטיות של כל ישות כזו ואת הקשרים ביניהן. לדוגמה, המודל יכול להכיל שתי ישויות: סטודנטים וקורסים. הישות קורס יכולה להכיל תכונות, כגון: שם הקורס, מספר הקורס, מספר נקודות זכות ועוד. כמו כן, קיימים קשרים בין סטודנטים וקורסים, כמו לדוגמה: סטודנט **לומד** בקורס.

C אוסף כללים המגדירים את ה**אילוצים** (**C**onstraints) החלים על המבנה שהוגדר. להלן מספר דוגמאות לאילוצים שיכולים להופיע: לא ייתכנו שני קורסים בעלי אותו מספר קורס, מספר נקודות הזכות לא יעלה על חמש לכל קורס, סוג קורס יכול להיות שיעור, מעבדה או סמינר, מרצה יכול ללמד לכל היותר שלושה קורסים במקביל ועוד.

O אוסף ה**אופרטורים** (**O**perators) שניתן להפעיל על המבנה שהוגדר. המודל צריך להכיל אופרטורים המאפשרים לשלוף נתונים ולשנות את תוכנם. לדוגמה, על המודל להכיל אופרטורים שיאפשרו שליפה של כל הקורסים המקנים שלוש נקודות זכות ומעלה, או אופרטור המאפשר לשנות כתובת של סטודנט וכד'.

המרכיבים של המבנה והאילוצים מייצגים את ההיבט **הסטטי** שיבוא לידי ביטוי בהגדרת בסיס הנתונים. האופרטורים מייצגים את ההיבט **הדינמי** שיבוא לידי ביטוי בתוכניות היישום המבצעות פעולות שונות על הנתונים המנוהלים בבסיס הנתונים.

תהליך עיצוב מודל הנתונים

שאלה מרכזית הנשאלת בעת שמנתחים ומקימים מערכת מידע היא: כיצד לעצב את מבנה בסיס הנתונים, כך שישרת בצורה הטובה ביותר את תוכניות היישום השונות? נזכיר שבסיס נתונים אינו משרת יישום אחד בלבד, אלא אמור לשמש את כלל היישומים הפועלים בארגון.

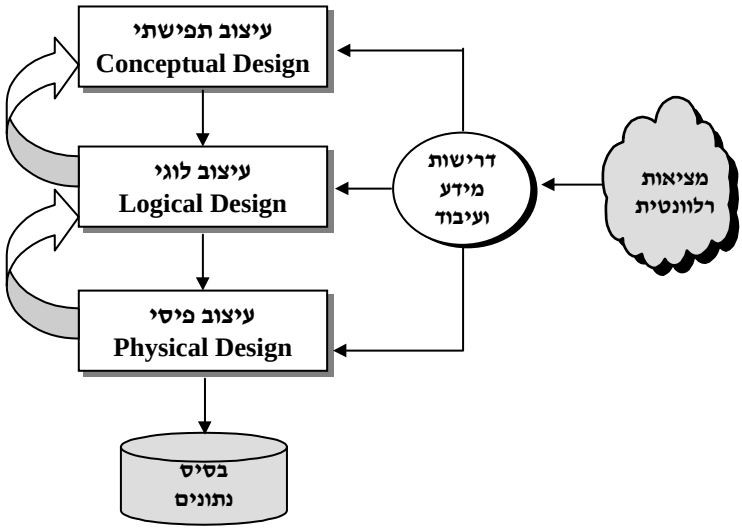
המציאות שעבורה אנו מקימים את המודל יכולה להכיל מאות ולעיתים אלפי **אלמנטים של נתונים** (Data Elements) ואחת הבעיות המרכזיות היא, כיצד לארגן אותם במבנה לוגי אחד. מבנה לוגי זה חייב להיות יציב ככל הניתן, ועליו לייצג בצורה הטובה ביותר את המבנה הפנימי של הנתונים. המבנה צריך להיות גמיש מספיק, כדי לאפשר ביצוע שינויים ללא זעזוע קשה מדי במערכות המידע שכבר פועלות.

במהלך השנים התגבשה תפישה המתייחסת לתהליך הניתוח, התכנון והעיצוב של בסיס הנתונים כאל תהליך המורכב משלושה שלבים:

- **שלב העיצוב התפישתי** (Conceptual Design) עוסק בעיצוב מבנה המידע "הטהור", המשוחרר מכל האילוצים הנובעים משימוש בחומרה ובתוכנה. הקלט לשלב זה הוא אוסף כל הדרישות, כפי שבאו לידי ביטוי במהלך ניתוח הדרישות.
- **שלב העיצוב הלוגי** (Logical Design) עוסק בתרגום המבנה התפישתי שהתקבל בשלב הקודם לסכימה של מערכת DBMS מסוימת. בשלב זה נלקחים בחשבון האילוצים של מערכת זו.
- **שלב העיצוב הפיסי** (Physical Design) עוסק בתרגום הסכימה למבנה פיסי, הלוקח בחשבון אילוצי ביצוע ויעילות, נפחי אחסון, שיטות גישה לנתונים (אינדקסים, מצביעים), הקבצת נתונים על הדיסק (Clusters) ועוד.

התהליך הינו תהליך איטרטיבי (חוזר על עצמו), כך שמעבר לשלב מתקדם יותר – יכול להשפיע על החלטות שהתקבלו בשלב מוקדם יותר.

כל אחד משלושת השלבים הוא בעל חשיבות מרובה, ויש לתת לו את תשומת הלב המתאימה. נרחיב את הדיון בכל אחד משלושת השלבים.



תרשים 3.1: תהליך עיצוב בסיס הנתונים.

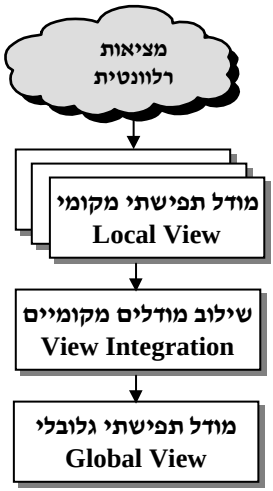
המודל התפישתי (Conceptual Model)

מודל זה מייצג את ההבנה והידע שלנו על **המציאות הרלוונטית לבעיה**. מטרת שלב זה לארגן את הנתונים ואת הקשרים ביניהם לצורה שמובנת על ידי מנתחי המערכת ומשתמשיה העתידיים. המודל התפישתי נטול כל אילוצים שיכולים לנבוע ממערכת DBMS כלשהי, אילוצי חומרה או כל אילוץ טכנולוגי אחר. תהליך בניית המודל התפישתי הינו בדרך כלל תהליך איטרטיבי באופיו (חוזר על עצמו), כך שכל איטרציה מקרבת את המודל המתקבל למודל התואם את המציאות הארגונית. לעיתים, הבעיה שמנסים לפתור גדולה בהיקפה ולכן, מקובל לפתור אותה בשלבים. בדרך זו נבנה בכל שלב **מודל תפישתי מקומי** (Local Conceptual Model/View) המייצג רק חלק מהבעיה. לאחר סיום בניית המודלים המקומיים נבצע את תהליך **שילוב המודלים המקומיים** (View Integration) לקבלת **מודל תפישתי גלובלי** (Global Data Model/View), כפי שמוצג בתרשים 3.2.

תפקיד תהליך ניתוח הנתונים הוא לבנות מודל של המציאות הרלוונטית, המהווה חלק קטן יחסית של המציאות האמיתית – **העולם הממשי**. חלק ניכר ממאמץ בניית מודל הנתונים התפישתי מוקדש ל"צמצום" המציאות המכילה מאות ואלפי אלמנטים, למציאות רלוונטית. **מציאות רלוונטית** זו תכיל רק את הנתונים שהארגון מגיע למסקנה שהתועלת באיסופם, אחסונם וניהולם עולה על העלות הכרוכה בכך. אנו בונים את המודל כדי לתאר את המציאות הרלוונטית, שהינה רק חלק מהמציאות, ועבורה מוקמת מערכת המידע.

כדי להציג את ההבדל בין המציאות האמיתית לבין המציאות הרלוונטית, נתייחס לניתוח מערכת במפעל תעשייתי כלשהו. המציאות, מבחינת המפעל, מורכבת ממספר רב של אובייקטים הכוללים: מוצרים, חומרי גלם, לקוחות, ספקים, הזמנות, שרטוטים, הוראות הרכבה, מכונות, עובדים, תקציבים, מתחרים, תהליכי ייצור, מלאים ועוד.

נוסף למידע שמקורו במפעל, ההנהלה מעוניינת לקבל מידע גם בנושאים אחרים, כגון תהליכי הייצור של המתחרים, תחזית הרכישות של הלקוחות, המצב הפיננסי של לקוחות פוטנציאלים, מדיניות הריבית, מצב שער המניה של מתחרים ועוד. כל אלה הם חלק מהמציאות בה פועל הארגון, אולם לא כל הנתונים האלה יוכנסו למערכת המידע מסיבות שונות, שביניהן עלות איסוף גבוהה, כמויות גדולות של נתונים, חוסר ודאות ועוד.



תרשים 3.2: תהליך בניית המודל התפישתי הגלובלי.

שני מודלים תפישתיים מקובלים בתחום זה:

- **מודל ישויות-קשרים** –**ER** (Entity Relationship), שפותח על ידי Peter Chen [CP66].
- **המודל הסמנטי** – **SDM** (Semantic Data Model) שפותח על ידי Hammer, McLeod [HM81].

מודל ישויות-קשרים מוצג בהרחבה בהמשך, יחד עם המתודולוגיה לעיצוב המודל התפישתי.

המודל הלוגי (Logical Model)

המודל הלוגי נגזר מהמודל התפישתי. מודל זה מותאם למערכת DBMS מסוימת ומכיל אילוצים שונים הנובעים מהצורה שבה מפתחי מערכת DBMS בחרו לייצג את המודל. מקובל לכנות מודל זה גם בשם **סכימה** (Schema), על פי ההגדרה של ANSI/SPARC. המודל הלוגי הוא למעשה הסכימה של המציאות הרלוונטית כפי שהיא מיוצגת במערכת DBMS מסוימת.

במהלך השנים הוגדרו מודלים לוגיים שונים, אולם רק שלושה מהם זכו לתפוצה נרחבת ויושמו במערכות DBMS מסחריות: **המודל ההיררכי, המודל הרשתי והמודל הטבלאי.** המודל הטבלאי הפך עם השנים לנפוץ ביותר, ולו יוקדש הפרק הבא הכולל תיאור פורמלי שלו. חשוב לציין שבמקביל למערכות DBMS שהתבססו על אחד משלושת המודלים

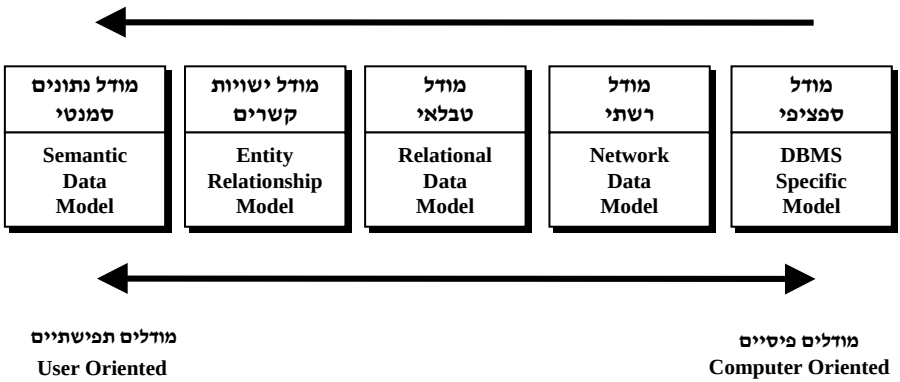
הקלאסיים, התפתחו מערכות מסחריות המשתמשות בשילוב כלשהו בין המודלים האלה או רק בחלק ממודל מסוים.

המודל הלוגי עומד בבסיסה של כל תוכנה לניהול בסיסי נתונים. על כן באופן טבעי חלים עליו אילוצים מסוימים ומגבלות, אשר נובעים בעיקר מהצורך לטפל במבני הנתונים בצורה יעילה, נוחה ומובנת למחשב. המעבר מהמודל התפישתי למודל הלוגי דורש ביצוע המרות מסוימות. המעבר ממודל תפישתי למודל טבלאי קל באופן משמעותי לעומת המעבר לכל אחד מהמודלים האחרים. משמעות ה"מעבר" תובהר בהמשך.

המודל הפיסי (Physical Model)

מודל לוגי ניתן ליישום במערכת DBMS במספר מבנים פיסיים שונים. לדוגמה, נוכל לבנות אינדקס על טבלה מסוימת, או נוכל לוותר עליו. דוגמה אחרת היא פיצול הטבלה למספר מחיצות (Partitions) פיסיות המתאימות במיוחד לניהול טבלאות גדולות מאוד. למבנים פיסיים אלה משמעות רבה מבחינת ביצועי המערכת, אולם אין להם השפעה על המודל הלוגי עצמו. בדוגמה זו, קיום או אי-קיום האינדקס או פיצול הטבלה למספר מחיצות, אינו משנה את המודל הלוגי אלא רק את צורת המימוש שלו באמצעי האחסון שקיימים במערכת הממוחשבת שבה פועלים. מנהל בסיס הנתונים, בשיתוף עם מנתח המערכת, חייב לבנות את המודל הפיסי המתאים ביותר לדרישות העיבוד ולמודל הלוגי. תוכניות היישום והמשתמשים במערכות RDBMS כמעט ואינם חשופים למודל הפיסי. מערכת RDBMS היא זו שמשתמשת במודל הפיסי כדי לנהל את הנתונים, לאפשר גישה אליהם ולאחזר אותם בצורה היעילה ביותר.

מערכות DBMS מהדור הראשון (כגון IMS או Total) היו מבוססות על מודל לוגי בעל קשר ותלות חזקים מאוד למודל הפיסי. רוב האילוצים במודל הלוגי נבעו מהזיקה הגבוהה בינו לבין המודל הפיסי. במערכות מתקדמות המבוססות על המלצות CODASYL, כגון IDMS, פחתה זיקה זו ונוצרה הפרדה ברורה יותר בין המודל הלוגי לבין המודל הפיסי. רק עם הופעת המודל הטבלאי ומערכות RDBMS המבוססות עליו, נוצרה הפרדה ברורה וחדה בין המודל הלוגי לבין המודל הפיסי. תרשים 3.3 מראה את התפתחות המודלים על ציר הזמן מימין לשמאל, וכיצד המודלים הפכו תלויים פחות במבנה הפיסי.



תרשים 3.3: התפתחות המודלים העיקריים.

מודל ישויות-קשרים
(Entity Relationship Data Model)

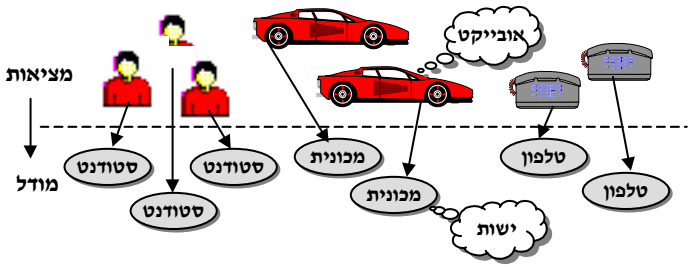
מודל ישויות-קשרים הינו מודל תפישתי המאפשר בניית **מודל של המציאות** באופן גרפי ותוך שימוש במספר מושגים קטן. מודל ישויות-קשרים משוחרר מכל היבט הקשור למחשב ולצורה בה הוא מיושם במודל לוגי כלשהו. המודל פותח והוצג לראשונה על ידי Dr. Peter Chen בשנת 1976, ומשמש מאז כלי חשוב בעבודת מנתח המערכת בעת בניית מודל נתונים.

ישויות

המושג הבסיסי והחשוב ביותר של **מודל ישויות-קשרים** הוא **ישות**.

ישות **Entity**	***ישות*** **מייצגת במודל אובייקט ממשי או מופשט, שיש לו קיום במציאות והוא בעל משמעות ועניין בהקשר מסוים.**

כפי שניתן לראות מההגדרה, הישות מייצגת כל אובייקט או רעיון שיש לו קיום משל עצמו במציאות והוא מעניין את הארגון בהקשר של בעיה מסוימת. בהמשך נבחין בין המונח "אובייקט" המתייחס לדבר כלשהו במציאות, לבין המונח "ישות" המייצג את האובייקט במודל התפישתי.



תרשים 3.4: המעבר מאובייקטים לישויות.

המעבר מהמונח "אובייקט" למונח "ישות" הוא למעשה המעבר מהמציאות עצמה אל המודל המייצג אותה. בתיאור הגרפי של המודל נשתמש באליפסה כדי לתאר ישות.

כל סביבה עתירה באובייקטים שהם ייחודיים לה. האובייקט יכול להיות ממשי או מופשט, כפי שנראה בדוגמאות אלו:

- **עצם** – כמו בניין, ספר, מכונית, מכונה במפעל, פריט במלאי.
- **גוף חי** – כמו עובד במפעל, סטודנט במכללה, חולה בבית חולים.
- **מושג מופשט או רעיון** – כמו קורס במכללה, מבחן נהיגה, טיסה, תעסוקה.
- **אירוע** – כמו דיווח נוכחות במפעל, תנועה בחשבון הבנק שלנו, כניסת פריט למלאי.

במציאות הסובבת אותנו יש מספר כמעט אינסופי של אובייקטים, ולכן השאלה היא כמובן איזה אובייקטים מעניינים ואיזה לא. זה אחד הנושאים המרכזיים שמעצב מודל הנתונים מטפל בהם, והוא כמובן תלוי במידה רבה בדרישות הפונקציונליות של מערכת המידע שמקימים.

במודל התפישתי קיימת אבחנה בין ישות "חזקה" לבין ישות "חלשה".

ישות חזקה / חלשה **Strong/Weak Entity**	***ישות חזקה*** **מוגדרת כישות שקיומה העצמאי במודל אינו תלוי בקיומה של ישות אחרת.** ***ישות חלשה*** **מוגדרת כישות שקיומה העצמאי במודל מותנה בקיומה של ישות אחרת.**

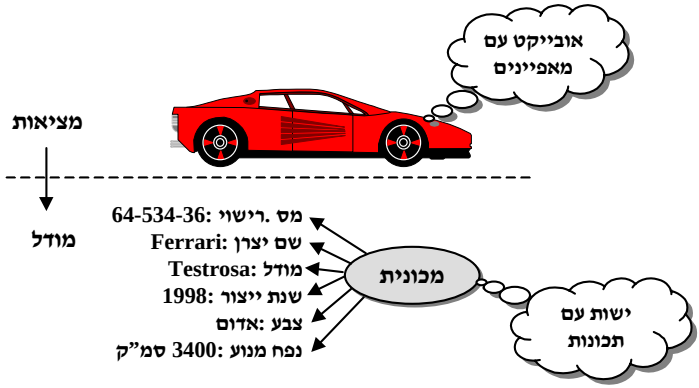
לדוגמה, הישויות עובד, סטודנט, ספר, פריט יכולות להיחשב ישויות חזקות. דוגמה לישות חלשה היא הישות ציון, כי קיומה מותנה בקיום הישות סטודנט והישות קורס. לא ייתכן שהישות ציון תתקיים, אם אחת משתי הישויות האחרות אינה קיימת. דוגמה אחרת היא הישות ילד של העובד. בהנחה שאנו מעוניינים לנהל מידע על הילדים של העובדים, ברור לנו שזו ישות חלשה, כי קיומה מותנה בקיומה של הישות עובד.

תכונה (Attribute)

תכונה **Attribute**	***תכונה*** **מוגדרת כמאפיין כלשהו של אובייקט שהוא בעל משמעות ועניין במודל.**

לכל אובייקט במציאות יש מספר רב של מאפיינים (Characteristics). למשל, לאובייקט סטודנט יש מאפיינים כגון מספר זהות, שם, כתובת, תאריך לידה, גובה, צבע עיניים, סוג דם, צבע שיער, משקל וכד'. בדומה לשאלה איזה אובייקטים רלוונטיים למודל ואיזה לא, נוכל להחיל שאלה זו גם לגבי מאפייני האובייקט. רק מאפיינים רלוונטיים יהפכו לתכונות של ישות. תכונות הישות הם האלמנטים הבסיסיים ביותר שמעניינים אותנו בהקשר של המודל התפישתי. המעבר ממאפיין של אובייקט לתכונה של ישות הוא כמו המעבר מאובייקט לישות, כלומר: מעבר מהמציאות עצמה אל מודל המייצג אותה.

מבין מאפייני האובייקט, רק חלקם יהפכו לתכונות של ישות לאחר שנמצא שמאפיינים אלה הם החשובים בהקשר מסוים. למשל סביר להניח שבמערכת מינהל אקדמי נרצה להתייחס למאפיינים כגון שם, כתובת, תאריך לידה, אבל אין חשיבות למאפיינים כגון צבע עיניים, גובה, סוג דם, משקל וצבע שיער. אם בשלב כלשהו יוחלט שיש צורך לטפל גם בבדיקות דם תקופתיות של הסטודנטים, נוכל להפוך את המאפיין סוג דם של סטודנט לתכונה של הישות סטודנט.



תרשים 3.5: מיפוי המציאות למודל של המציאות.

מקובל להבחין בין תכונה פשוטה לבין תכונה מורכבת.

תכונה פשוטה/מורכבת **Simple/Compound Attribute**	***תכונה פשוטה*** **מוגדרת כתכונה המכילה מרכיב אחד בלבד ואינה ניתנת לחלוקה נוספת.** ***תכונה מורכבת*** **מוגדרת כתכונה המכילה מספר רכיבים וניתן לחלק אותה למרכיביה.**

לדוגמה, התכונות מין, שכר לימוד, שם קורס או מספר נקודות זכות הן תכונות פשוטות שאינן ניתנות לחלוקה נוספת שתהיה בעלת משמעות לוגית כלשהי. אם נחלק את התכונה שכר לימוד נקבל את הספרות הבודדות המרכיבות את הסכום, ואין להן כל משמעות בצורה זו. לעומת זאת, התכונות כתובת הסטודנט ושם המרצה הן תכונות מורכבות, כי ניתן לחלק אותן ולקבל חלקים בעלי משמעות. את כתובת הסטודנט ניתן לחלק לעיר, מיקוד, שם רחוב ומספר בית. את שם המרצה ניתן לחלק לשם משפחה ושם פרטי. לפעמים יש צורך לנהל את התכונה ברמה הנמוכה ביותר על פי מרכיביה הפשוטים, ולעיתים ניתן לנהל את התכונה כתכונה מורכבת.

מקובל להבחין בין נתונים מובנים לנתונים לא מובנים.

נתון מובנה **Structured Data**	***נתון מובנה*** **הוא נתון בעל טיפוס נתונים (Data Type) בסיסי ובעל אורך ידוע.**

דוגמאות לנתונים מובנים: מחיר פריט, שם לקוח, משך שיחת טלפון, שם מו"ל של ספר, כתובת ספק הציוד, יתרה במלאי של פריט, כמות שהוזמנה, היתרה בחשבון הבנק, תאריך התנועה האחרונה בחשבון הבנק, שם סטודנט, ציון בקורס, שכר לימוד וכד'. לתכונות אלו אנו יכולים להגדיר טיפוס נתונים ומספר תווים הדרושים לתיאור שלהן.

נתבונן במספר דוגמאות כדי להסביר זאת:

- ציון בבחינה: נתון בעל טיפוס נתונים מסוג מספר שלם שיכול להכיל מספר בין 0 ל-100.
- שכר לימוד: נתון בעל טיפוס נתונים מסוג מספר עשרוני (decimal) שיכול להכיל שתי ספרות אחרי הנקודה העשרונית.
- שם סטודנט: שם סטודנט הוא בעל טיפוס נתונים מסוג מחרוזת תווים (Character). האורך של המחרוזת צריך להיקבע בהתאם למציאות הרלוונטית. ניתן למשל לקבוע שהאורך יהיה 40 תווים, כי סביר להניח שיספיק עבור רוב שמות הסטודנטים. אם האורך לא יספיק, שם הסטודנט יקוצץ ורק 40 התווים הראשונים יוכנסו לקובץ. כפי שנראה בהמשך, מערכות RDBMS המודרניות מאפשרות הגדרה של טיפוס נתונים מסוג מחרוזת משתנה (Varchar) כדי לחסוך במקום אחסון כאשר המחרוזת קצרה ולאפשר גם מחרוזות ארוכות. תכונה בעלת טיפוס נתונים Varchar תוכל להכיל מחרוזת תווים באורך משתנה עד אורך מקסימלי כלשהו, שחובה להגדירו מראש.

ניתן להתייחס לתכונה כאל אלמנט אטומי. זו למעשה הרמה הנמוכה ביותר של הנתון שיש לה משמעות עבור הארגון. לדוגמה, לאות השלישית בשם הסטודנט אין חשיבות בפני עצמה, אלא רק כחלק משם הסטודנט. באופן דומה, לספרה השנייה בציון הבחינה אין כל משמעות בפני עצמה.

נתון לא מובנה **Unstructured Data**	***נתון לא מובנה*** **מוגדר כנתון שאינו בעל טיפוס נתונים בסיסי ואין לו אורך ידוע מראש.**

לדוגמה, ניתן להגדיר את תיאור הקורס כתכונה של הישות קורס. נניח שתיאור הקורס הוא מסמך HTML שיכול להכיל רצף כלשהו של תווים, מספרים, תמונות, צליל וגרפיקה. במצב זה לא ניתן להגדיר לתכונה שם קורס טיפוס נתונים בסיסי, וגם לא ברור מראש מה יהיה האורך שלו. דוגמה אחרת יכולה להיות התכונה תמונת הסטודנט של הישות סטודנט. התמונה היא רצף כלשהו של סיביות שמתורגם על ידי תוכנה מסוימת לתצוגת דמות על המסך או במדפסת. בדרך כלל נתונים בעלי טיפוס נתונים כזה נשמרים כרצף של סיביות.

חלק מתכונות הישות יכולות להיות מתוארות על ידי נתונים מובנים וחלקן – על ידי נתונים לא מובנים. לישות סטודנט ניתן להגדיר תכונות המתוארות על ידי נתונים מובנים, כגון מספר סטודנט, שם סטודנט, כתובת מגורים, תאריך לידה. אפשר להגדיר עבור הישות סטודנט גם תכונות המתוארות על ידי נתונים לא מובנים, כגון תמונת הסטודנט, תצלום של קורות חיים, תצלום של תעודת בגרות וכד'. לישות מרצה ניתן להגדיר תכונות המתוארות על ידי נתונים מובנים, כגון מספר זהות, שם מרצה, תואר אקדמי, חלקיות משרה וכד'. לישות פריט ניתן להגדיר תכונות המתוארות על ידי נתונים מובנים כגון מספר קטלוגי, תיאור פריט, יחידת מידה, מחיר הפריט ותכונות המתוארות על ידי נתונים לא מובנים כגון דף מידע על הפריט ותרשים סכמתי של המעגלים החשמליים.

טכנולוגיית המחשוב עסקה בשנותיה הראשונות כמעט אך ורק בניהול נתונים מובנים. רק בשנות ה-90 ובעיקר עם הרחבת השימוש במחשב האישי לצרכים גרפיים, ובמיוחד עם הופעת האינטרנט, התרחב מאוד השימוש בנתונים לא מובנים. בשנים האחרונות החלו גם אנשי טכנולוגיית בסיסי הנתונים לתת את דעתם לסוג זה של נתונים. הפתרונות הפשוטים יותר היו שימוש בטיפוס נתונים חדש מסוג BLOB (Binary Large Object) – התייחסות אל האובייקט כאל רצף סיביות, שהמערכת יודעת לאחסן ולאחזר אותו. עם הזמן החלה להתפתח טכנולוגיה של **מערכות לניהול בסיסי נתונים מונחי אובייקטים – ODBMS** (Object Oriented DataBase Management Systems), שמכירה מספר רב יותר של תכונות אובייקט ומסוגלת לפעול עליו במספר רב יותר של אופרטורים.

למרות החשיבות של הנתונים הלא מובנים, הטיפול בהם במסגרת הספר אינו רב ומתרכז בעיקר בפרק 18 המציג את מערכות ODBMS. הספר ממוקד רובו ככולו בניהול ועיבוד נתונים מובנים, שהוא נושא חשוב ומורכב בפני עצמו.

ערך של תכונה (Attribute Value)

ערך **Value**	**ערך של תכונה מוגדר כתוכן התכונה בנקודת זמן מסוימת.**

לכל תכונה של ישות יש **ערך** (Value) כלשהו. לדוגמה, לתכונה מספר זהות יכול להיות ערך כגון 234512387, לתכונה תאריך יכול להיות ערך כמו 23/10/1985. מקובל לקרוא לערך של תכונה בשם **נתון** (Data), המציין עובדה גולמית ובסיסית המתייחסת לישות. כפי שניתן להבין מתוך ההגדרה, הערך של התכונה יכול להשתנות במהלך הזמן.

אחת הנקודות החשובות שיש לבחון היא: האם התכונה **חייבת** להכיל ערך כלשהו (והמשמעות היא שחובה לספק את הנתון), או שהתכונה יכולה להיות ריקה ללא ערך (והמשמעות היא שהנתון הוא רשות ואין חובה לספק אותו). אם הנתון הוא רשות, הוא יכול להכיל **ערך רֵיק** (Null Value), אשר מציין שהנתון חסר או לא ידוע. נתון כזה יכול להיות במודל הנתונים, אולם בנקודת זמן מסוימת ייתכן שאינו מכיל ערך או שערכו בלתי ידוע. כפי שנראה בהמשך, מערכות RDBMS תומכות באופן ישיר בערכים חסרים.

מקובל להבחין בין תכונות שיש להן ערך אחד בלבד לבין תכונות מרובות ערכים.

תכונה עם ערך בודד או מרובת ערכים **Single/Multiple Value Attribute**	***תכונה עם ערך בודד* מכילה בכל נקודת זמן ערך אחד בלבד. *תכונה מרובת ערכים* יכולה לקבל בכל נקודת זמן ערך אחד או יותר.**

לדוגמה, התכונות מספר תעודת זהות, שם הסטודנט, מספר הקורס, מין הסטודנט יכולות להכיל בכל נקודות זמן ערך אחד בלבד. תכונות כגון ציוני בגרות, תארים קודמים יכולות לקבל מספר ערכים שונים בו-זמנית. לדוגמה, התכונה ציוני בגרות יכולה לקבל באותו זמן את כל הערכים האלה: 85, 90,67, 89, 95, 58. כפי שניתן לראות, במצב זה צריך להשתמש בתו כלשהו כדי להפריד בין הערכים השונים. כאשר עלינו לטפל במערכות ממוחשבות בתכונות מרובות ערכים, עלינו לדעת מראש את המספר המקסימלי של הערכים שיש לנהל. רוב המערכות המסחריות לניהול בסיסי נתונים, למעט Adabas, אינן תומכות באופן ישיר בתכונות מרובות ערכים. כפי שנראה בשלב מאוחר יותר, הדרך לנהל תכונות מרובות ערכים היא לפרק ולנהל אותן כאוסף של תכונות בעלות ערך בודד.

טיפוס נתונים (Data Type)

טיפוס נתונים **Data Type**	***טיפוס נתונים* מגדיר את סוג הערכים שתכונה יכולה לקבל.**

טיפוסי הנתונים הבסיסיים ביותר המקובלים:

- **מספר שלם** (Integer): תכונה היכולה לקבל ערך של מספר שלם, חיובי או שלילי.
- **מספר עשרוני** (Decimal): תכונה היכולה לקבל ערך של מספר עשרוני כלשהו, שבו מספר ספרות לפני הנקודה העשרונית ומספר ספרות אחריה.
- **מחרוזת תווים** (Character): תכונה היכולה לקבל כערך אוסף תווים הכולל ספרות, אותיות או סימנים מיוחדים.
- **בוליאני** (Boolean): תכונה היכולה לקבל את הערך "אמת" או "שקר".

לכל אחד מטיפוסי הנתונים האלה יש אוסף אופרטורים שמאפשרים לבצע פעולות על הערכים שהם מייצגים. לדוגמה, אם טיפוס הנתונים הוא מספר שלם או מספר עשרוני, ניתן להפעיל אופרטור אלגברי כגון חיבור, חיסור, כפל, שורש וכד'. אם טיפוס הנתונים הוא מסוג מחרוזת תווים ניתן לבצע שרשור (Concatenation) של שתי מחרוזות. בנוסף לטיפוסי הנתונים הבסיסיים, מקובל להשתמש במספר טיפוסי נתונים נוספים, כגון תאריכים, זמן, כסף ועוד.

מרחב ערכים (Attribute Domain)

מרחב הערכים **Attribute Domain**	***מרחב הערכים* מוגדר כאוסף כל הערכים החוקיים שתכונה יכולה לקבל.**

מרחב הערכים מגדיר את פוטנציאל הערכים שתכונה מסוימת יכולה לקבל. קיימות שיטות שונות להגדרתו:

- **הגדרת טווח ערכי:** הגדרת החסם התחתון והחסם העליון בטווח הערכים, שתכונה יכולה לקבל. לדוגמה, מספר נקודות זכות של קורס יכול להיות כל מספר שלם בין 1 ל-5. בצורת רישום פורמלית יותר, נוכל לכתוב:

$$X = \{X \mid 1 \le X \le 5 \text{ או } X = \text{שלם}\}$$

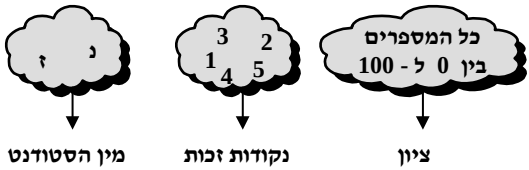
- **הגדרה מפורשת של הערכים החוקיים:** במקרים שבהם מספר הערכים שתכונה יכולה לקבל הוא קטן וסופי, נוכל לכתוב את אוסף כל הערכים החוקיים. לדוגמה, מין הסטודנט הוא זכר או נקבה, סוג הקורס הוא שיעור, סמינר או מעבדה.

$$X = \{X \mid X = \text{שיעור} \text{ או } X = \text{מעבדה} \text{ או } X = \text{סמינר}\}$$

- **הגדרת טיפוס הנתונים ואורכו:** יש מצבים שבהם מספר הערכים החוקי גדול מאוד, ולכן נוכל רק לקבוע מהו טיפוס הנתונים ומה אורכו. למשל, הערך תעודת זהות מיוצג על ידי מספר שלם בן תשע ספרות, שכר הלימוד הוא מספר עשרוני בעל חמש ספרות לפני הנקודה ושתי ספרות אחריה, שם הסטודנט הוא מחרוזת תווים בת 30 תווים.

$$X = \{X \mid X \text{ מחרוזת של 30 תווים}\}$$

להגדרת מרחב הערכים יש חשיבות רבה בבניית מודל הנתונים, כי הוא מגדיר אם התכונה מייצגת ערך נומרי (כגון ציון), או אלפאביתי (כגון שם סטודנט), ואת שיטת הייצוג של הערך הזה בבסיסי הנתונים. למשל, טיפוס הנתונים של העמודה מספר קורס יהיה מחרוזת באורך 5; טיפוס הנתונים של העמודה מין הסטודנט יהיה מחרוזת באורך תו אחד שיכול להכיל את האות ז או נ; טיפוס הנתונים של העמודה מספר נקודות הזכות הוא נומרי ויכיל את המספרים השלמים בין 1 ל-5.

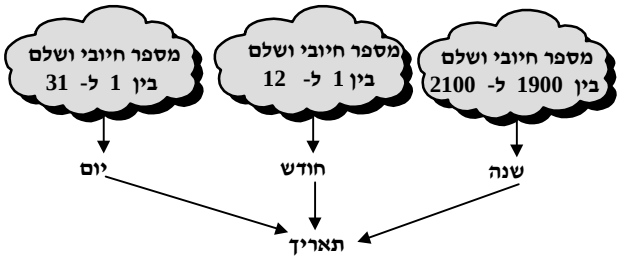


תרשים 3.6: תכונות ומרחבי הערכים שלהן.

תכונות שונות יכולות לקבל את הערכים שלהן מאותו מרחב ערכים. לדוגמה, התכונה מספר טלפון בבית והתכונה מספר טלפון בעבודה תקבלנה את ערכיהן ממרחב מספרי הטלפון. במקרה זה נאמר, שלאותו מרחב ערכים יש **תפקיד** (Role) שונה והוא בא לידי ביטוי בשם השונה שניתן לכל תכונה. מרחב הערכים של מספרי טלפון יוגדר כאוסף כל המספרים השלמים במבנה YY-XXXXXXX, כאשר YY הוא מספר המייצג את אזור החיוג ו- XXXXXXX הוא מספר הטלפון בתוך אזור החיוג.

תרשים 3.7: שני תפקידים שונים לאותו מרחב ערכים.

מרחב ערכים יכול להיות מבוסס בעצמו על מרחבי ערכים אחרים. למשל, מרחב הערכים של תאריך הוא תאריכים ומרחב זה בעצמו יכול להיות מורכב ממרחבי הערכים יום – מספר חיובי שלם בין 1 ל- 31, חודש – מספר חיובי שלם בין 1 ל- 12 ומרחב הערכים שנה.



תרשים 3.8: דוגמה של מרחב ערכים המורכב ממרחבי ערכים.

אחת הבעיות הקשות ביותר העומדות בפני מעצב מודל הנתונים היא קביעת התכונות של הישויות השונות. לדוגמה, בזמן הגדרת מודל הנתונים עבור מערכת ניהול משאבי אנוש של עובדי המפעל, החליט המעצב להשתמש בתכונות כגון שם עובד, תאריך תחילת עבודה, ניסיון מקצועי, כתובת ועוד. הוא לא בחר את התכונה סיווג בטחוני, מפני שבשלב זה היא אינה רלוונטית למודל. בעתיד, ייתכן מצב שבו המפעל יזכה במכרז בטחוני כלשהו והמידע אודות הסיווג הבטחוני של עובדיו יהיה חשוב. במקרה זה צריך יהיה להוסיף תכונה חדשה לישות עובדים.

בעיה אחרת היא, מתי לייצג אובייקט מסוים במציאות כישות, ומתי לייצג אותו כתכונה של ישות. אבחנה זו היא לעיתים סובייקטיבית ותלויה בנסיבות ובהקשר. לדוגמה, במערכת המידע של המכללה, הקורס הוא אחד מהאובייקטים העיקריים, ולכן הוא יהפוך לישות במודל הנתונים. לעומת זאת, במערכת מידע לניהול משאבי אנוש, ייתכן שכל מה שאנו מבקשים לדעת זה את שם הקורס האחרון בו העובד השתתף, ולכן במקרה זה האובייקט קורס יהפוך לתכונה של הישות עובד ולא יהיה ישות בפני עצמה. האבחנה בין ישות לבין תכונה יכולה להתבצע רק בהקשר לדרישות המידע.

לסיכום, החשיבות של מרחב הערכים נובעת מסיבות אלו:

- הוא מגדיר את האופרטורים החוקיים שניתן להפעיל. לדוגמה, על תכונה של טיפוס נתונים נומרי אפשר להפעיל אופרטור כפל, אבל לא ניתן להפעיל אופרטור זה על תכונה של טיפוס נתונים אלפאביתי.
- הוא מגדיר את אוסף הערכים החוקיים של כל תכונה.

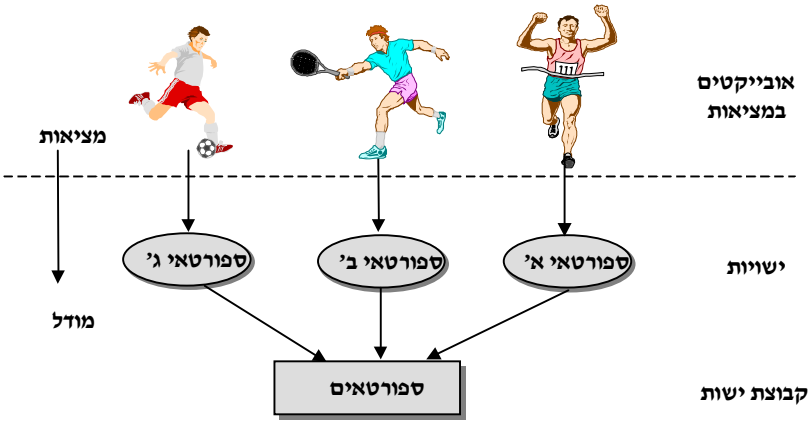
- הוא מגדיר בין איזה תכונות ניתן להשוות. לדוגמה, למרות שמספר סטודנט הוא ערך נומרי ומספר קורס גם הוא ערך נומרי, אלה שני מרחבי ערכים שונים ואין כל הגיון להשוות ביניהם.
- הוא משמש להגדרת טיפוס הנתונים והאורך הנדרש לייצוג התכונה במערכת מידע ממוחשבת.

קבוצת ישות (Entity Type)

סוג ישות **Entity Type**	***סוג ישות*** **מוגדר כאוסף של ישויות מאותו סוג.**

הארגון אינו עוסק בניהול נתונים על ישות אחת בודדת, אלא עוסק בקבוצת ישויות מאותו סוג. אוסף כל הישויות מאותו סוג ייקרא **קבוצת ישות**. מבחינת המודל אלו ישויות מאותו סוג ובעלות אותן תכונות, אבל הערכים של התכונות הללו יהיו שונים בישויות השונות, כלומר בין המופעים (Occurrences) השונים של הישות. מבחינת מודל הנתונים, ברור שנתמקד בקבוצה ולא בישות הבודדת, ננסה לבנות אשכולות של ישויות דומות מבחינת התכונות שלהן ונקבץ אותן יחד לקבוצת ישות אחת.

נציג מספר דוגמאות של אובייקטים במציאות ואת קבוצת הישויות שתוגדר במודל. המכללה שבדוגמה שלנו מעוניינת לנהל נתונים על כל הסטודנטים שלומדים בה, על כל המרצים שמלמדים, על כל הקורסים המוצעים לסטודנטים במועדים שונים, על כל ההישגים של הסטודנטים בכל קורס בו הם למדו ועוד. כל סטודנט הוא אובייקט ובמודל הוא יהפוך לישות. אוסף כל הסטודנטים באוניברסיטה יוגדר כקבוצת הישות סטודנטים, אשר מורכבת מכל הסטודנטים. הקורסים מבוא לכלכלה, מבוא למדעי המחשב, אלגברה לינארית הם אובייקטים במציאות וישויות המרכיבות את קבוצת הישות קורסים. כל הספורטאים השייכים למועדון ספורט מסוים הם אובייקטים במציאות והם גם מרכיבים את קבוצת הישות ספורטאים במודל.



תרשים 3.9: המעבר מאובייקטים במציאות לישויות ולקבוצת ישות במודל.

במציאות יש הבדל בין אובייקט אחד לאחר. כל אובייקט מיוצג על ידי ישות עם תכונות ולכן, מנקודת המבט של המודל ההבדלים בין הישויות יבוא לידי ביטוי בערכים השונים שהתכונות של הישויות השונות תקבלנה. למשל, לכל ישות סטודנט יש תכונה הנקראת מספר זהות, אבל הערך שתכונה זו תקבל יהיה שונה עבור כל סטודנט.

כפי שנראה בהמשך, ניתן לייצג כל קבוצת ישות כ**טבלה**, שכל תכונה בה מיוצגת על ידי עמודה וכל ישות מיוצגת על ידי שורה. על הרעיון הזה של **מיפוי קבוצת הישות לטבלה**, מבוסס **המודל הטבלאי**, אותו נציג בפרק הבא.

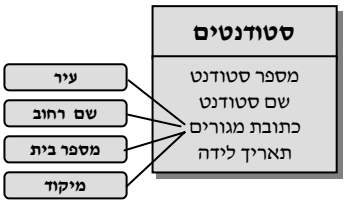
מבחינה גרפית, המודל משתמש במלבן כדי לייצג קבוצת ישות. בראש המלבן יופיע **שם הקבוצה** – תמיד בלשון רבים – כדי לבטא את העובדה שזו קבוצה ולא ישות בודדת. בתוך המלבן רשומות כל התכונות השייכות לאותה קבוצה, ואשר ינוהלו בנפרד עבור כל מופע של ישות.

תרשים 3.10: דוגמה של שלוש קבוצות ישות עם התכונות שלהן.

דרך מקוצרת לרישום הישויות תהיה על ידי כתיבת שם הקבוצה ומייד אחריה, בסוגריים, רשימת כל התכונות שלה. לדוגמה, את קבוצת הישות קורסים נוכל לרשום בצורה זו:

קורסים (קוד קורס, שם קורס, נקודות זכות, סוג קורס, קוד מחלקה)

אם התכונה היא תכונה מורכבת ומבקשים לתאר את מרכיביה בנפרד, ניתן לתאר את הקשר בין המרכיבים בצורה זו:



תרשים 3.11: צורת ייצוג של תכונה מורכבת.

אם התכונה היא מרובת ערכים, מקובל לרשום לידה את מספר המופעים המירבי, אם הוא ידוע, או n – כדי לייצג את העובדה שמספר המופעים המירבי אינו ידוע. לדוגמה, אם אותו קורס מוצע על ידי מחלקות שונות, התכונה מספר מחלקה הופכת לתכונה מרובת מופעים. מכיון שאין אנו יודעים את מספר הפעמים שקורס יכול להיות מוצע על ידי מחלקות שונות, כתבנו ליד התכונה חסם עליון n, המייצג מספר כלשהו.

תרשים 3.12: תכונה מרובת ערכים.

מפתחות ישות (Entity Keys)

כפי שראינו, קבוצת ישות אחת יכולה להכיל מופעים של מספר רב, ולמעשה מספר בלתי מוגבל, של ישויות שונות. נשאלת השאלה, כיצד ניתן להבחין בין ישויות שונות השייכות לאותה קבוצת ישות. התשובה היא: על ידי ה"מפתח".

מפתח אפשרי **Candidate Key**	***מפתח אפשרי* של קבוצת ישות מוגדר כאוסף תכונות K, המזהות באופן חד-ערכי מופע של ישות מסוימת בתוך קבוצת ישות E.**

כל מפתח חייב לקיים את שני הכללים האלה:

- **זיהוי חד-ערכי (Unique Identification):** הערך של המפתח מזהה באופן חד-ערכי את הישות בתוך קבוצת הישות.
- **מינימליות (Non Redundancy):** לא ניתן לסלק מתוך K (המכיל תכונה אחת או יותר) אף תכונה מבלי להרוס את התנאי הראשון של הזיהוי החד-ערכי. המינימליות באה להבטיח שהאוסף K יכיל רק את התכונות הדרושות לזיהוי חד-ערכי של המפתח.

לכל קבוצת ישות ניתן להגדיר אוסף כלשהו של תכונות המזהות באופן חד-ערכי את הישות. משמעות הזיהוי החד-ערכי היא שלא תיתכנה שתי ישויות שונות המכילות את אותו הערך בתכונות המוגדרות כמפתח עיקרי. לדוגמה, בקבוצת הישות סטודנט, התכונה מספר סטודנט מזהה באופן חד-ערכי סטודנט כלשהו, והדבר מבטיח לנו שאין שני סטודנטים שונים בעלי אותו מספר סטודנט. דוגמה אחרת יכולה להיות התכונה מספר קורס המזהה באופן חד-ערכי את הישות קורס בתוך קבוצת הישות קורסים.

לקבוצת ישות אחת יתכנו מספר מפתחות. כך, לקבוצת הישות סטודנט יש שני מפתחות שונים: מספר סטודנט ומספר זהות. כל תכונה כזו מהווה מפתח בפני עצמה. לקבוצת הישות חייל יש שני מפתחות: מספר זהות ומספר אישי. כל אחת מתכונות אלו היא **מפתח אפשרי**. לדוגמה:

סטודנטים (מספר סטודנט, מספר תעודת זהות, שם סטודנט, עיר מגורים)

כל אחת מהעמודות מספר סטודנט ומספר תעודת זהות יכולות לשמש בנפרד כמפתח. במקרה כזה נאמר שלקבוצת הישות יש שני מפתחות אפשריים.

מבין כל המפתחות האפשריים של טבלה עלינו לבחור מפתח אחד, אשר מבחינת המודל ישמש כ**מפתח עיקרי**.

מפתח עיקרי **Primary Key**	***מפתח עיקרי* מוגדר כאחד מבין המפתחות האפשריים אשר נבחר לשמש כמפתח של קבוצת הישות וישמש לזיהוי חד-ערכי של הישויות.**

מפתח עיקרי חייב להכיל ערך כלשהו מתוך מרחב הערכים, ואסור לו להיות **ריק** (Null Value). שאר התכונות של קבוצת הישות יכולות לקבל ערך ריק.

לכל קבוצת ישות יש מפתח עיקרי אחד, ומפתח אפשרי אחד או יותר. באותם מקרים בהם יש רק מפתח אפשרי אחד, קיימת זהות בין המפתח האפשרי לבין המפתח העיקרי. **השיקולים** בבחירת המפתח העיקרי קשורים באופן הדוק למקרה שבו דנים. הם יכולים להיות – יציבות לאורך זמן (כלומר שהתכונה לא תפסיק לזהות את הישויות באופן חד-ערכי), שהמפתח יהיה קצר וקומפקטי וכד'.

מפתח יכול להיות פשוט (Simple), או מורכב (Composite).

מפתח פשוט/מורכב **Simple/Compound Key**	**כאשר המפתח של קבוצת הישות מורכב מתכונה אחת בלבד, מקובל לומר שהוא *מפתח פשוט*, וכאשר הוא מורכב ממספר תכונות מקובל לומר שהוא *מפתח מורכב*.**

בחלק מקבוצות הישות מספיקה תכונה אחת לזהות את הישות. הבה נתבונן בקבוצת ישות מורכבת יותר, כמו למשל ציון בקורס. קבוצה זו מייצגת את העובדה שסטודנט מסוים נרשם לקורס כלשהו בסמסטר כלשהו ומכילה את הציון בבחינה שהסטודנט קיבל. במקרה זה, המפתח של קבוצת הישות הוא מפתח מורכב – שילוב של מספר קורס, מספר סטודנט וסמסטר. רק השילוב הזה יזהה באופן חד-ערכי כל מופע. באותו קורס יכולים ללמוד מספר סטודנטים, אותו סטודנט יכול ללמוד במספר קורסים ובאותו סמסטר יתכנו מספר סטודנטים שנרשמו לאותו הקורס. על כן, רק השילוב יוכל לזהות באופן חד-ערכי את הישות.

מבחינה גרפית, המפתח העיקרי יסומן במודל על ידי קו תחתי מתחת לשמות התכונה או התכונות המרכיבות את המפתח.

תרשים 3.13: צורת סימון המפתח העיקרי של קבוצת ישות.

נשתמש בצורת הרישום המקוצרת של קבוצת הישות, ונסמן את המפתח באותה דרך.

קורסים (קוד קורס, שם קורס, נקודות זכות, סוג קורס, קוד מחלקה)

בהמשך נגדיר גם קשרים בין ישויות וקבוצות ישות. בשלב זה נאמר רק שקבוצת הישות יכולה להכיל גם מפתח זר.

מפתח זר **Foreign Key**	***מפתח זר*** **מוגדר כאוסף תכונות FK (תכונה אחת או יותר) המופיע בקבוצת ישות E_1, המשמש גם כמפתח עיקרי בקבוצת ישות E_2, כלומר $FK_{E1} = PK_{E2}$. ערכי המפתח הזר בקבוצה E_1 הם תת-קבוצה של ערכי המפתח העיקרי בקבוצה E_2, או שהם מקבלים את הערך Null.**

לדוגמה, בקבוצת הישות קורסים, התכונה קוד מחלקה היא מפתח זר, מכיון שהיא משמשת כמפתח עיקרי בקבוצת הישות מחלקות. המפתח הזר הינו בעל חשיבות ביצירת קשרים בין קבוצות ישות שונות במודל הטבלאי.

תרשים 3.14: מפתחות עיקריים וזרים.

את שתי קבוצות הישות נוכל לייצג באמצעות שתי הטבלאות האלו:

קורסים (קוד קורס, שם קורס, נקודות זכות, סוג קורס, קוד מחלקה)

מחלקות (קוד מחלקה, שם מחלקה, שם ראש המחלקה)

היררכיות סמנטיות (Semantic Hierarchies)

ננסה להבין מהם התהליכים המנחים את מעצב מודל הנתונים להבנת המציאות, לזיהוי האובייקטים, למיפוי האובייקטים כישויות, למיפוי אובייקטים לקבוצות ישות שונות ולקביעת תכונותיהן. Smith&Smith [SS80] הציעו תשובה אפשרית לתהליך זה. בבסיס התהליך נמצא רעיון ה**הפשטה** (Abstraction).

מקור הרעיון או תפישת ההפשטה הוא במחקרים בתחום האינטליגנציה המלאכותית ובשיטות לייצוג ידע (Knowledge Presentation). במשך השנים אומצה תפישה זו על ידי מעצבי המודל ישויות-קשרים. הם קבעו את שם המודל: **מודל ישויות-קשרים משופר** – EER (Enhanced Entity Relationship Model). Elmasri ו- Navathe [EN94] תרמו רבות לפיתוח המודל המשופר והוא מוצג בהרחבה בספרם.

הפשטה (Abstraction)

הפשטה **Abstraction**	***הפשטה*** **מוגדרת כתהליך לוגי המתרחש במוחו של האדם כאשר הוא בוחר להתמקד במספר מאפיינים של אוסף אובייקטים ולהתעלם ממאפיינים אחרים.**

בני אדם מפעילים את תהליכי ההפשטה בכל פעם שהם מתמקדים במספר מסוים של מאפיינים שהם מייחסים להם חשיבות בהקשר מסוים, ובוחרים להתעלם מאוסף אחר של מאפיינים של אותם אובייקטים שאינם רלוונטיים לדעתם. זהו תהליך תפישתי (קונספטואלי) שבו מסתירים פרטים מסוימים של אובייקט בודד ומתמקדים בכלל, כלומר במאפיינים המשותפים לאוסף האובייקטים.

האובייקט ספר מתייחס לאוסף של דפים מודפסים הכרוכים יחדיו, האובייקט חדר מתייחס לאוסף קירות, רצפה, תקרה ואולי גם ריהוט מסוים וכן הלאה. בשתי דוגמאות אלו תהליך ההפשטה מתייחס לפרטים מסוימים ומתעלם מפרטים אחרים. ישנם ספרים שונים מבחינת מספר העמודים, גודל הספר, סוג הכריכה וכד׳, אבל למרות זאת אנו מתעלמים משוני זה ביניהם ומתמקדים בעובדה הפשוטה שכולם ספרים ומשמשים לקריאה.

בהמשך נציג שלושה תהליכי הפשטה: סיווג, הכללה וצירוף. שלושת תהליכי הפשטה אלה נותנים הסבר אפשרי כיצד נבנות קבוצות ישות. כפי שנראה, נוצרות היררכיות מורכבות מאוד, שנכנה אותן בשם **היררכיות סמנטיות** (Semantic Hierarchy). היררכיות סמנטיות הן דרך נוחה מאוד לבניית קבוצות ישות, פירוקן לתת-קבוצות והגדרת התכונות של כל אחת מהן. רעיונות אלה מהווים בסיס להבנת **מודלים מוכווני-אובייקטים** (Object Oriented Models).

סיווג (Classification)

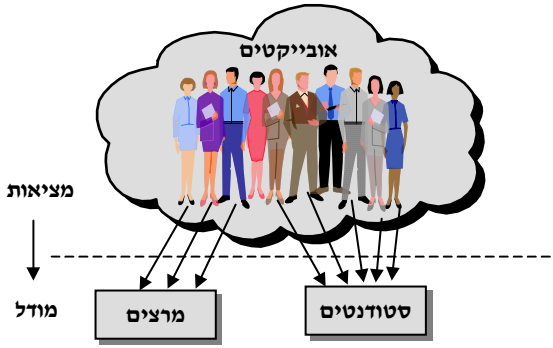
סיווג **Classification**	***סיווג*** **מוגדר כתהליך הפשטה המאפשר להגדיר מושג כלשהו כקבוצה. תהליך זה מאפשר לבנות קבוצה מתוך הפרטים ולכן מקובל לכנות אותו** ***תהליך המרה מן הפרט אל הכלל*** **(Token to Type).**

תהליך הסיווג הוא תהליך ההפשטה הבסיסי ביותר והוא מאפשר לבנות קבוצה מתוך פרטים מסוימים. הקבוצה יכולה להיות קבוצת ישות שנבנית מתוך האובייקטים במציאות, או קבוצה מופשטת כלשהי שנבנית מתוך פרטים כלשהם.

אם נתבונן למשל באובייקטים הרלוונטיים למכללה שבדוגמה שלנו, נמצא שם אנשים, בניינים, כיתות לימוד, ציוד מעבדה ועוד. מבין כלל האובייקטים במציאות נסווג אובייקטים מסוימים כסטודנטים, ואובייקטים אחרים כמרצים, למרות ששני האובייקטים האלה הם בני אדם. ההחלטה לסווג בני אדם מסוימים לאוסף אחד ואת האחרים לאוסף אחר, הינה תהליך הפשטה. בדוגמה שלנו, תהליך ההפשטה נבנה מתוך התמקדות ב**תפקידי** האובייקטים – יש כאלה הלומדים במכללה ויש כאלה המלמדים

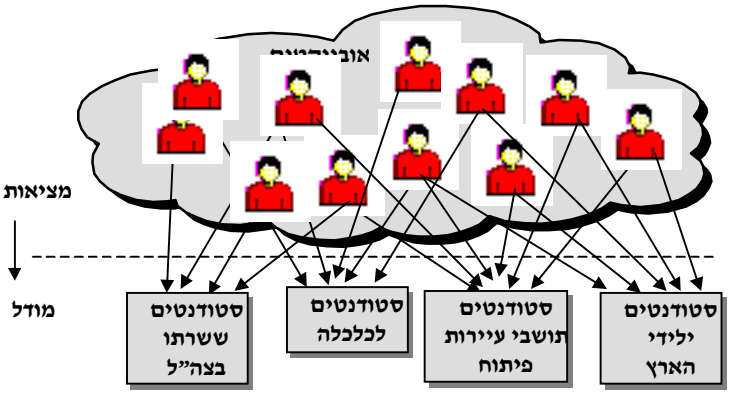
במכללה. ברור לנו שאין שני סטודנטים זהים זה לזה, אולם אנו מתמקדים במאפיין המשותף לכולם – הם לומדים במכללה. את המרצים אנו מסווגים לקבוצה אחרת, מרצים, בהתבסס על המאפיין המשותף לכולם – הם מלמדים במכללה (ראה תרשים 3.15).

באופן דומה, קבוצת הישות מכוניות היא קבוצה שחבריה הם כל המכוניות. לצורך סיווג כמכונית אנו מתעלמים ממאפיינים כגון צבע המכונית, נפח מנוע, סוג הילוכים, סוג המכונית – מכונית ספורט, מכונית נוסעים, משאית – ומתמקדים רק ביכולת הבסיסית של המכונית להסיע אנשים או מטענים ממקום למקום.



תרשים 3.15: סיווג אנשים כסטודנטים וכמרצים.

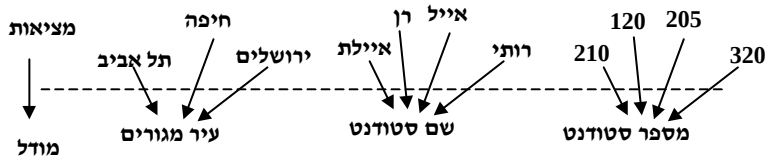
ניתן לסווג את אותם האובייקטים בדרכים שונות, ובכל פעם נתמקד במאפיין אחר שלהם.



תרשים 3.16: סיווגים שונים של אותה קבוצת אובייקטים.

לתהליך שבו אנו עוברים מן האובייקט הבודד (הסטודנט, המרצה) אל קבוצת הישות (מרצים, סטודנטים), מקובל גם לקרוא **המרה מהפרט אל הכלל**.

תהליך הסיווג יכול להסביר לא רק את הדרך בה נוצרות קבוצות ישות, אלא גם את הדרך שבה נוצרות קבוצות מופשטות נוספות ההופכות בסופו של דבר לתכונות של קבוצות ישות במודל הנתונים. לדוגמה, ניתן להגדיר קבוצות כגון עיר מגורים, מספר סטודנט, שם סטודנט שהן מושגים מופשטים הנוצרים מתוך פרטים כלשהם. המושג עיר מגורים, למשל, הוא מושג מופשט שנוצר בתהליך סיווג של ערים שונות: חיפה, תל אביב, ירושלים ואחרות. תהליך זה מתמקד בערים השונות בהקשר של איזו עיר משמשת כעיר מגורים של סטודנט ומתעלמת מכל שאר המאפיינים האחרים של ערים, כמו מספר התושבים, שטח בקמ"ר ועוד.

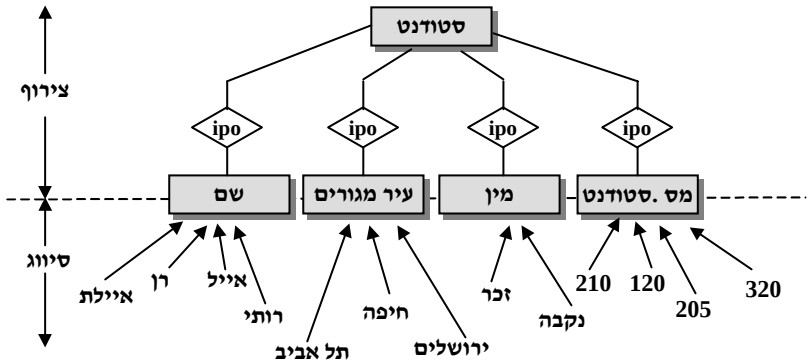


תרשים 3.17: תהליך הסיווג כתהליך לבניית תכונות.

צירוף (Aggregation)

צירוף Aggregation	**צי*רוף* מוגדר כתהליך הפשטה, המאפשר להגדיר קבוצה חדשה מתוך קבוצות קיימות.**

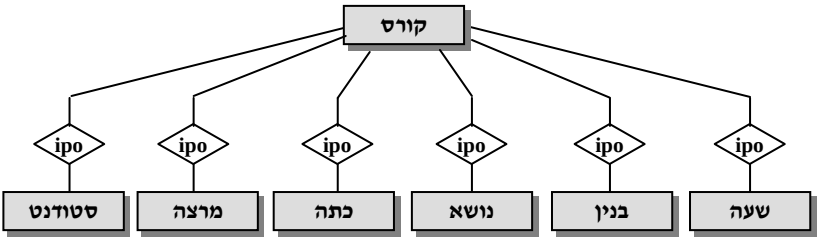
בסעיף קודם ראינו שניתן להסביר תהליך בנייה של מושגים מופשטים, כגון עיר מגורים או שם סטודנט על ידי סיווג. כך מקבלים קבוצות שונות ונפרדות זו מזו. תהליך הצירוף מאפשר להדביק קבוצות שונות אל קבוצה אחרת. כך, למשל, נוכל להדביק לישות סטודנט את התכונות שלה. מקובל לקרוא באנגלית לקשר הצירוף גם בשם **is part of**, ובקיצור – **ipo**. לדוגמה, ניתן לומר ש- "City **is part of** a student". את תהליך הצירוף מסמנים באופן גרפי על ידי מעוין שבתוכו רשום ipo .



תרשים 3.18: תהליך של סיווג וצירוף.

התרשים מראה כיצד הישות סטודנט נוצרת מתוך שני תהליכי הפשטה של הסיווג ושל הצירוף.

כדוגמה נוספת נתייחס לקבוצת הישות קורס, המבטאת קשר בין הקבוצות מרצה, סטודנט, נושא נלמד, כתה, בניין ושעה. קבוצת הישות קורס מבטאת את **הקשר** בין כל הקבוצות המפורטות. נשים לב שחלק מהקבוצות הן ישויות (סטודנט, מרצה) וחלקן – תכונות (כתה, נושא נלמד, בניין ושעה). כפי שנאמר, ההחלטה אם קבוצה תהפוך לישות או לתכונה היא בידי המעצב ונעשית בהקשר לדרישות המידע.



תרשים 3.19: היררכיה של צירוף.

הכללה (Generalization)

הכללה **Generalization**	***ההכללה*** **מוגדרת כתהליך הפשטה המאפשר לבנות קבוצה המהווה קבוצת-על (Super Set) של קבוצות אחרות. לפנינו** ***תהליך המרה מהכלל אל הכלל*** **(Type to Type).**

תהליך ההכללה מאפשר לבנות קבוצות ישות המהוות **קבוצת-על**, או **קבוצת ישות-על**. הן קרויות כך מכיון שהן ברמת הפשטה גבוהה יותר מזו של תת-הקבוצות של הישויות הבונות אותן ונמצאות ברמת הפשטה נמוכה יותר. תהליך זה מנסה לזהות את הדומה בין הישויות השונות ולהתעלם ככל האפשר מהשונה ביניהן.

תהליך הסיווג של כל עובדי המכללה בדוגמה שלנו הוא אשר בנה את קבוצות הישות האלו: טכנאים, מרצים זוטרים, מרצים בכירים, עובדי ספריה, עובדי מנהלה, עובדי מעבדה. תהליך הפשטה נוסף מזהה שקיים "דמיון" בין המרצים הזוטרים, המרצים הבכירים ועובדי הספריה. דמיון זה מתבטא בעובדה שהם שייכים לסגל האקדמי של האוניברסיטה, המקנה להם זכויות מסוימות, כגון שנת שבתון. לעומת קבוצות הישות הללו, שאר קבוצות הישות כמו טכנאים, עובדי מעבדה, עובדי מנהלה משתייכים לסגל המנהלי. הבחנה זו מאפשרת לבנות שתי קבוצות ישות ברמה גבוהה יותר: סגל מנהלי וסגל אקדמי. נשים לב שבתהליך זה התעלמנו מהשוני בין סוגי האוכלוסיה השונים במכללה והתמקדנו במושג נרחב יותר: הזכויות של קבוצות אלו הנובעות מהסכם העבודה שלהן. ניתן להתקדם צעד נוסף, להתעלם מהסכמי העבודה השונים ולשייך את כולם לקבוצת-על אחת בשם עובדי המכללה.

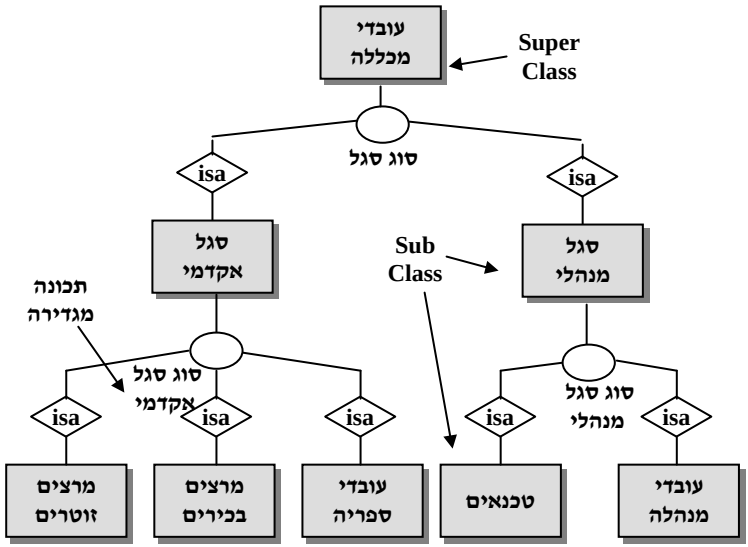
תהליך ההכללה מאפשר לבנות היררכיות של קבוצות ישות, שמקובל לכנות אותן בשם **היררכיות הכללה** (Generalization Hierarchies), או כפי שהן ידועות גם בשמן הכוללני, **היררכיות סמנטיות** (Semantic Hierarchies).

באותה מידה שבה בונים את ההיררכיה מלמטה כלפי מעלה, ניתן לטעון גם שתהליך ההפשטה הוא הפוך, שבו מתחילים **מקבוצת ישות-על** (Super Class) ובאופן איטרטיבי מפצלים אותה ל**תת-קבוצות** (Sub Classes). על כן מקובל לקרוא לתהליך **התמחות** (Specialization).

תהליכי ההכללה וההתמחות הם תהליכי הפשטה זהים, שההבדל ביניהם הוא רק בנקודת ההתחלה: האם מתחילים מקבוצות ברמות נמוכות ובונים מהן קבוצות ברמות גבוהות יותר, או שמתחילים מקבוצות ברמות גבוהות ומחלקים אותן לקבוצות ברמות נמוכות יותר.

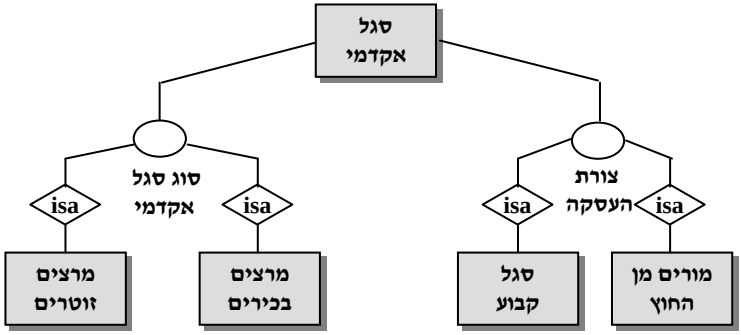
מכיון שאנו עוסקים בישויות מאותו סוג, עובדי מכללה, ורק מחלקים אותן בחלוקות משנה, מקובל לקרוא לקשר הזה גם בשם **isa**, המבטא את העובדה שכל חבר בקבוצת-על הוא גם חבר בתת-קבוצה. כלומר, טכנאי הוא חבר סגל מנהלי, חבר סגל מנהלי הוא עובד מכללה וכך הלאה. באנגלית נוכל לומר: "a faculty member **is an** employee" או: "a technician **is a** member of the administrative team".

לכל היררכיה יש להגדיר את התכונה המגדירה אותה (Subtype Discriminator). זוהי תכונה שהערך שלה קובע לאיזה ענף של ההיררכיה יש לשייך את הישות. התכונה המגדירה של סגל אקדמי או סגל מנהלי היא סוג סגל, שהיא תכונה של הקבוצה עובדי מכללה, ועל פיה יבוצע הסיווג.



תרשים 3.20: היררכית הכללה.

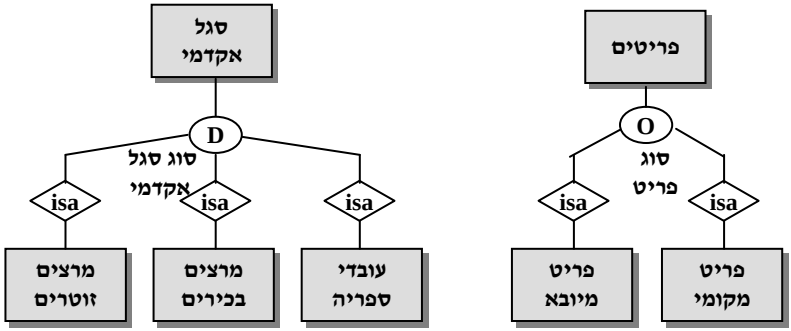
על פי תכונה מגדירה שונה ניתן לחלק קבוצה אחת לתת-קבוצות שונות, שהן למעשה היררכיות שונות.



תרשים 3.21: חלוקת קבוצה אחת למספר היררכיות שונות.

בתרשים 3.21 מוצגת דוגמה של חלוקת הקבוצה סגל אקדמי לשתי היררכיות שונות; האחת – על פי התכונה המגדירה סוג סגל אקדמי, והשנייה – על פי התכונה המגדירה צורת העסקה.

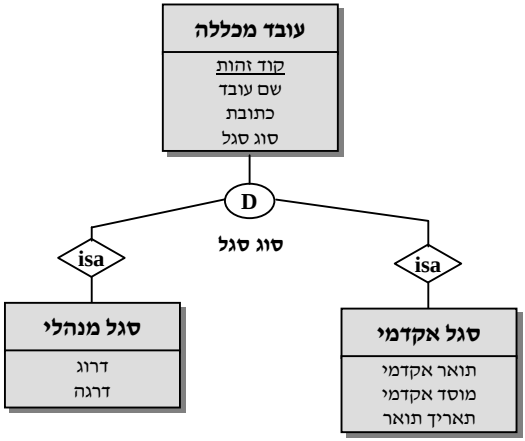
לפעמים קיימים אילוצים הקובעים אם מותר לחבר בתת-קבוצה אחת להיות חבר גם בתת-קבוצה אחרת (כמו למשל, האם בעל דרכון של מדינה אחת יכול להיות גם בעל דרכון של מדינה אחרת כלשהי). אם קיים אילוץ המונע חברות מסוג זה, נאמר שמתקיים **אילוץ של קבוצות זרות** (**D**isjoint Constraint). אם מותרת חפיפה, שבה חבר בתת-קבוצה אחת יכול להיות חבר גם בתת-קבוצה אחרת, נאמר שמתקיים **אילוץ חפיפה** (**O**verlap Constraint).



תרשים 3.22: ייצוג אילוצי חברות בתרשים ישויות-קשרים משופר.

בתרשים 3.22 מוצגת צורת רישום האילוצים. בהנחה שכל עובד יכול להשתייך בדיוק לתת-קבוצה אחת, רשמנו אילוץ של קבוצות זרות בהיררכיה הסמנטית של הסגל האקדמי. דוגמה נוספת היא שפריטים הנקנים על ידי מפעל יכולים להירכש בחו"ל או בארץ, ואולי אפילו משני מקורות בו-זמנית, ולכן רשמנו את אילוץ החפיפה בהיררכיה הסמנטית של הפריטים.

אחת התכונות המעניינות של היררכיות סמנטיות נובע מעיקרון **ההורשה** (Inheritance). כל קבוצה ברמה נמוכה יותר יורשת את כל תכונותיה מכל הקבוצות שנמצאות מעליה בהיררכיה.

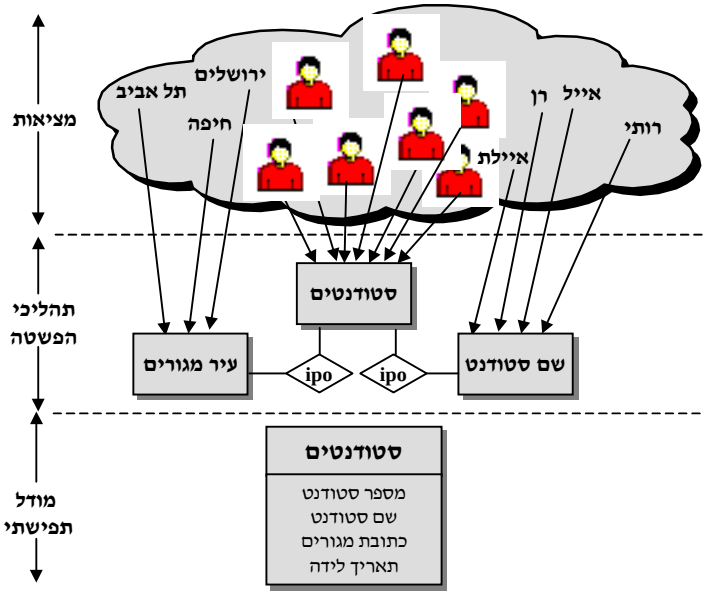


תרשים 3.23: הורשת תכונות בהיררכית הכללה.

תרשים 3.23 מציג דוגמה להיררכית **הכללה**, אשר בה מופיעה קבוצת ישות אחת עובד מכללה, ולה שתי תת-קבוצות: סגל אקדמי וסגל מנהלי. התכונות מספר זהות, שם, כתובת, סוג סגל עוברות בירושה מהקבוצה עובד אוניברסיטה אל כל תת-הקבוצות שלה. מכאן ברור, שגם לעובד אקדמי יש תכונה כגון שם עובד, למרות שהיא אינה מוצגת במפורש. זוהי צורת רישום קומפקטית מאוד.

לסיכום הדיון בתהליכי ההפשטה, הראינו שלושה תהליכי הפשטה שונים המשמשים לבניית המושגים והרעיונות השונים בהבנת המציאות ובבניית מודלים המתארים את המציאות. כלומר, אלה הם כלים בבניית מודל נתונים תפישתי.

תרשים 3.24 קושר בין כל הרעיונות שהוצגו עד כה. מעצב מודל הנתונים מתבונן על המציאות ומשתמש בתהליכי הפשטה שונים כדי לבנות את "ההבנה שלו של המציאות". אחר כך הוא בונה את המודל התפישתי ומשתמש לצורך זה בסמלים גרפיים שונים.



תרשים 3.24: המעבר מהמציאות למודל של המציאות המתבצע על ידי תהליכי הפשטה.

טבלת תיאור של קבוצת ישות

לא כל הידע אודות העולם הממשי אשר נצבר במהלך תהליך עיצוב המודל התפישתי, ניתן לתיאור על ידי מודל ישויות-קשרים כפי שהצגנו עד כה. הבעיה בולטת במיוחד כשצריך להגדיר אילוצים שונים. אין, למשל, כל דרך פורמלית לבטא במודל ישויות-קשרים את העובדה שקורס יכול להקנות בין 1 ל-4 נקודות זכות, או שהערך הכולל של ההזמנות שפרויקט מוציא חייב להיות קטן או שווה לתקציב הרכש שלו. מודלים כגון SDM (Semantic Data Model) הם בעלי כושר ביטוי טוב יותר בנושאים אלה, ולכן הקורא המעוניין מופנה אליהם.

השלמת כושר הביטוי הסמנטי של מודל ישויות-קשרים יכול להיעשות על ידי שימוש בטבלה מיוחדת שנבנית לכל קבוצת ישות. טבלה זו תכיל את הידע הנוסף על הקבוצה שלא בא לידי ביטוי במודל הרגיל. תהיה בה כניסה אחת עבור כל תכונה של הקבוצה שתפרט את תיאור התכונה, האילוצים שלה וכל מידע נוסף שנצבר עליה.

קשרים (Relationships)

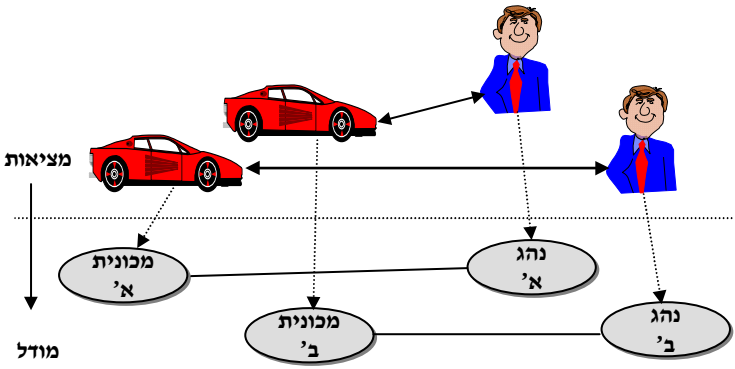
עד כאן התמקדנו בישויות, בקבוצת ישויות ובתכונות של ישויות. כפי שנראה, זו רק מחצית המלאכה בבניית מודל נתונים. המחצית השנייה עוסקת בהגדרת הקשרים בין הישויות השונות.

התבוננות בקבוצות הישות השונות מראה שמתקיימים קשרים ביניהן. זיהוי ואבחון נכון של הקשרים בין הקבוצות חשוב לא פחות מזיהוי הקבוצות עצמן. הטיפול המלא בקשרים אלה הוא המבדיל בין בסיס נתונים לבין סתם אוסף של קבצים.

קשר **Relationship**	***קשר* מוגדר כיחס בעל משמעות בין ישויות שונות.**

הרי מספר דוגמאות:

קשר לומד קיים בין ישויות בקבוצת סטודנטים לבין ישויות בקבוצת קורסים. הישות מכונית ששייכת לקבוצת הישות מכוניות, קשורה לישות נהג בקבוצת הישות נהגים. משמעות אפשרית אחת לקשר זה יכולה להיות בעלות, כלומר מכונית מסוימת היא בבעלות נהג מסוים, או נהג מסוים הוא הבעלים של מכונית מסוימת. יכולה להיות לקשר זה גם משמעות אחרת, כמו איזה נהג גרם תאונת דרכים עם איזו מכונית.



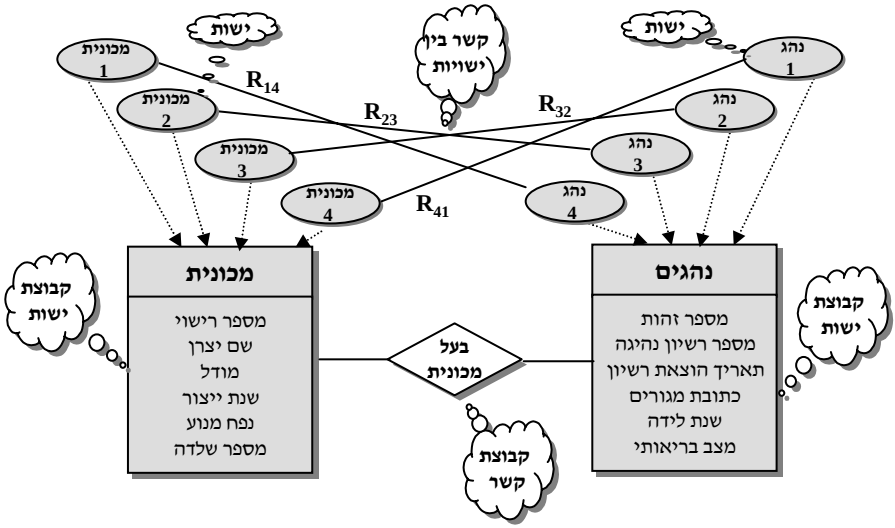
תרשים 3.25: קשר בין אובייקטים במציאות לבין ישויות במודל.

ראינו שיש הבדל בין ישות לקבוצת ישות. כך יש גם הבדל בין קשר לבין קבוצת קשר.

קבוצת קשר **Relationship Type**	***קבוצת קשר* מוגדרת כאוסף של כל הקשרים בעלי אותה משמעות בין הישויות השונות.**

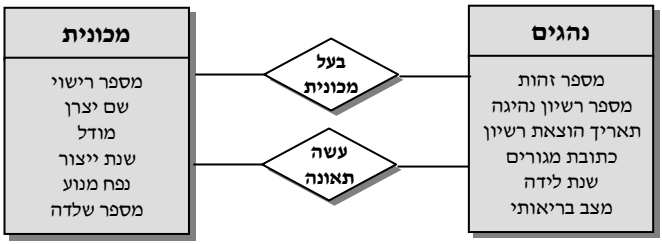
במודל ישויות-קשרים נסמן קשר על ידי מעוין הנושא את **שם הקשר**, שהוא המשמעות הסמנטית של הקשר. לכל קשר יש שם, מכיון שזוהי הדרך להבין מה משמעותו.

בתרשים 3.26 אנו רואים שבין הישויות השונות מתקיימים קשרים. לדוגמה, הקשר R_{23} מייצג את הקשר בין מכונית מספר 2 לבין נהג מספר 3. בעת בניית מודל ישויות-קשרים, כל הישויות הבודדות הופכות לקבוצות ישות שונות והקשרים בין הישויות הופכים לקבוצת קשר.



תרשים 3.26: ההבחנה בין קשר לבין קבוצת קשר.

בין שתי קבוצות ישות שונות יתכנו מספר קשרים שונים, ולא קשר אחד בלבד. כך, בין הישות נהג והישות מכונית יכול להתקיים קשר אחד במשמעות של בעלות, וקשר אחר – במשמעות תאונה, כלומר שהנהג גרם תאונת דרכים עם המכונית.



תרשים 3.27: קשרים שונים בין שתי קבוצות ישות.

דוגמה אחרת יכולה להיות קשר בין קבוצת הישות עובדים לבין קבוצת הישות פרויקטים. קשר אחד יכול להיות בעל משמעות משתתף המציין איזה עובד משתתף באיזה פרויקט, וקשר נוסף יכול להיות בעל משמעות מנהל שמציין איזה עובד מנהל איזה פרויקט.

בין קבוצות שונות ייתכנו מספר רב של קשרים. מעצב בסיס הנתונים צריך לבחור מתוכם רק את הקשרים הרלוונטיים שמעניינים את הארגון.

דרגת הקשר (Relationship Degree)

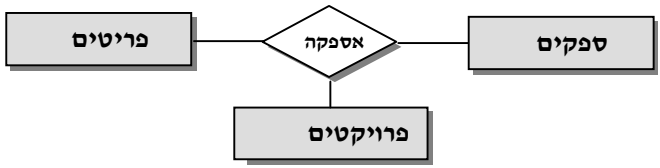
דרגת הקשר **Relationship Degree**	***דרגת הקשר* מוגדרת כמספר קבוצות הישות המשתתפות בקשר.**

להלן מספר דוגמאות לקשרים בדרגות שונות:

- **קשר בדרגה 2 (קשר בינארי)**. בקשר זה משתתפות שתי קבוצות ישות שונות. לדוגמה, הקשר לומד בין הקבוצות סטודנטים וקורסים.

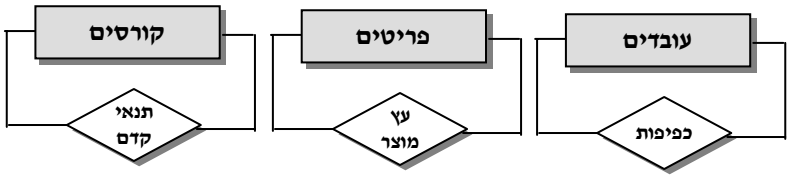
תרשים 3.28: דוגמה לקשר בדרגה 2, בינארי.

- **קשר בדרגה 3 (קשר טרנארי)**. בקשר זה משתתפות שלוש קבוצות ישות שונות. לדוגמה, הקשר אספקה מגדיר מהו הפריט שמסופק על ידי ספק כלשהו לפרויקט.



תרשים 3.29: דוגמה לקשר בדרגה 3, טרנארי.

- **קשר בדרגה 1 (קשר רפלקסיבי)**. קשר זה מתקיים בין ישויות שונות באותה קבוצת ישות. לדוגמה, רשימת הקורסים המהווים תנאי קדם לקורס כלשהו, או רשימת הפריטים המרכיבים מוצר מסוים (נקרא גם עץ מוצר), או רשימת עובדים הכפופים לעובד מסוים.



תרשים 3.30: דוגמאות לקשרים רפלקסיביים.

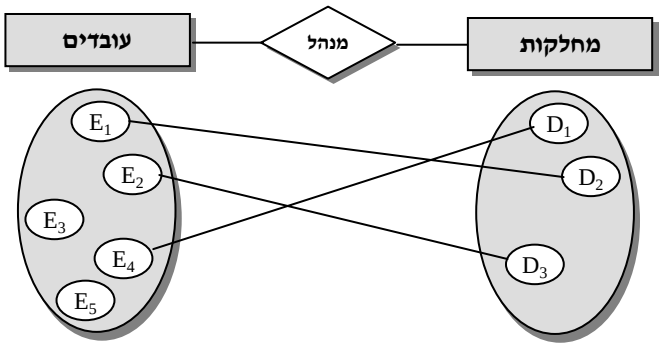
בנוסף לסוגי קשרים אלה ייתכנו קשרים גבוהים יותר מדרגה 3, אם כי מצב זה נדיר.

פונקציונליות הקשר (Relationship Functionality)

פונקציונליות הקשר **Relationship Functionality**	***פונקציונליות הקשר* מוגדרת כסוג המיפוי הקיים בין הקבוצות המשתתפות בקשר. הפונקציונליות יכולה להיות מסוג 1:1, 1:N או N:M.**

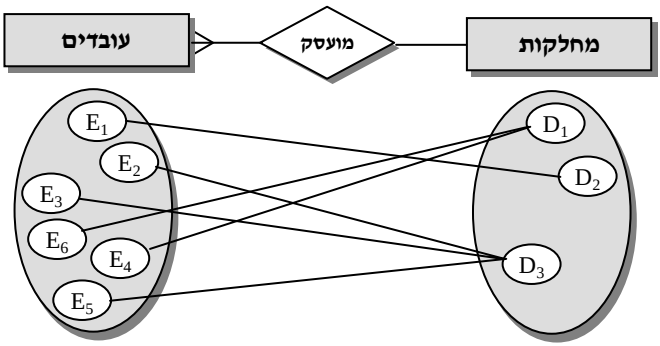
הפונקציונליות של הקשר מגדירה אילוץ על מספר הישויות שיכולות להשתתף בכל צד של הקשר. כפי שנאמר בהגדרה, אנו מבחינים בשלושה סוגי פונקציונליות של קשרים.

- **קשר חד-חד-ערכי (1:1).** בקשר זה ניתן לבצע התאמה בין ישות בקבוצת ישות אחת לבין ישות בקבוצת ישות אחרת; כלומר, ניתן להגדיר פונקציה משני צידי הקשר. נאמר שהקשר מנהל בין הקבוצות עובדים ומחלקות, הוא מסוג 1:1 אם כל עובד יכול לנהל מחלקה אחת לכל היותר, וכל מחלקה יכולה להיות מנוהלת על ידי עובד אחד בלבד. מקובל לסמן קשר מסוג זה על ידי קו רגיל משני צידי הקשר.



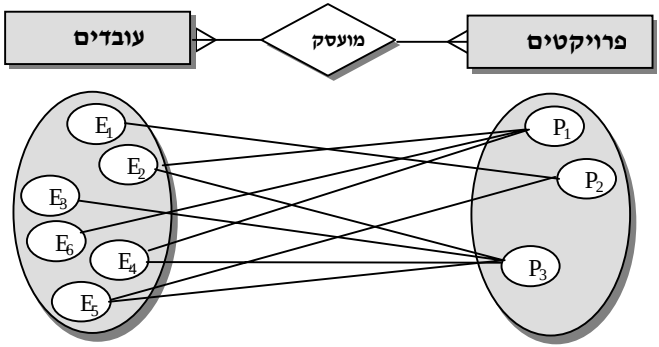
תרשים 3.31: דוגמה לקשר חד-חד-ערכי בין שתי קבוצות.

- **קשר חד-רב-ערכי (1:N).** בקשר זה ניתן לבצע התאמה בין ישות בקבוצת ישות אחת לבין ישות בקבוצת ישות אחרת, אולם לא להיפך. כלומר, ניתן להגדיר פונקציה רק בכיוון אחד של הקשר. נאמר שהקשר מועסק בין קבוצות הישות עובדים לבין מחלקות הוא מסוג 1:N. הסיבה לכך היא שכל עובד מועסק על ידי מחלקה אחת בלבד, אולם מחלקה אחת יכולה להעסיק מספר בלתי מוגבל של עובדים. מקובל לסמן קשר מסוג זה על ידי משולש בצד של הקשר הרב-ערכי.



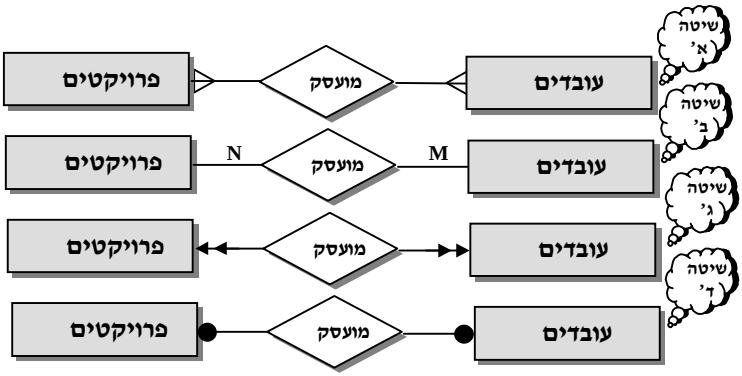
תרשים 3.32: קשר חד-רב-ערכי בין שתי קבוצות.

- **קשר רב-רב-ערכי (N:M).** בקשר זה שני צדי הקשר אינם פונקציונליים, כי לא ניתן להגדיר פונקציית התאמה בכיוון כלשהו. לדוגמה, הקשר משתתף בין הקבוצות עובדים ופרויקטים הוא קשר מסוג N:M, כי עובד אחד יכול להשתתף במספר פרויקטים, ובפרויקט אחד יכולים להשתתף מספר עובדים. מקובל לסמן קשר מסוג זה על ידי משולש בכל אחד מצדי הקשר.



תרשים 3.33: קשר רב-רב-ערכי בין שתי קבוצות.

במשך השנים השתנו צורות הרישום של הפונקציונליות, וניתן למצוא בספרים שונים העוסקים במודלים של ישויות-קשרים מספר סמלים שונים. תרשים 3.34 מציג צורות רישום מקובלות.



תרשים 3.34: צורות ייצוג שונות לפונקציונליות של קשר רב-רב-ערכי.

בהמשך נשתמש בשיטה א' לייצוג הפונקציונליות במודל התפישתי.

קרדינליות הקשר (Relationship Cardinality)

הפונקציונליות של הקשר לא התייחסה במפורש למספר הישויות המשתתפות בכל צד של הקשר אלא רק עשתה את ההבחנה הבסיסית האם זו ישות אחת או יותר. לפעמים הגדרה זו כללית מדי, מאחר ויש צורך להגדיר אילוצים מפורשים – לפחות אחד, לא פחות מארבע ולא יותר מ-20 וכדי. לשם הגדרה מפורשת של האילוץ משתמשים במושג **קרדינליות** הקשר. הקרדינליות מצטרפת לפונקציונליות כדי להגדיר את הקשר בצורה מדויקת יותר.

קרדינליות הקשר **Relationship Cardinality**	***קרדינליות הקשר* מוגדרת כמספר הישויות המינימלי והמקסימלי בקבוצת ישות J הקשורות לישות אחת בקבוצת ישות I.**

לדוגמה, הקשר מנהל בין הקבוצות עובדים לבין מחלקות הוא מסוג 1:1, אולם הקשר אינו סימטרי. עובד אחד לכל היותר **יכול** לנהל מחלקה (כלומר, חלק מהעובדים אינם מנהלי מחלקות), אולם לכל מחלקה **חייב** להיות מנהל אחד. אי-סימטריות זו ניתנת לביטוי על ידי הקרדינליות, אם נגדיר את המספר המינימלי והמקסימלי של הישויות המשתתפות בקשר. בדוגמה שלנו, הקשר בין עובד למחלקה יהיה בעל קרדינליות (0,1), כלומר עובד יכול שלא להיות קשור למחלקה, או להיות קשור למחלקה אחת לכל היותר. הקשר ההפוך בין המחלקה לבין העובד יהיה בעל קרדינליות (1,1), שבו מחלקה חייבת להיות קשורה לעובד אחד בלבד.

תרשים 3.35: צורת סימון עבור קרדינליות של קשר.

נעיין בדוגמה נוספת של הקשר מועסק בין קבוצות הישות עובדים לבין פרויקטים. נניח שידועים האילוצים הבאים: עובד יכול להשתתף בארבעה פרויקטים לכל היותר ופרויקט יכול להעסיק בין 1 ל-20 עובדים. הקשר הוא מסוג N:M, אולם ללא סימון קרדינליות לא ניתן לבטא את האילוצים האלה. תרשים 3.36 מציג את הקשר עם האילוצים.

תרשים 3.36: קשר עם אילוצי קרדינליות.

במקרים שבהם אין אילוצי קרדינליות ניתן לרשום $(0, \infty)$: סימון ערך מינימלי אפס וערך מקסימלי אינסוף. לחילופין אפשר לא לרשום את אילוץ הקרדינליות כלל.

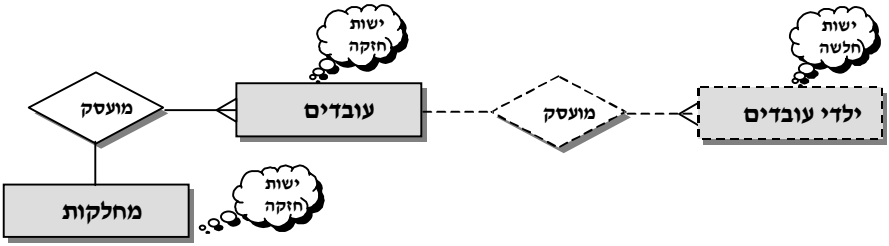
מקובל לקרוא לאילוץ הקרדינליות גם בשם אילוץ השתתפות בקשר (Participation Constraint). מקובל לקרוא לקשר **מנדטורי** או **מוחלט** (Total), אם כל ישות חייבת להיות קשורה לישות אחרת כלשהי; או לומר שזהו קשר **רשות** או **חלקי** (Partial) כאשר רק חלק מהישויות חייבות להשתתף בו. ניתן לבטא אילוצים אלה על ידי סימון הקרדינליות.

תלות קיומית (Existence Dependence)

תלות קיומית **Existence Dependence**	***תלות קיומית*** **בין קבוצה A ל-B מוגדרת כמצב שבו קיום ישות בקבוצת ישות A מותנה בקיום ישות בקבוצת ישות אחרת B.**

תלות קיומית היא למעשה מצב של **ישות חלשה** (Weak Entity), שכבר הזכרנו. לדוגמה, הקשר בין קבוצות הישות ילדי העובדים וקבוצת הישות עובדים הוא כזה, שישות בקבוצה עובדים יכולה להתקיים באופן בלתי תלוי, ואילו ישות בקבוצה ילדי העובדים יכולה להתקיים רק אם היא קשורה לישות כלשהי בקבוצת הישות עובדים. במקרה זה קיימת תלות קיומית בין הקבוצה ילדי העובדים לבין הקבוצה עובדים. בין הקבוצות עובדים ומחלקות לא קיימת תלות קיומית כלשהי, והמשמעות שכל ישות בכל אחת מהקבוצות יכולה להתקיים באופן בלתי תלוי מקיום ישות בקבוצה האחרת.

מבחינה גרפית, המלבן והמעוין של הישות החלשה יסומנו בקווים מרוסקים.



תרשים 3.37: ישויות חזקות וחלשות.

תלות בזמן

קשר יכול להשתנות עם הזמן. לדוגמה, הקשר מועסק בין קבוצות הישות עובדים לבין קבוצת הישות מחלקות יכול להיות מסוג 1:N, אם הוא מייצג את הקשר בין העובדים שמועסקים ברגע נתון על ידי מחלקה מסוימת. אם יש לארגון עניין לנהל את ההיסטוריה של תעסוקת העובד, קשר מסוג זה הופך לקשר N:M, כי אותו עובד יכול לעבוד במספר מחלקות שונות במהלך תקופת עבודתו בארגון.

מעצב מודל הנתונים צריך לתת את דעתו למימד הזמן כדי לקבוע את פונקציונליות הקשר המתאימה בין קבוצות הישויות. יש חשיבות רבה לאפיון נכון של סוג הקשר בזמן המיפוי של המודל התפישתי למודל לוגי. בפרק הבא נראה שצורת המימוש של קשר חד-רב-ערכי שונה מצורת המימוש של קשר רב-רב-ערכי. שינוי סוג הקשר לאחר שתוכניות יישום כבר נכתבו, עלול לגרור ביצוע שינויים בתוכניות היישום.

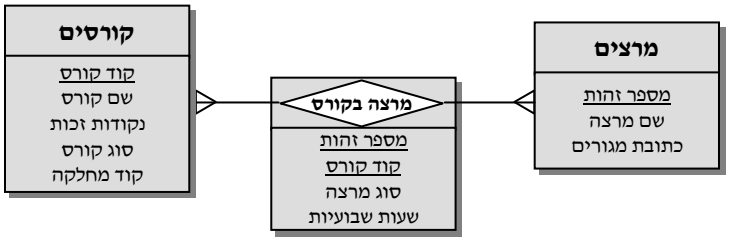
קשר נושא מידע

עד עכשיו התייחסנו אל הקשר כיחס הקיים בין ישויות שונות בקבוצות ישות שונות או באותה קבוצת ישות. בנוסף לקשר המתקיים בין הישויות הוא גם יכול לשאת מידע, שאינו אלא תכונות, בדומה לתכונות שיש לכל ישות.

קשר נושא מידע	***קשר נושא מידע* מוגדר כקשר המכיל את המפתחות העיקריים של שתי קבוצות ישויות שהוא קושר ביניהן, וגם תכונות נוספות.**

לדוגמה נתייחס לקשר מלמד שהוא בעל פונקציונליות N:M בין קבוצות הישות מרצים לבין קורסים. כאשר רוצים לדעת רק איזה מרצה מלמד באיזה קורס ולהיפך, לפנינו קשר רגיל שאינו נושא מידע. נניח שבקורס אחד יכולים ללמד מספר מרצים. מבין כל המרצים בקורס, מרצה אחד מוגדר כמרצה ראשי והאחרים הם כמרצים עוזרים. בנוסף, מספר השעות לשבוע שכל מרצה מלמד אינו אחיד. במצב זה, המידע על סוג המרצה (ראשי או עוזר) ומספר השעות השבועיות שלו הן **תכונות** של הקשר. ניתן לראות שתכונות אלו שייכות לקשר בין שתי הקבוצות ולא לקבוצת ישות כלשהי בנפרד. במקרה זה, בדומה לקבוצת ישות, הקשר כולל תכונות.

נסמן קשר נושא מידע על ידי מעוין מוקף במלבן, כרמז לכך שהקשר הוא גם קבוצת ישות.



תרשים 3.38: קשר נושא מידע.

נשים לב שבתרשים 3.38 רשמנו גם את המפתח של הקשר, שבמקרה שלנו הוא מפתח המורכב מהתכונות מספר זהות וקוד קורס, והוספנו את התכונות המיוחדות לקשר.

ייצוג קבוצות ישות וקשרים על ידי טבלאות

מערכות המידע צריכות לנהל את המושגים שנסקרו עד כה – קבוצת ישות, תכונות של ישות וקשרים בין ישויות – ולכן הוצעו במשך השנים שיטות שונות לייצוג מושגים אלה במערכות ממוחשבות. השיטות שהיו נפוצות בשנים הראשונות היו קרובות מאוד לצורת הייצוג במחשב. קבוצת הישות הוגדרה כקובץ, הישות – כרשומה, והתכונה – כשדה ברשומה. הקשר בין הישויות היה מיוצג בשיטות שונות, כמו כתיבת תכונות קשר בשתי רשומות, או על ידי שימוש במצביעים (Pointers).

עם הופעת המודל הטבלאי, שהוצג לראשונה בשנת 1970 על ידי E.F. Codd, נוצרה הסכמה רחבה שהדרך הנוחה והברורה ביותר לייצג את כל המושגים היא באמצעות **טבלה**. Codd הציע את ההקבלה הזו: קבוצת ישות תיוצג על ידי טבלה, הישות – על ידי שורה בטבלה, והתכונה – על ידי עמודה בטבלה. הקשרים ייוצגו על ידי הוספת עמודות קשר בטבלאות שונות.

תרשים 3.39 מציג את הישויות והתכונות על ידי טבלה שכותרתה קבוצת הישות סטודנטים. כל שורה מייצגת מופע אחד של הישות, סטודנט מסוים, וכל עמודה מייצגת תכונה כלשהי, כמו מספר סטודנט, שם סטודנט, עיר מגורים.

סוג ישות

תכונה

סטודנטים

מס .סטודנט	שם סטודנט	עיר
105	Moshe	Haifa
210	Dan	Tel Aviv
107	Eyal	Tel Aviv
110	Ran	Haifa
245	Yoel	Haifa
240	Ayelet	Tel Aviv
200	David	Tel Aviv
310	Tova	Jerusalem

ישות

תרשים 3.39: טבלה המייצגת את קבוצת הישות סטודנטים.

כעת נייצג קשר בין קורס לבין מחלקה אקדמית שמציעה אותו, על ידי יצירת קשר בין קבוצת הישות קורסים לבין קבוצת הישות מחלקות.

מפתח עיקרי

מפתח זר

קורסים

מס .קורס	שם קורס	סוג קורס	נק .זכות	קוד מחלקה
C-200	Programming	LAB	4	CS
C-300	Pascal	LAB	4	CS
C-55	Data Base	CLASS	3	CS
M-100	Linear Algebra	CLASS	3	MT
M-200	Numeric Analysis	CLASS	3	MT
B-10	Marketing	CLASS	2	BS
B-40	Operations Res.	SEMIN	3	BS

מפתח עיקרי

מחלקות

קוד	מחלקה	ראש מחלקה
CS	Computer Science	Dr. Israel
MT	Mathematics	Prof. Levy
BS	Business	Dr. Eyal
CH	Chemistry	Prof. Doron

תרשים 3.40: קשר בין טבלת מחלקות לבין טבלת קורסים.

נוכל לראות בתרשים שתי טבלאות שלכל אחת מהן מפתח עיקרי. כדי להגדיר קשר בין שתי טבלאות אלו הוספנו את התכונה קוד מחלקה לטבלת הקורסים. בכל אחת מהשורות של קורס מופיע קוד המחלקה שמציעה את הקורס. מקובל לקרוא לתכונה זו בשם **מפתח זר** (Foreign Key). זו הפנייה מטבלה אחת אל **מפתח ראשי** ששייך לטבלה **אחרת**.

על ההקבלה הזו בין המושגים המופשטים של **קבוצת ישות**, **ישות ותכונה** לבין המושגים הממשיים של **טבלה**, **שורה ועמודה** מבוססות המערכות לניהול בסיסי נתונים טבלאיים שהן נושא ספר זה.

תרשים ישויות-קשרים (Entity-Relationship Diagram)

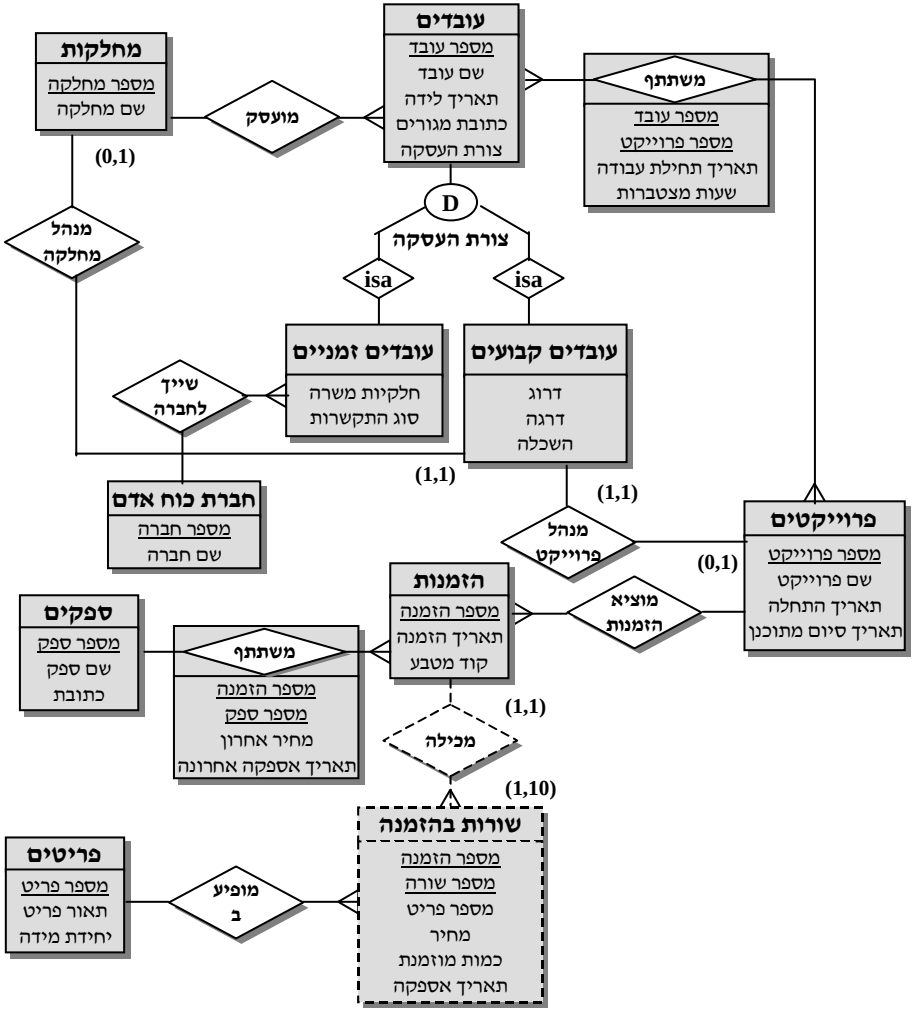
תרשים ישויות-קשרים מציג באופן גרפי את הנושאים האלה:

- אוסף כל קבוצות הישות הרלוונטיות.
- אוסף כל התכונות של כל קבוצה.
- המפתח העיקרי של כל קבוצה.
- אוסף כל הקשרים בין הקבוצות השונות, או בין הישויות השונות של אותה קבוצה.
- אוסף התכונות של כל קשר נושא מידע.
- זיהוי פונקציונליות הקשר (N:M, 1:N, 1:1).
- אילוצי קרדינליות הקשר, במידה וקיימים.

היתרון העיקרי של תרשים ישויות-קשרים הינו בכך שהוא כלי גרפי פשוט ובלתי תלוי בצורת מימוש המודל במערכת ממוחשבת כלשהי. קל להשתמש בתרשים ככלי תקשורת בין מנתחי המערכות, מעצבי בסיס הנתונים והמשתמשים.

דוגמה א': יש לבנות תרשים ישויות-קשרים עבור חברה תעשייתית. בחברה מספר מחלקות המעסיקות עובדים ומבצעות פרויקטים שונים. החברה מעסיקה עובדים קבועים ועובדים זמניים. את העובדים הזמניים היא שוכרת מחברות השמה שונות. במחזור חייו של פרויקט יוצאות הזמנות לרכישת פריטים שונים, שכל אחת מהן נשלחת לספק אחד בלבד ויכולה להכיל עד 10 שורות של פריטים שונים.

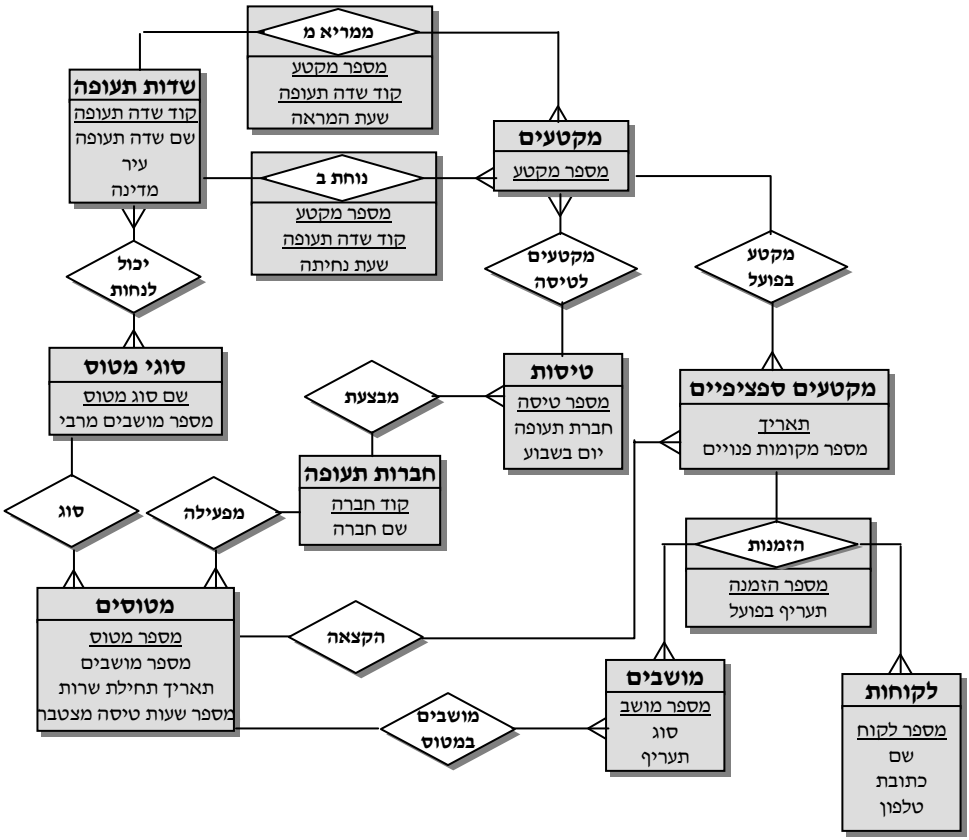
בתרשים 3.41 מוצג מודל תפישתי פשטני מאוד לעומת מודל אמיתי. בדרך כלל, לא ניתן לרשום על התרשים את כל התכונות של כל קבוצת ישות או קשר, ומקובל להשתמש בנספח המציג את כל התכונות. על התרשים מציגים את קבוצות הישות בלבד, את המפתחות שלהן ואת הקשרים בין הקבוצות.



תרשים 3.41: מודל תפישתי פשטני של ישויות-קשרים - ניהול פרויקטים.

דוגמה ב': יש לבנות תרשים ישויות-קשרים של חברה המאגדת מספר חברות תעופה שונות ומפעילה מספר מטוסים מסוגים שונים. בכל מטוס מספר מושבים מסוגים שונים (מושבים במחלקת תיירים, במחלקת עסקים, במחלקה ראשונה) ולכן החברה מוכרת אותם בתעריפים שונים. מטוסים אלה מורשים לנחות בשדות תעופה מסוימים בלבד (למשל מטוס מסוג קונקורד לא יכול לנחות בשדה תעופה אילת). כל חברת תעופה מפעילה מספר טיסות היוצאות בימים מסוימים של השבוע משדות תעופה מסוימים לשדות יעד כלשהם. כל טיסה מחולקת למספר קטעים (Legs) מתוכננים. לדוגמה, הטיסה מתל אביב לניו יורק יכולה להיות מחולקת לשני קטעים – תל אביב לונדון ולונדון ניו-יורק. בכל קטע, המטוס ממריא משדה תעופה מסוים בשעה כלשהי ונוחת בשדה תעופה מסוים בשעה אחרת כלשהי. כל קטע מתבצע בפועל בתאריכים שונים על ידי מטוסים שונים. לכל קטע בפועל מקבלת החברה הזמנות מלקוחות ומקצה להם מושב מסוים לתאריך מסוים.

התעריפים בהזמנות יכולים להיות שונים מהתעריף המוצע לכל מושב בגלל הנחות שונות שהחברה נוהגת להציע ללקוחותיה.



תרשים 3.42: תרשים ישויות-קשרים - איגוד חברות תעופה.

סיכום

ניתן לראות את המודל התפישתי כשלב ביניים, שבו מנתח המערכת, או מעצב בסיס הנתונים, בונה מודל נתונים המשוחרר מאילוצים ומתמקד בהגדרה של הישויות, התכונות שלהן והקשרים ביניהן. במובן זה, המודל שמתקבל הוא "טהור וטבעי". היתרונות של בניית מודל תפישתי בשלב ראשון ובניית מודל לוגי בשלב שני הם:

- עבודת עיצוב בסיס הנתונים פשוטה ביותר בעקבות חלוקת המטלה לשני שלבים בעלי פלט מוגדר.
- קל יותר לתכנן את המודל התפישתי מאשר את המודל הלוגי, משום שהוא משוחרר מאילוצים ומשיקולי יעילות.

- המודל התפישתי הוא מסמך חשוב המתעד את מודל הנתונים של הארגון, בשעה שהמודל הלוגי מותאם למערכת DBMS מסוימת. לכן, קשה לעיתים להסיק מהמודל הלוגי מה היה המבנה המקורי של המודל התפישתי לפני שהופעלו עליו אילוצים.
- המודל התפישתי ברור יותר למשתמשים, ולכן קל יותר להסביר אותו ולקבל הערות ותיקונים.

המודל התפישתי הינו התוצר העיקרי של תהליך ניתוח הנתונים וחייב לשרת מספר רב של משתמשים ותוכניות יישום. לכל משתמש או תוכנית יישום עשויה להיות נקודת מבט שונה על הנתונים. זהו **מודל הנתונים של המשתמש** או **מבט המשתמש** (User View), או כמו שהוא קרוי לעיתים, **מודל מקומי** (Local View).

לתהליך שבו בונים מודל תפישתי אחד גלובלי מתוך אוסף של מודלים מקומיים, קוראים בשם **שילוב מודלים**. לעיתים זהו תהליך מורכב, כי בין המודלים המקומיים השונים ייתכנו סתירות שונות. לדוגמה, שמות שונים לאותה תכונה, שמות זהים לתכונות שונות, חוסר עקביות בתיאור הקשרים או בסוג הקשרים וכד׳. תהליך שילוב המודלים צריך להתמודד עם מצבים אלה ולפתור אותם במטרה לקבל מודל תפישתי אחד שיכול לשרת את כולם. תהליך השילוב פותר סתירות סמנטיות, ולכן אין תחליף להבנה של מעצב המערכת וליכולת שלו לאתר את המצבים האלה ולפתור אותם.

השלבים העיקריים הם:

- הבנת דרישות המידע והעיבוד בהתבסס על המציאות הרלוונטית ליישום.
- בניית מודלים מקומיים. כל מודל כזה מבטא את נקודת המבט המקומית (של משתמש, של תוכנית יישום או של פעילות מסוימת) על הנתונים.
- שילוב המודלים המקומיים למודל תפישתי גלובלי אחד, תוך פתרון הסתירות השונות, במידה וקיימות. המודל הגלובלי המתקבל הינו מנורמל. את המודל הגלובלי מציגים באמצעות תרשים ישויות-קשרים ועל ידי אוסף של טבלאות לתיאור קבוצה.

שאלות חזרה ותרגילים

שאלות חזרה

1. הסבר מהם ההבדלים בין שלושת המודלים: מודל תפישתי, מודל לוגי, מודל פיסי. מדוע יש חשיבות להבחין בין שלושת סוגי מודלים אלה.
2. הסבר מהו התהליך לבניית מודל תפישתי גלובלי מתוך מודלים תפישתיים מקומיים. מדוע לא לבנות ישר את המודל התפישתי הגלובלי?
3. הגדר את המושגים: *ישות, קבוצת ישות, תכונה, ערך, תכונה מרובת ערכים, מרחב ערכים, קשר, קבוצת קשר, טיפוס נתונים, מפתח עיקרי, מפתח אפשרי*.
4. הסבר מהם המאפיינים של קשר בין קבוצות ישויות.

5. הסבר מהו קשר רפלקסיבי. תן מספר דוגמאות לקשרים מסוג זה.

6. מהם ההבדלים בין קרדינליות קשר לבין פונקציונליות הקשר. מדוע יש צורך בשני המושגים?

7. הסבר מהם תהליכי ההפשטה וכיצד תהליכים אלה משתתפים בתהליך עיצוב מודל הנתונים.

8. הגדר את כל סוגי המפתחות שאתה מכיר. מדוע יש חשיבות למושג המפתח בבניית מודל הנתונים ?

תרגילים

1. חברה העוסקת בהפצת סרטי קולנוע מעוניינת לבנות מערכת מידע שתנהל את הנתונים הבאים: שם סרט, אורך הסרט בדקות, סוג הסרט (קומדיה, מתח, פעולה, דרמה, סרט מצויר וכד'), תאריך הפקת הסרט, שם שחקן הקולנוע המשתתף בסרט, שנת לידה של שחקן הקולנוע, ארץ לידה של שחקן הקולנוע, כתובת נוכחית של שחקן הקולנוע, שם החברה שהפיקה את הסרט, כתובת החברה שהפיקה את הסרט. בסרט קולנוע אחד יכולים להשתתף מספר שחקני קולנוע ושחקן קולנוע אחד יכול להשתתף במספר רב של סרטים. סרט קולנוע יכול להיות מופק על ידי מספר חברות הפקה וחברת הפקה יכולה להפיק מספר סרטים. בנה בסיס נתונים מנורמל המתאים לניהול נתונים אלה. עצב את בסיס הנתונים המתאים.

2. בהמשך לבעיה המתוארת בתרגיל 1 לעיל, נניח שחברת ההפצה מבקשת לשלם תמלוגים לשחקני הקולנוע בגין הפצת הסרטים. לכל שחקן יש חוזה נפרד עם החברה המפיקה עבור כל סרט שהוא משחק בו. חוזה זה מכיל נתונים כגון מספר החוזה, שם שחקן הקולנוע, שם חברת ההפקה, שם סרט הקולנוע, תאריך החוזה, אחוז מהתמלוגים שמגיע לשחקן, תקופה בשנים שבה זכאי השחקן לקבל את התמלוגים. לצורך הפשטות נניח שאם הסרט הופק על ידי מספר חברות הפקה, החוזה של שחקן הקולנוע הוא עם אחת מבין החברות האלו. עדכן את תרשים ישויות-קשרים כך שישקף את דרישת המידע החדשה.

3. בהמשך לבעיה המתוארת בתרגילים 1 ו-2 לעיל, נניח שלכל סוג סרט יש לנהל מספר נתונים ייחודיים. לדוגמה עבור סרט פעולה יש לנהל על רמת הפעלולים בסרט ואילו עבור סרט מצוייר יש לדעת האם הוא הופק בטכניקה דו-מימדית או תלת-מימדית ועל איזה ספר מבוססת העלילה. עדכן את תרשים ישויות-קשרים כך שישקף את דרישת המידע החדשה.

4. נניח שאנו מבקשים לנהל מידע לכל סטודנט במכללה על התארים הקודמים שלו. לכל תואר קודם יש לנהל מידע כגון שם המוסד להשכלה גבוהה, סוג המוסד (אוניברסיטה, מכללה, מכון ללימודים מתקדמים וכד'), תאריך תחילת לימודים, תאריך סיום לימודים, תארים אקדמיים. סטודנט יכול לקבל מספר תארים אקדמיים במהלך לימודיו במוסד להשכלה גבוהה. לדוגמה הוא יכול לקבל תואר בכלכלה ותואר בסטטיסטיקה או למשל תואר בחינוך וביחסים בינלאומיים. עבור כל

אחד מהתארים הנ״ל יש לנהל את הציון הסופי. עצב את קבוצת הישות של הסטודנט כך שנוכל לנהל מידע זה.

5. הנהלת הליגה לכדורגל החליטה לבנות מערכת מידע לניהול פעילותה. להלן הנתונים הרלוונטיים:

א. בליגה פעילות מספר אגודות ספורט. לכל אגודה יש שם, יושב ראש וכתובת משרד. לכל אגודה יכול להיות מספר קבוצות פעילות (לדוגמה, לאגודת הפועל יש מספר קבוצות בליגה – הפועל תל-אביב, הפועל באר-שבע וכד׳).

ב. לכל קבוצת כדורגל יש שם, יושב ראש וכתובת של המשרד.

ג. לכל קבוצת כדורגל יש מאמן. לכל מאמן יש לנהל נתונים כגון: שם מאמן, כתובת פרטית, האם סיים קורס מאמנים ותאריך סיום קורס מאמנים. מאמן יכול לאמן במהלך הזמן מספר קבוצות. למען הפשטות נניח, שבמהלך העונה הוא יכול לאמן רק קבוצה אחת, אולם בעונות שונות הוא יכול לאמן קבוצות שונות. כמובן שהוא יכול לאמן את אותה קבוצה מספר עונות.

ד. בכל קבוצה משחקים שחקנים. לכל שחקן יש לנהל נתונים כגון: שם שחקן, תעודת זהות, כתובת פרטית ותאריך לידה. שחקן יכול לשחק בעונות שונות בקבוצות שונות, אולם בעונה אחת הוא יכול לשחק רק בקבוצה אחת. בכל עונה יש לשחקן חוזה חדש הקובע את השכר החודשי ואת הפרמיה על כל נקודת ליגה.

ה. לכל קבוצת כדורגל יש מגרש כדורגל. למגרש כדורגל יש לנהל נתונים כגון: שם מגרש הכדורגל, קיבולת וסוג מגרש (פתוח או מקורה).

ו. קבוצה משתתפת במשחקים נגד קבוצות אחרות. לכל משחק יש לנהל נתונים כגון: מי היא הקבוצה המארחת, מי היא הקבוצה אורחת, באיזה מגרש מתקיים המשחק, שעת המשחק, תוצאת המשחק ותאריך המשחק.

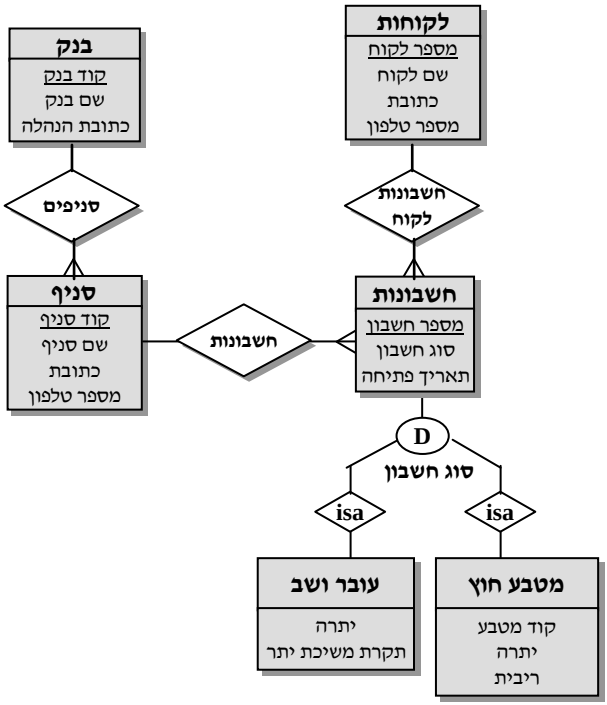
ז. לכל שחקן יש לנהל נתונים אודות מספר השערים שהוא הבקיע ומספר הדקות שהוא שיחק בכל אחד מהמשחקים בהם הוא השתתף.

יש לבנות את מודל הנתונים התפישתי הכולל את כל הישויות, מפתחות, תכונות, קשרים, פונקציונליות. יש להציג תרשים ישויות-קשרים מתאים.

6. בנה תרשים ישויות-קשרים עבור מערכת רכש המטפלת בהזמנת פריטים מספקים שונים, עוקבת אחר קבלת הפריטים המוזמנים למחסני החברה וקולטת את חשבוניות הספק בגין אספקת פריטים אלה. נסה לחשוב מהן הישויות המתאימות לבעיה זו ואיזה קשרים קיימים בין הישויות. רשום את כל ההנחות שהנחת לצורך בניית המודל.

7. עליך לעצב בסיס נתונים רפואי. לכל חולה יש לנהל נתונים על תעודת זהות, שם וכתובת. לכל רופא יש לנהל נתונים על תעודת זהות, שם והתמחויות. חולה מבקר אצל רופא בתאריכים שונים. לכל ביקור יש לנהל נתונים על תאריך הביקור, התרופות שנרשמו, הבדיקות שנערכו והתוצאות של הבדיקות. בנה תרשים ישויות-קשרים מתאים לבעיה זו.

8. תרשים ישויות-קשרים המופיע בתרגיל זה מכיל חלק ממודל של מערכת בנקאית.



א. הסבר באופן מילולי את דרישות המידע שהובילו לבניית המודל הזה.

ב. נניח שכל לקוח חייב להיות בעל חשבון אחד לפחות ולא יותר מ- 20 חשבונות. כיצד תבטא אילוץ זה במודל?

ג. נניח שלקוח יכול להיות בעל חשבונות בסניף אחד בלבד. כיצד תבטא אילוץ זה במודל?

פרק 4

פיתוח ועיצוב מודל הנתונים הטבלאי

1. מבוא
2. רקע היסטורי
3. סקירת מודל הנתונים הרשתי וההיררכי
4. מושגי יסוד בתורת הקבוצות
5. המודל הטבלאי
6. המרת מודל תפישתי למודל טבלאי
7. אלגברה טבלאית (Relational Algebra)
8. סיכום
9. שאלות חזרה ותרגילים

מבוא

לאחר עיצוב מודל הנתונים התפישתי, מגיע השלב השני בתהליך עיצוב בסיס הנתונים – **עיצוב מודל נתונים לוגי** (Logical Data Model Design) – שבו בונים את סכימת בסיס הנתונים. הסכימה מציגה את מכלול הנתונים והקשרים ביניהם בצורה נוחה ככל הניתן, המתאימה לדרישות העיבוד של תוכניות היישום השונות. להבדיל ממודל נתונים תפישתי, הסכימה **תלויה** באופן ברור במערכת ניהול בסיסי הנתונים בה נשתמש.

בפרק הקודם הסברנו שבמשך השנים התפתחו שלושה מודלים לוגיים של נתונים: המודל ההיררכי, המודל הרשתי והמודל הטבלאי. מקובל להתייחס למודל נוסף וייחודי, מודל הנתונים של מערכת Adabas. אחת הבעיות המרכזיות של המודל ההיררכי, הרשתי ושל מערכת Adabas, היא הקושי של משתמשים שאינם אנשי מקצוע בעיבוד נתונים להבין את המודל, עקב הקשרים המסועפים הקיימים בין הנתונים.

מצב זה גרם לכך שבמקביל להופעת מערכות DBMS המבוססות על המודל ההיררכי והמודל הרשתי, לא פסק המאמץ לחפש דרכים ושיטות להצגה פשוטה יותר של מודל הנתונים, ושפה בעלת עוצמה שתאפשר טיפול נוח. זהו הרקע לפיתוח **מודל הנתונים הטבלאי** (Relational Data Model).

המודל הוצג לראשונה על ידי Dr. Edgar F. Codd בשנת 1970, שטען כי היתרון העיקרי של המודל הוא באפשרות ההצגה המדויקת והפשוטה של הנתונים בבסיס הנתונים. הדיוק נובע מכך שהמודל מתבסס על התורה המתמטית של הקבוצות ומשתמש באופרטורים לטיפול **ביחסים** (Relations). הפשטות מושגת על ידי מימוש היחסים במבנה פשוט של **טבלאות** (Tables) המוכר ומובן לכל, ועל ידי טיפול בטבלאות באמצעות מספר קטן של אופרטורים פשוטים ובעלי עוצמה רבה. אחד המאפיינים החשובים של מודל הנתונים הטבלאי הוא: **הפעלת אופרטור על טבלה גורמת ליצירת טבלה חדשה**. בכל מערכות DBMS שלא היו מבוססות על המודלים שקדמו למודל הטבלאי, האופרטורים פעלו על מבנה הנתונים במתכונת של רשומה אחת בלבד.

פשטות המבנה ועוצמת האופרטורים הן רק חלק ממאפייני המודל. מאפיין חשוב נוסף של המודל הטבלאי הוא ההפרדה המאוד ברורה בין המבנה הפיסי של הנתונים לבין המבנה הלוגי שלהם, כפי שהוא מוצג למשתמש. במודל זה אין לתוכנית היישום כל אפשרות להתייחס למבנה הפיסי, כמו למשל למצביעים ולסדר שבו מאוחסנות הרשומות בבסיס הנתונים. במודלים הקודמים, תוכניות היישום היו צריכות "להכיר" את צורת אחסון הנתונים ולהשתמש בפקודות כגון GET NEXT, FIND PRIOR וכד', אשר מבוססות על "ידיעה" זו.

הדרישה להסתרת המבנה הפיסי ברורה מאוד ומטרתה להשאיר את המודל חופשי להכנסת שינויים בעתיד וגמיש מבחינת אפשרויות הגישה לנתונים. כלומר, הכוונה היא להסיר כל מגבלה ולהשיג את מירב היעילות. כמובן ששתי דרישות אלו נראות נוגדות, כי במודלים הקודמים הושגה היעילות (בניית מצביעים מראש בתוך מבנה הנתונים) על חשבון הגמישות (כל שינוי במבנה המודל מחייב שינוי בתוכניות היישום).

התכונות העיקריות של המודל הטבלאי הן פשטות בתכנון המודל, גמישות בעיצוב ובשינויים, יכולת התמודדות עם שינויים במבנה ובניהול הנתונים, ועוצמה בטיפול בנתונים. תכונות אלו הפכו אותו למודל הנפוץ ביותר. מנתח המערכת יכול מעתה לתכנן מבנה נתונים בצורה פשוטה ומובנת גם למשתמש ולבצע שינוי במבנה זה, במידת הצורך. מהנדס התוכנה יכול לטפל בנתונים באמצעות אופרטורים בעלי עוצמה רבה והדבר משפר את הפריון באופן משמעותי. משתמש הקצה יכול להבין את מבנה הנתונים של הארגון ומסוגל לאחזר נתונים באמצעות שפת שאילתות פשוטה, ללא צורך בתיווך יחידת מערכות המידע. הפשטות של המודל שימשה מנוף לפיתוח מערכות תומכות החלטה שמיושמות באמצעות **מערכות מחסן נתונים**, ומאפשרות למשתמשים תחקור נתונים באופן ישיר.

המטרה העיקרית של המודל הטבלאי היא לתת למשתמשים השונים **כלי עבודה**, אשר יהיה פשוט להבנה ולתפעול, ואשר יאפשר להם להציג את **הנתונים** הנדרשים עצמם ולא **איך** לאחזר אותם מתוך בסיס הנתונים. המשתמש במודל הטבלאי מתרכז בהפקת **המידע** ואינו מתרכז בטיפול **ברשומות** כמקובל במודלים הישנים יותר.

מודל הנתונים הטבלאי מתאים יותר לסביבות המחשוב המורכבות של ימינו. התאמה זו הינה תולדה של שני מאפיינים חשובים: **פשטות המבנה**, המתבטאת בשימוש בטבלאות בלבד, ו**עוצמת שפת הטיפול בנתונים**, המאפשרת קבלת תוצאת האחזור בבת אחת (ולא רשומה אחר רשומה כמו במודלים הקודמים). מאפיינים אלה של המודל מאפשרים לבסס עליו שפות תכנות מדור רביעי שאינן פרוצדורליות באופיין; ניתן לבזר את הנתונים ביתר קלות בין מחשבים שונים ברשת תקשורת, כי המודל אינו מכיל מצביעים בין טבלאות; אפשר לבנות סביבו מודל שרת-לקוח, שבו מחשב הלקוח מבקש נתונים כלשהם והשרת מבצע את כל הלוגיקה של האחזור ושולח ללקוח את התשובה בבת-אחת, מבלי להעמיס את רשת התקשורת; אפשר לבסס עליו יישומים מתקדמים הפועלים בסביבת האינטרנט בגלל יכולתו לנהל סוגי נתונים מסוגים שונים, ועוד.

בפרק זה

- נסקור את הרקע ההיסטורי להתפתחות מודל הנתונים הטבלאי.
- נסקור בקצרה את המודלים הלוגיים שהתפתחו במשך השנים: המודל ההיררכי, המודל הרשתי והמודל הטבלאי.
- נציג מספר מושגי יסוד מתורת הקבוצות: קבוצה, פעולות על קבוצה (איחוד, חיתוך, הפרש, מכפלה קרטזית ועוד), יחסים ופונקציות מיפוי בין קבוצות. מושגים אלה דרושים להבנת מודלים של נתונים באופן כללי והמודל הטבלאי בפרט.
- נסקור את מושגי היסוד של המודל הטבלאי: יחסי סכימה טבלאית, שיטת ייצוג הקשרים במודל הטבלאי.
- נציג את האלגברה הטבלאית, שפה תיאורטית שבאמצעותה ניתן לאחזר נתונים מטבלאות שונות.

רקע היסטורי

ההסטוריה של טכנולוגיית בסיסי הנתונים הטבלאיים קצרה יחסית. בשנת 1970 פרסם המתמטיקאי Dr. Edgar F. Codd, שעבד באותה תקופה במעבדות המחקר של חברת יבמ, מאמר [CE70] אשר היווה את אבן היסוד של הטכנולוגיה החדשה. Codd היה באותה תקופה היוזם של פרויקט System R, שמטרתו היתה לחפש שיטות טובות ונוחות יותר לגישה וטיפול בנתונים, בהשוואה למערכות שהיו מוכרות באותה תקופה. במסגרת הפרויקט הוכח שניתן לבנות מערכת DBMS שתאפשר גישה וטיפול נוח בנתונים, אולם מבחינה טכנולוגית מערכות אלו צרכו משאבי מחשב גדולים מאוד, שלא היו זמינים וזולים באותה עת.

במסגרת המאמר הגדיר Codd גישה מתמטית להגדרה וטיפול בנתונים ולראשונה הוכח, שבתנאים מסוימים ניתן להתייחס אל קובץ כאל "יחס" (Relation) מתמטי, מושג הלקוח מתורת הקבוצות. גם קשרים בין קבצים, שהם מרכיב מרכזי בטכנולוגיית בסיסי נתונים, ניתנים לייצוג על ידי קבוצה ואין צורך להשתמש במבנה מיוחד (המצביע) לשם כך. בהתבסס על אבחנה זו, הציג Codd שפה המורכבת ממספר אופרטורים לטיפול בקבוצות.

המאמר עורר תהודה בעיקר באוניברסיטאות ולא בקהיליית המשתמשים, כי השימוש בטרמינולוגיה מתמטית לא "דיבר" אליהם. נדרשו יותר מ-10 שנות מחקר והתפתחויות הן בחומרה והן בתוכנה עד למימוש הטכנולוגיה החדשה. בין השנים 1980 ל-1985 הופיעו המערכות המסחריות הראשונות והעיקריות המבוססות על טכנולוגיה זו: מערכת Oracle של חברת Oracle Corp., מערכת Ingress של חברת Computer Associates (מערכת זו פותחה במקור על ידי Relational Technology Inc., נמכרה לחברת Ask ונמצאת כיום בבעלות חברת CA), מערכת Rdb של חברת Oracle Corp. (מערכת זו פותחה במקור על ידי חברת Digital ונמכרה לפני מספר שנים לחברת Oracle) ומערכת SQL/DS של חברת יבמ (ששימשה בסיס לפיתוח המערכת DB/2 של חברת יבמ למחשבים מרכזיים ומחשבי ביניים, ולשרתים הפועלים בסביבת Unix). המערכות הראשונות "סבלו" בתחילת הדרך מבעיות ביצועים בעת עיבוד בסיסי נתונים גדולים ומורכבים, ולכן השימוש בהן היה מוגבל.

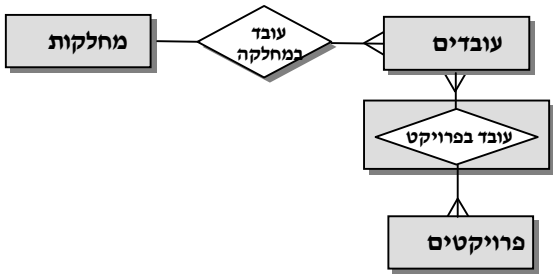
המשך גידול עוצמת המחשבים, הופעת מערכת ההפעלה Unix וחדירתה לעולם עיבוד הנתונים, במקביל להמשך שיפור מתמיד של מערכות RDBMS, הביאו לכך שהן הפכו לפופולריות. מערכות חדשות המשיכו להופיע, וביניהן מערכת Informix Dynamic Server של חברת Informix Software Inc., מערכת Sybase ASE של חברת Sybase, מערכת SQL Server של חברת Microsoft ומספר רב מאוד של מערכות נוספות.

בשנת 1981 זכה Codd בפרס היוקרתי ביותר בתחום המחשוב, ACM Turing Award, המוענק על ידי האגודה האמריקאית לעיבוד אינפורמציה (ACM), על תרומתו החשובה לעולם המחשוב, הן בפיתוח המודל הטבלאי והן בבניית היסודות להבנה טובה יותר של שיטות לעיצוב מודלים של נתונים (תהליך הנירמול שיוצג בפרק הבא).

סקירת שלושת המודלים הקלאסיים

לפני דיון מפורט במודל הטבלאי, נציג בקצרה את שלושת המודלים הלוגיים החשובים והנפוצים: המודל הרשתי, המודל ההיררכי והמודל הטבלאי. המודל הטבלאי מוצג כאן רק כדי לתת לקורא תמונה כללית, ובהמשך נרחיב את הדיון בו. כפי שנראה, ניתן לתאר את המציאות בעזרת כל אחד מהמודלים במידה שונה של פשטות, בהירות ונוחות. המודלים הוותיקים יותר מייצגים את הישות באמצעות הרשומה, בעוד המודל הטבלאי מייצג את הישות על ידי שורה בטבלה. ההבדל המהותי בין המודלים בא לידי ביטוי בצורת ניהול הקשרים.

לצורך הצגת המודלים האלה נשתמש במודל תפישתי המכיל שלוש קבוצות ישות ושני קשרים ביניהם.



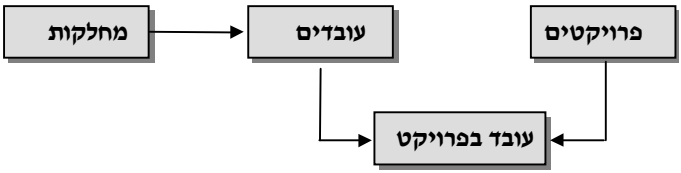
תרשים 4.1: מודל תפישתי.

המודל מכיל את קבוצות הישות האלו: מחלקות, פרויקטים, עובדים. בין הקבוצות קיים קשר עובד במחלקה, המגדיר קשר חד-רב-ערכי מקבוצת המחלקות לקבוצת העובדים, והקשר עובד בפרויקט המגדיר קשר רב-רב-ערכי מקבוצת הפרויקטים לקבוצת העובדים. כפי שניתן לראות, הקשר עובד בפרויקט הוא קשר נושא מידע.

דיון מקיף יותר במודל הרשתי ובמודל ההיררכי ניתן למצוא בספריו של המחבר "**בסיסי נתונים טבלאיים ושפת SQL**" [HR93] ו"**בסיסי נתונים – עקרונות, מודלים ויישומים**" [HR84] – שניהם ב**הוצאת הוד-עמי**.

המודל הרשתי (Network Data Model)

המודל הרשתי הוגדר בצורה פורמלית על ידי ארגון CODASYL, שהקים צוות עבודה בנושא בסיסי נתונים בתחילת שנות ה-70. המודל מאופיין על ידי הגדרת שני סוגי רשומות: **רשומת אב** (Owner Record) **ורשומת בן** (Member Record). בין שני סוגי רשומות אלו קיים קשר חד-רב-ערכי (1:N). משמעות הדבר היא שמודל הנתונים הרשתי רואה את הנתונים כאוסף של סוגי רשומות שונים הקשורים ביניהם בקשר חד-רב-ערכי בלבד. כדי לטפל בקשר רב-רב-ערכי, המודל מחייב לפרק אותו לשני קשרים מסוג חד-רב-ערכי.



תרשים 4.2: ייצוג המודל התפישתי על ידי מודל רשתי.

הקשר החד-רב-ערכי בין רשומת אב לרשומת בן יסומן על ידי חץ הפונה מרשומת האב לרשומת הבן. כפי שניתן לראות, המודל מטפל בקלות במודל התפישתי והשוני העיקרי הוא בהוספת סוג רשומה חדש, רשומת עובד בפרויקט, שתכיל את נתוני הקשר.

המודל מאפשר לקשור רשומת אב אחת למספר סוגי רשומות בן שונות, וגם מאפשר לרשומת בן אחת להיות קשורה לסוגי רשומות אב שונים. כל קשר ביניהם יהיה תמיד מסוג חד-רב-ערכי.

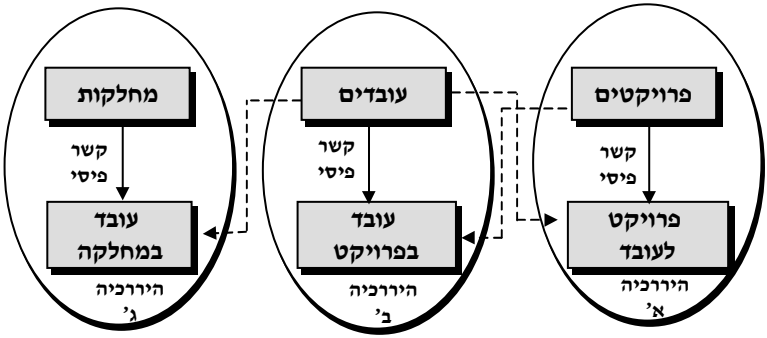
המודל הרשתי יושם במספר רב של מערכות DBMS מסחריות ושימש נושא לעבודות מחקר שונות. המודל מאפשר גישה יעילה מאוד לנתונים, אך דורש ממהנדס התוכנה להכיר את מבנה בסיס הנתונים ואת מסלולי הגישה לנתונים. לדוגמה, תוכנית היישום חייבת להכיר את המבנה המופיע בתרשים שראינו ולנווט בתוך מבנה זה באמצעות פקודות מיוחדות. למשל, קיימת פקודה מיוחדת להגיע ישירות אל רשומת אב באמצעות מפתח, פקודה מיוחדת הקוראת את רשומת הבן הראשונה בקשר, פקודה הקוראת את הרשומה הבאה בקשר ופקודה מיוחדת המאפשרת לקרוא את רשומת האב הקשורה לרשומת בן מסוימת. חוסר הגמישות של המודל מתבטא בעובדה ששינוי מסלולי גישה מחייב שינוי בתוכניות היישום.

המודל הרשתי מתבסס על אוסף של מצביעים אשר קושרים את הרשומות השונות. הוספת קשר חדש בין סוגי רשומות, מחייב לעיתים את פריקת בסיס הנתונים, שינוי ההגדרות בסכימה וטעינה מחדש של הרשומות, מכיון שהמצביעים הם חלק מן הרשומות (תופסים מקום פיסי).

המודל ההיררכי (Hierarchical Data Model)

המודל ההיררכי ותיק יותר מהמודל הרשתי ונמצא בשימוש זמן רב במערכות DBMS מסחריות. המודל פותח על ידי חברת יבמ ומשמש את מערכת IMS, שהיתה משך תקופה ארוכה אחת ממערכות DBMS הנפוצות והיעילות ביותר. מערכת זו ממשיכה לפעול גם כיום ומשרתת מספר רב של ארגונים גדולים מאוד.

גם המודל ההיררכי משתמש בשני סוגי רשומות, **רשומת אב ורשומת בן**, שביניהן קיים קשר חד-רב-ערכי (1:N). ההבדל העיקרי בין מודל זה לבין המודל הרשתי נובע מהעובדה שרשומת בן יכולה להיות קשורה לרשומת אב אחת בלבד. המבנה המתקבל כתוצאה מצורת קשר זו הוא מבנה היררכי של **עץ נתונים**, שבו יש לכל רשומה בעץ מסלול גישה אחד בלבד.



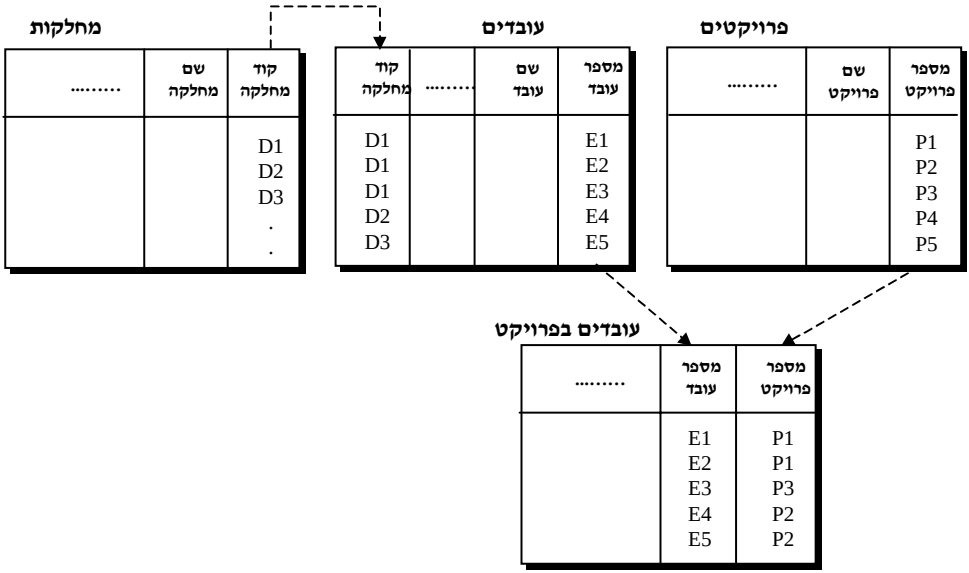
תרשים 4.3: ייצוג המודל התפישתי על ידי מודל היררכי.

כפי שניתן לראות ממבט ראשון, מודל זה מורכב יותר מהמודל הרשתי, כי הוא מחייב לפרק את כל הקשרים המופיעים במודל התפישתי לאוסף של היררכיות נפרדות. בדוגמה שלנו צריך היה לבנות שלוש היררכיות שונות. בתוך ההיררכיות מתקיימים קשרים פיסיים ובין ההיררכיות מתקיימים קשרים לוגיים (מופיעים בקו מרוסק בתרשים). למרות שבאופן מעשי מימוש הקשרים הפיסיים והלוגיים הוא באמצעות אוסף של מצביעים, קיימים כללים נוקשים של מה מותר ומה אסור לבצע עם הקשרים השונים. כללים אלה גורמים לבניית מודל נתונים מורכב יותר ממודל רשתי, מפני שבמודל הרשתי מיוצגים הקשרים בצורה טבעית יותר. לכל רשומה יש מסלול גישה יחיד, ולכן ניתן ליישם את המודל בצורה פשוטה יותר.

בדומה למודל הרשתי, גם המודל ההיררכי מכיל אוסף פקודות ייחודיות לטיפול בהיררכיות, וגם כאן תוכנית היישום תלויה במבנה הנתונים. כל שינוי במבנה הנתונים מחייב גם שינוי בתוכניות היישום ולעיתים כרוך גם בפריקת וטעינת בסיס הנתונים כדי לפנות מקום למצביעים חדשים.

מודל הנתונים הטבלאי (Relational Data Model)

המודל הטבלאי מאפשר תיאור הנתונים בצורה טבעית ביותר, על ידי אוסף של טבלאות ואין כל צורך להוסיף מבנים מיוחדים שמטרתם העיקרית היא סיוע לייצוג הנתונים במחשב. בכך הוא מאפשר אי-תלות נתונים מרבית. המודל הטבלאי מייצג את **כל** הנתונים באמצעות טבלאות, ובו כל טבלה מתאימה לקבוצת ישות, כל שורה לישות וכל תכונה מיוצגת על ידי עמודה. ייצוג הקשרים בין הישויות מתאפשר באמצעות עמודות זהות בטבלאות שונות.



תרשים 4.4: ייצוג המודל התפישתי על ידי מודל טבלאי.

כפי שניתן לראות, המודל הטבלאי הופך כל קבוצת ישות לטבלה בעלת עמודות. את הקשרים בין קבוצות הישות מממש המודל הטבלאי על ידי עמודות זהות המופיעות בטבלאות שונות, ותוך התבססות על העובדה שערכים זהים מופיעים בשתי העמודות. למשל הקשר החד-רב-ערכי בין טבלת מחלקות לבין טבלת עובדים ממומש על ידי הוספת העמודה מספר מחלקה לטבלה עובדים. את הקשר הרב-רב-ערכי בין עובדים ופרויקטים מממש המודל הטבלאי בצורה דומה למודל הרשתי, על ידי הוספת טבלת קשר חדשה המכילה את העמודות משתי הטבלאות. צורת מימוש זו של המודל התפישתי היא ישירה ופשוטה מאוד להבנה גם על ידי משתמשים שאינם מקצועני מחשב. המודל מבוסס על שפת גישה מיוחדת בשם SQL, אשר שונה באופן מהותי מכל שפות הגישה שהמודלים האחרים תומכים בהן. חלק ג' בספר זה מוקדש לסקירת שפת SQL לעומק.

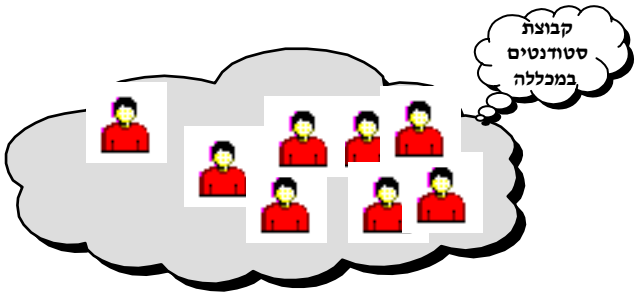
מושגי יסוד בתורת הקבוצות

בסעיף זה נציג מספר מושגי יסוד מתורת הקבוצות הקשורים למודל הנתונים הטבלאי. הצגתם תיעשה בצורה לא פורמלית ותסתייע במספר רב של דוגמאות. קורא שמכיר מושגים אלה יכול לעבור לסעיף הבא.

קבוצה (Set)

קבוצה **Set**	***קבוצה* היא אוסף של אובייקטים.**

ניתן לתאר את הקבוצה בשתי צורות: על ידי **הצגת כל האובייקטים** שלה, או על ידי **הגדרת כללים** ההופכים אובייקט לחבר בקבוצה. נסמן את הקבוצה באותיות גדולות (A, B, ...) ואת האובייקטים באותיות קטנות (a, b, ...) בהתאמה.



תרשים 4.5: ייצוג גרפי של קבוצה.

ניתן להגדיר את הקבוצה גם על ידי ביטוי המציג את כל חברי הקבוצה:

A = סטודנטים במכללה = {משה, אייל, רן, דינה, איילת, ציון, ...}

תיאור כל האובייקטים של הקבוצה אפשרי בקבוצות קטנות בלבד, ולכן ניתן לתאר קבוצה על ידי כלל המגדיר מי הם חברי הקבוצה.

A = סטודנטים במכללה = { a | a הוא סטודנט במכללה}

משמעות הביטוי היא שהקבוצה A, קבוצת סטודנטים במכללה, מכילה את אובייקט a המקיים את התנאי ש - a הוא סטודנט במכללה.

קבוצה יכולה להיות סופית או אינסופית. **קבוצה סופית** מכילה **בדיוק** n אובייקטים שונים (n הוא מספר שלם חיובי). אם הקבוצה אינה ממלאת תנאי זה, אזי היא **קבוצה אינסופית**. למשל, הקבוצה סטודנטים במכללה היא קבוצה סופית, אולם הקבוצה מספרים חיוביים זוגיים היא קבוצה אינסופית.

הקבוצה מאופיינת בכך שכל אובייקט מופיע בה פעם אחת בלבד, וסדר האובייקטים בה חסר משמעות. גודל הקבוצה נקבע על ידי מספר האובייקטים החברים בקבוצה.

העובדה שאובייקט מסוים **הינו חבר בקבוצה** תסומן כך:

$$a \in A$$

העובדה שאובייקט מסוים **אינו חבר בקבוצה** תסומן כך:

$$b \notin A$$

לדוגמה, נוכל לרשום:

{סטודנטים במכללה} ∈ אייל **{סטודנטים במכללה} ∉ יצחק**

קבוצה ריקה (Null Set) אינה מכילה אובייקטים. למשל, הקבוצה סטודנטים לספרות סינית תהיה ריקה, אם המחלקה לספרות סינית נמצאת בשלבי הקמה וטרם קלטה סטודנטים. במשך הזמן יצטרפו אליה אובייקטים (סטודנטים) והקבוצה תפסיק להיות ריקה.

ניתן להגדיר **תת-קבוצה** (Subset), שבה כל אובייקט החבר בתת-הקבוצה הינו חבר גם בקבוצה המלאה אבל לא להפך. דוגמה:

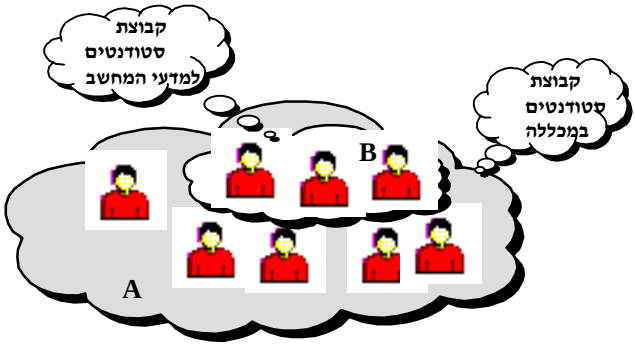
A = {סטודנטים במכללה} = {a | a הוא סטודנט במכללה}

B = {סטודנטים למדעי המחשב} = {b | b סטודנט במחלקה למדעי המחשב}

קבוצה B הינה תת-קבוצה לקבוצה A, מכיון שלא כל הסטודנטים במכללה לומדים מדעי המחשב, אבל ההיפך אינו נכון. מקובל לסמן עובדה זו בצורה זו:

$$B \subset A$$

באופן גרפי נוכל לתאר תת-קבוצה בדרך זו:



תרשים 4.6: B היא תת-קבוצה של A.

קבוצות זרות הן קבוצות נפרדות, שאינן חופפות. הגדרה של קבוצות נפרדות אינה מונעת מצב בעתיד, שבו הן תתלכדנה או תחפופנה זו את זו באופן חלקי או מלא.

לדוגמה, נגדיר שתי קבוצות של סטודנטים באותה שנה במחלקה למתמטיקה: קבוצה הלומדת אלגברה לינארית וקבוצה הלומדת תורת הקבוצות. כלומר, לפנינו קבוצה אחת (סטודנטים הלומדים באותה שנה במחלקה למתימטיקה) בת שתי תת-קבוצות. נניח, שתת-הקבוצה שלומדת אלגברה לינארית מתבטלת, וכל הסטודנטים ילמדו תורת הקבוצות בלבד. במצב החדש יש התלכדות של שתי תת-הקבוצות. אפשרות אחרת היא שהסטודנטים יחליטו להשתתף הן בלימודי תורת הקבוצות והן בלימודי אלגברה לינארית. תלמידים אלה ישתתפו בשתי תת-קבוצות שמלכתחילה היו **זרות** זו לזו, כלומר **ללא כל חפיפה**.

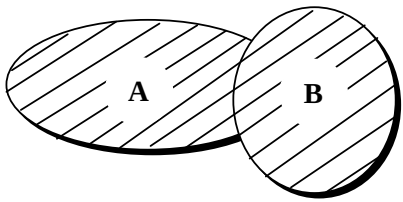
כדוגמה נוספת נגדיר שתי קבוצות של פריטים: **פריטים מייצור עצמי ומשמשים למכירה בלבד** ו**פריטים נרכשים** כחומרי גלם מיצרנים אחרים ומשמשים לייצור בלבד. לכאורה יש כאן מצב ברור של שתי תת-קבוצות זרות, אך עם זאת ייתכן שבעתיד לקוח כלשהו ירצה לקנות מהמפעל חומר גלם כלשהו. במקרה זה, פריטים מתת-הקבוצה של חומרי גלם יימצאו גם בתת-הקבוצה של פריטים למכירה.

פעולות בקבוצות

ניתן לבטא את יחסי הגומלין בין קבוצות על ידי מספר פעולות עיקריות, כפי שנפרט בהמשך.

איחוד **Union**	**איחוד שתי קבוצות A ו-B, שיסומן $A \cup B$, יוצר קבוצה חדשה שמכילה את כל האובייקטים החברים בקבוצה A <u>או</u> חברים בקבוצה B.** $A \cup B = \{x \mid x \in A$ **או** $x \in B\}$

ובאופן גרפי:



תרשים 4.7: קבוצת האיחוד.

לדוגמה, נגדיר שתי קבוצות של סטודנטים:

A = סטודנטים במחלקה למדעי המחשב = { אייל, דינה}

B = סטודנטים במחלקה למנהל עסקים = {רן, איילת, דינה}

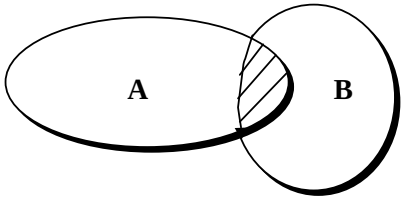
האיחוד בין שתי הקבוצות יהיה:

$A \cup B$ = סטודנטים למדעי המחשב ולמנהל עסקים = {אייל, רן, דינה, איילת}

נשים לב, שדינה הלומדת בשתי המחלקות, מופיעה פעם אחת בלבד בקבוצת האיחוד.

חיתוך **Intersection**	**חיתוך שתי קבוצות A ו-B, שיסומן $A \cap B$, יוצר קבוצה חדשה שמכילה את כל האובייקטים השייכים לקבוצה A <u>וגם</u> לקבוצה B.** $A \cap B = \{x \mid x \in A$ **וגם** $x \in B\}$

ובאופן גרפי:



תרשים 4.8: קבוצת החיתוך.

אם נשתמש בדוגמה הקודמת, קבוצת החיתוך תכיל אובייקט אחד, את הסטודנטית ששמה דינה, המופיעה גם בקבוצה A וגם בקבוצה B.

$A \cap B$ = סטודנטים למדעי המחשב וגם למנהל עסקים = {דינה}

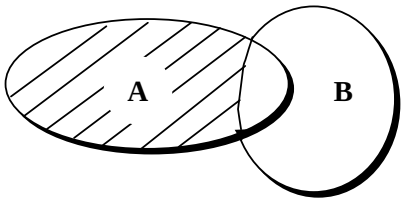
אם $A \cap B = 0$, החיתוך יוצר קבוצה ריקה, כלומר שתי הקבוצות הן קבוצות זרות (Disjoint).

הפרש **Difference**	**ההפרש בין קבוצה A ו-B, שיסומן A - B, יוצר קבוצה חדשה שמכילה את האובייקטים השייכים לקבוצה A <u>ואינם</u> שייכים לקבוצה B.** $A - B = \{x \mid x \in A \text{ ו } x \notin B\}$

לדוגמה, נגדיר קבוצת הפרש בין הסטודנטים במחלקה למנהל עסקים לבין הסטודנטים במחלקה למדעי המחשב. כלומר, קבוצה שתכיל את הסטודנטים במחלקה מנהל עסקים בלבד ושאינם לומדים גם במחלקה מדעי המחשב. לפי הדוגמה הקודמת, יהיו אלה הסטודנטים רן ואיילת.

A - B = סטודנטים למנהל עסקים בלבד = {רן, איילת}

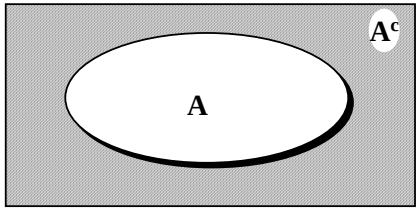
ובאופן גרפי:



תרשים 4.9: קבוצת ההפרש.

משלים **Complement**	**המשלים של קבוצה A, שיסומן A^c, יוצר קבוצה חדשה שמכילה אובייקטים שאינם שייכים לקבוצה A. יש להגדיר קבוצה אוניברסלית U המכילה את <u>כל</u> האובייקטים האפשריים בהקשר הנדון.** $A^c = \{x \mid x \in U \text{ ו } x \notin A\}$

ובאופן גרפי:



תרשים 4.10: המשלים לקבוצה A.

קבוצה U בדוגמה זו היא קבוצת כל הסטודנטים במכללה. קבוצת המשלים של הסטודנטים למדעי המחשב היא כל הסטודנטים שאינם לומדים במחלקה זו.

A^c = סטודנטים שאינם לומדים מדעי המחשב = {רן, איילת}

נגדיר עכשיו את מושג המכפלה. לשם כך נגדיר תחילה את המושג **זוג סדור** (Ordered Pair) המורכב משני אובייקטים, a ו-b. כאשר a הוא האובייקט הראשון ו-b הוא האובייקט השני, נסמן את הזוג (a,b). אפשר להגדיר סדר כלשהו של אובייקטים. למשל רביעייה סדורה תירשם כך: (a,b,c,d).

מכפלה קרטזית **Cartesian Product Set**	**נניח ש A ו-B הן שתי קבוצות. קבוצת המכפלה, או המכפלה הקרטזית (Cartesian Product) של A ו-B, שתסומן A ⊗ B, מכילה את קבוצת כל הזוגות הסדורים (a,b), כך שהאובייקט הראשון בא מקבוצה A והאובייקט השני בא מקבוצה B.** **$A \otimes B = \{(a,b) \mid a \in A$ ו $b \in B\}$**

נגדיר את קבוצת המכפלה של קבוצת הסטודנטים במחלקה מדעי המחשב וקבוצת הסטודנטים במחלקה מנהל עסקים. המכפלה הקרטזית בין שתי קבוצות אלו תגדיר את **כל** האפשרויות של זוגות סטודנטים בשתי המחלקות.

A ⊗ B = {(אייל, רן), (אייל, איילת), (אייל, דינה), (דינה, רן), (דינה, איילת), (דינה, דינה)}

ניתן להגדיר מכפלה קרטזית של קבוצה עם עצמה. אם קבוצה A מורכבת מאלמנטים [x,y], הקבוצה A ⊗ A תהיה:

$$A \otimes A = \{(y,y),(y,x),(x,y),(x,x)\}$$

ניתן להגדיר את פעולת המכפלה על מספר כלשהו של קבוצות. לדוגמה, המכפלה הקרטזית של קבוצות - A, B ו - C שתסומן A ⊗ B ⊗ C, מורכבת מכל השלשות הסדורות, כך ש-:

$$A \otimes B \otimes C = \{(a,b,c) \mid a \in A,\ b \in B,\ c \in C\}$$

לדוגמה נתונות קבוצות:

$$A = \{a,b,c\} \quad B = \{x,y\} \quad C = \{z,k\}$$

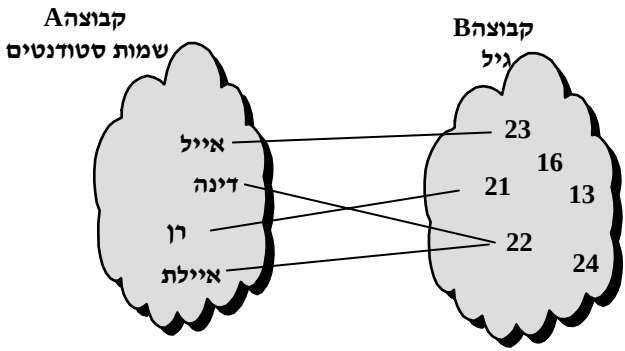
המכפלה הקרטזית שלהן תהיה:

$$\mathbf{A \otimes B \otimes C = \{(a, x, z), (b, x, z), (c, x, z), (a, y, z), (b, y, z), (c, y, z), (a, x, k), (b, x, k), (c, x, k), (a, y, k), (b, y, k), (c, y, k)\}}$$

פונקציות (Functions)

פונקציה היא מקרה מיוחד של קשר בין שתי קבוצות. נניח, שלכל אובייקט בקבוצה A קושרים בדיוק אובייקט אחד מתוך קבוצה B. שני אובייקטים מקבוצה A יכולים להיות קשורים לאובייקט אחד מקבוצה B, וייתכנו אובייקטים ללא כל קשרים שהם. אוסף כל הקשרים האלה הוא **פונקציה מ-A לתוך B**.

פונקציה **Function**	***פונקציה*** **היא תת-קבוצה של המכפלה הקרטזית** $A \otimes B$**, שבה כל אובייקט מתוך** A **מופיע פעם אחת בלבד. הפונקציה תסומן:** $\mathbf{f:A \rightarrow B}$



תרשים 4.11: פונקציה בין שם הסטודנט וגילו.

בדוגמה המוצגת בתרשים זה נוכל לומר, שקיימת פונקציה בין קבוצה A לבין קבוצה B, כי כל אובייקט בקבוצה קשור בדיוק לאובייקט אחד בקבוצה B אבל לא להיפך. כלומר, לא ניתן להגדיר פונקציה מקבוצה B לקבוצה A. הפונקציה מגדירה את **הקשר**, או את **המיפוי** (Mapping), בין אובייקטים בשתי קבוצות או באותה קבוצה.

כפי שראינו בפרק הקודם במסגרת הדיון על פונקציונליות הקשרים, ניתן להגדיר מספר סוגים של קשרים: קשר חד-חד-ערכי, קשר חד-רב-ערכי וקשר רב-רב-ערכי.

קשר חד-חד-ערכי (1:1) הוא הקשר הפשוט ביותר הקיים בין האובייקטים. לכל אובייקט בקבוצה אחת יש התאמה לאובייקט אחד בקבוצה השנייה ולהיפך. בצורה פורמלית נאמר שקיים קשר חד-חד-ערכי, אם קיימות שתי קבוצות A ו-B שניתן להגדיר עבורן שתי פונקציות $f:A\rightarrow B$ ו- $f:B\rightarrow A$. כזכור, הפונקציה מאפשרת לאובייקט הראשון להופיע פעם אחת בלבד.

קשר חד-רב-ערכי (1:N) מאפשר שתהיה לכל אובייקט בקבוצה אחת התאמה למספר אובייקטים בקבוצה השנייה, אבל לא להיפך. כלומר, בין אובייקטים בקבוצה השנייה לבין אובייקטים בקבוצה הראשונה קיימת התאמה חד-ערכית. בצורה פורמלית נאמר שקיים קשר חד-רב-ערכי אם קיימות שתי קבוצות A ו-B וניתן להגדיר עבורן רק פונקציה אחת, $f:A\rightarrow B$ או $f:B\rightarrow A$.

קשר רב-רב-ערכי (N:M) מציין שלכל אובייקט בקבוצה אחת יש התאמה למספר אובייקטים בקבוצה השנייה ולהיפך. בצורה פורמלית נאמר שקיים קשר רב-רב-ערכי, אם קיימות שתי קבוצות A ו-B ולא ניתן להגדיר אף אחת מהפונקציות $f:A\rightarrow B$ או $f:B\rightarrow A$.

המודל הטבלאי

המודל הטבלאי הוא מודל פורמלי לייצוג קבוצות ישות והקשרים ביניהן במערכות RDBMS. המודל ההיררכי והמודל הרשתי השתמשו למטרה זו בשתי אבני בניין, הרשומה והקשר. **הרשומה** מייצגת את הישות **והקשר** מייצג את הקשר שבין מופעים שונים של ישויות בקבוצות השונות (או באותה קבוצה במקרה של קשר רפלקסיבי).

המודל הטבלאי משתמש לאותה מטרה רק במבנה אחד, **היחס** (Relation) או **הטבלה** (Table). זוהי תרומה ראשונה בנושא פשטות מבנה הנתונים והבנתו.

יחס מתמטי (Mathematical Relation)

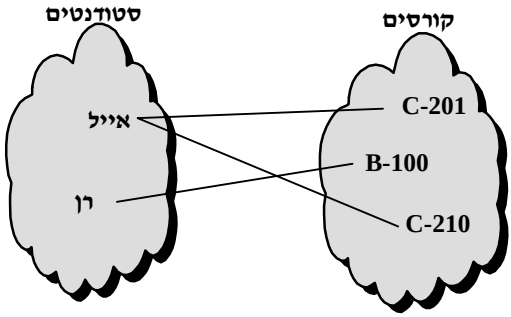
היחס הינו אחד המושגים החשובים במודל הנתונים הטבלאי. הוא מתאר קשר בין אובייקטים של אותה קבוצה, או בין אובייקטים של קבוצות שונות.

נתחיל מהגדרה פשוטה של יחס בינארי ונעבור לאחר מכן להגדרת יחס במימד n כלשהו.

יחס בינארי **Binary Relation**	**היחס הבינארי הוא תת-קבוצה של קבוצת המכפלה הקרטזית בין שתי קבוצות D_1 ו-D_2. הביטוי המתמטי שלו יהיה:** $R(D_1, D_2) \subseteq D_1 \otimes D_2$

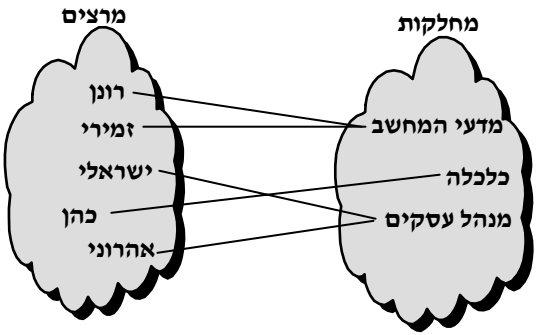
סדר הופעת האובייקטים ביחס בינארי הוא חשוב. למשל, בנקודת זמן כלשהי רק חלק מבין כל הזוגות האפשריים (סטודנט, קורס) מייצגים קשר קיים בין סטודנטים ל-קורסים.

בדוגמה שבתרשים 4.12, האובייקט הראשון יהיה תמיד מתוך קבוצת סטודנטים והאובייקט השני יהיה תמיד מתוך קבוצת קורסים. בדוגמה זו לא קיימים הקשרים "אייל - B-100" ו"רן - C-201" ו"רן - C-210".



תרשים 4.12: ביטוי גרפי ליחס בינארי.

נציג דוגמה נוספת ליחס בינארי ונגדיר לשם כך שתי קבוצות: קבוצה D_1 היא קבוצת מרצים במכללה וקבוצה D_2 היא קבוצת מחלקות במכללה. המכפלה הקרטזית $D_1 \otimes D_2$ מכילה את כל הזוגות הסדורים האפשריים בין מרצה לבין מחלקה, כאשר האובייקט הראשון יהיה מרצה והאובייקט השני יהיה מחלקה. במציאות קיים קשר בין מרצים לבין מחלקות, אך הקשר אינו מקיף את כל האפשרויות מכיון שרק חלק מהקבוצה מרצים מלמדים במחלקה מסוימת. היחס הבינארי הקושר בין שני האובייקטים הינו תת-קבוצה למכפלה הקרטזית.



תרשים 4.13: יחס בין מרצה למחלקה.

נרחיב עכשיו את הגדרת היחס הבינארי ליחס במימד כלשהו.

יחס **Relation**	**נתון אוסף של קבוצות $D_1,D_2...D_n$, לא בהכרח זרות. היחס הוא אוסף מסודר של איברים $\{d_1,d_2,...d_n\}$, כך שכל איבר d_j מתאים לקבוצה D_j, עבור $j=1,2,...n$. היחס יסומן על ידי $R(D_1,D_2,... D_n)$, כאשר R הוא שם היחס. באופן פורמלי נרשום:** $$R(D_1, D_2, ..D_n) \subseteq D_1 \otimes D_2 \otimes ... \otimes D_n$$

יחס במימד n הינו תת-קבוצה למכפלה הקרטזית של n קבוצות.

נתבונן ביחס במימד 3, המתאר מספר תכונות של נתוני סטודנט. היחס יירשם כך:

סטודנטים במכללה = (מספר סטודנט, שם סטודנט, עיר מגורים)

כל אחת מהתכונות הינה קבוצה, כמו קבוצת מספרי הסטודנטים, קבוצת שמות הסטודנטים וקבוצת ערי מגורים. המכפלה הקרטזית של שלוש הקבוצות תהיה קבוצה גדולה מאוד, כי היא מייצגת את כל האפשרויות בהתאם למספר האובייקטים בכל אחת מהקבוצות. יחס במימד 3 הוא קבוצה חלקית למכפלה הקרטזית והוא יכיל אובייקטים כמספר הסטודנטים במכללה.

נוח מאוד לייצג את היחס המתמטי בצורת **טבלה**. למען הנוחות רשמנו את שמות העמודות בעברית ובאנגלית.

סטודנטים

STUDENT_ID	NAME	CITY
מס. סטודנט	שם	עיר
105	Moshe	Haifa
210	Dan	Tel Aviv
107	David	Tel Aviv
110	Ran	Haifa
245	Yair	Jerusalem
240	Lea	Haifa
310	Tova	Tel Aviv

תרשים 4.14: ייצוג יחס במימד 3 על ידי טבלה.

כלומר, ניתן לתאר את היחס כאוסף של **שורות ועמודות**, או בפשטות – **טבלה**. כל עמודה מייצגת תכונה כלשהי של הישות וכל שורה מייצגת ישות אחת בטבלה. הערך d_{ij}, המופיע בשורה i ועמודה j נלקח מתוך הקבוצה D_j המייצגת את **מרחב הערכים** (Domain) האפשרי של התכונה. בהקבלה למונחים מקובלים במערכות לניהול קבצים, עמודה מתייחסת ל**שדה**, ושורה מתייחסת ל**רשומה** וטבלה ל**קובץ**.

כפי שנאמר במבוא, אחת התרומות החשובות של Codd היתה באבחנה שקיים דמיון רב (אם כי לא זהות מלאה) בין **טבלה** לבין **היחס המתמטי**. מכיון שהערכים המופיעים בעמודות נלקחים מתוך מרחב הערכים האפשרי של התכונה, הרי שנוכל להגדיר קבוצה דמיונית הנוצרת בעקבות המכפלה הקרטזית של מרחבי הערכים השונים. מתוך קבוצה דמיונית זו נשלוף שורות מסוימות בלבד. ל**קבוצה** המתקבלת נקרא בשם **יחס** (Relation).

את היחס נוכל לממש במודל הנתונים בתור טבלה דו-מימדית ונוכל להתייחס אל כל טבלה כאל מקרה מיוחד של היחס המתמטי ובעקבות כך – להפעיל עליה אופרטורים מוכרים מעולם תורת הקבוצות.

נגדיר מהן התכונות הנדרשות מטבלה כדי שנוכל לראות אותה כמו **יחס**:

- כל עמודה מקבלת את ערכיה ממרחב ערכים אחד בלבד.
- סדר העמודות בטבלה אינו חשוב, כלומר תוכנית היישום אינה יודעת מהו סדר העמודות בטבלה ועל כן, כל שינוי בסדר זה לא ישפיע עליה.
- סדר השורות בטבלה אינו חשוב, כלומר תוכנית היישום אינה יודעת את סדר השורות בטבלה ועל כן, כל שינוי בסדר המיון לא ישפיע עליה.
- אין שתי שורות זהות בטבלה.
- כל ערך Dij בטבלה הינו ערך פשוט (חד-ערכי), כלומר ללא מופעים שחוזרים על עצמם.

להלן טבלת התאמה בין המונחים השונים, כדי למנוע אי הבנה:

מונח פורמלי	מונח מקובל במודל הטבלאי	מונח קודם ממבנה קבצים
יחס (Relation)	טבלה (Table)	קובץ (File)
שורה (Tuple)	שורה (Row)	רשומה (Record)
תכונה (Attribute)	עמודה (Column)	שדה (Field)
מרחב ערכים (Domain)	מרחב ערכים (Domain)	

ראינו עד עתה, שהיחס ניתן למימוש כטבלה מנקודת מבט המשתמש. **טבלה** היא מקרה מיוחד של המבנה המתמטי **יחס**, שיש לו הגדרה מדויקת, שלא כמו הגדרת הטבלה.

המודל הטבלאי מבוסס על העיקרון שתחת אילוצים מסוימים ניתן לראות את הטבלה כמו יחס וכתוצאה – להשתמש בתורת הקבוצות והיחסים לטיפול במבנה זה.

סכימה של טבלה (Relation Schema)

כל טבלה במודל הטבלאי ניתנת לתיאור על ידי סכימה. סכימה זו מגדירה את שם הטבלה, את שמות העמודות, את מרחבי הערכים של כל עמודה ואת המפתח העיקרי של הטבלה.

סכימה טבלאית **Relation Schema**	**סכימה של טבלה מגדירה את מבנה הטבלה הבודדת. נניח ש-R הוא שם הטבלה, Ai הוא השם של עמודה i ו-Di הוא מרחב הערכים של עמודה i. הסכימה של הטבלה R תהיה:** **$R\ (A_1{:}D_1,\ A_2{:}D_2,\ ..A_n{:}D_n)$, כך שמתקיים:** **$d_1 \in D_1,\ d_2 \in D_2\ ..d_n \in D_n$**

בדרך כלל מקובל לא לרשום את מרחב הערכים, ולכן נקבל:

$$R\ (A_1,\ A_2,\ ..A_n)$$

לדוגמה, נרשום את הסכימה של טבלת הקורסים:

קורסים (מספר קורס, שם קורס, מספר נקודות זכות)

מרחבי הערכים לא חייבים להיות זרים, ופירוש הדבר שעמודות שונות יכולות לקבל את הערכים שלהן מאותו מרחב ערכים. במקרה כזה נאמר שלאותו מרחב יש **תפקידים** (Roles) שונים בתוך אותה טבלה. מספר העמודות בסכימה של הטבלה נקראת **הדרגה של הטבלה** (Relation Degree).

נרשום את הסכימה של טבלת הקורסים ונשתמש בשמות לועזיים (כמקובל בתכנות). להפרדה של שמות עמודות בלועזית המורכבים ממילים אחדות נשתמש במקף תחתי (_) ולא במקף רגיל (-), כדי להבחין בין שמות עמודות, או טבלאות, לבין פעולת המינוס האלגברי.

Courses (Course_Id, Name, Points)

נרשום את הסכימה של הטבלה באופן מלא, ובכלל זה מרחב הערכים (טיפוס הנתונים).

Courses (Course_Id:String(7), Name:String(30), Points:Integer)

שיטת רישום זו אפשרית, אבל פחות נפוצה. בדרך כלל מקובל לרשום את הסכימה עם שמות העמודות בלבד.

מופע (שורה) **Relation Instance**	**מופע של סכימה R, שיסומן ב- r(R), הוא אוסף מסודר של ערכים <v1, v2, ... vn> שבו כל ערך v_i הינו ערך אחד מתוך מרחב הערכים D_i.**

כדי שנוכל לזהות כל שורה בטבלה באופן חד-משמעי, חייב להתקיים מצב בו הערך, שנמצא בעמודה אחת או יותר באותה שורה, הוא חד-ערכי לעומת כל השורות האחרות. לעמודה (או העמודות) זאת נקרא בשם **מפתח** (Key).

מפתח **Relation Key**	**נסמן ב- ti שורה i בטבלה R. אוסף עמודות K בטבלה R הוא מפתח אם מתקיים האילוץ ti[k] ≠ tj[k]. כלומר, הערך בעמודות K בשתי שורות i ו- j, כאשר i ≠ j, מכיל ערכים שונים.**

עמודת המפתח מקבלת **ערכים שונים** בכל אחת משורות הטבלה, והמשמעות היא שאין שתי שורות המכילות את אותו הערך בעמודת המפתח. ניתן לומר שהמיפוי בין עמודת המפתח לבין מרחב הערכים הוא חד-חד-ערכי. המפתח חייב לקיים את שני התנאים שהוסברו בפרק 3 שהם: זיהוי חד-ערכי ואי-כפילות.

מקובל לומר שכל אוסף של עמודות המקיים את התכונה הזו נקרא **מפתח-על** (Superkey) של טבלה R. נשים לב לכך שהגדרת מפתח-על אינה דורשת מינימליות, ומפתח-על יכול גם להכיל עמודות שאינן מוסיפות לזיהוי החד-ערכי של הרשומה. כדי להפוך מפתח-על למפתח רגיל, יש לסלק ממנו את כל העמודות המיותרות.

לדוגמה, אוסף העמודות {Course_Id, Name} הוא מפתח-על, אבל אינו מפתח, כי העמודה Name אינה מוסיפה לזיהוי החד-ערכי.

בסכימה של הטבלה נסמן את עמודות המפתח בקו תחתי. ניקח לדוגמה את הטבלה קורסים שבה מספר הקורס יכול לשמש כמפתח.

Courses (Course_Id, Name, Points)

בגלל החשיבות הרבה שיש למושג **מפתח** במודל הטבלאי – אמצעי לזיהוי השורה בטבלה – נחזור ונגדיר בהמשך את סוגי המפתחות השונים.

סכימה טבלאית (Relational Schema)

בסיס נתונים טבלאי מורכב ממספר כלשהו של טבלאות דו-מימדיות פשוטות. לכל טבלה יש סכימה נפרדת.

סכימה טבלאית **Relational Schema**	***סכימה טבלאית*** **מוגדרת כאוסף כל הסכימות של הטבלאות המרכיבות את בסיס הנתונים.**

לכל טבלה בסכימה חייב להיות **שם חד-ערכי** (Unique Name), כך שניתן יהיה לזהותה באופן חד-משמעי. לכל עמודה בטבלה חייב להיות שם חד-ערכי כדי שאפשר יהיה לזהות אותה באופן חד-משמעי. שם עמודה יכול לחזור על עצמו בטבלאות שונות, אולם לא באותה טבלה.

לדוגמה, נתון בסיס נתונים המורכב משלוש טבלאות: קורסים, סטודנטים ו-ציונים. להלן הסכימה הטבלאית של בסיס הנתונים:

קורסים (מספר קורס, שם קורס, מספר נקודות זכות)
סטודנטים (מספר סטודנט, שם סטודנט, עיר מגורים)
ציונים (מספר קורס, מספר סטודנט, סמסטר, מועד בחינה, ציון)

נרשום את הסכימה הטבלאית עם שמות הטבלאות והעמודות בלועזית.

Courses (Course_Id, Name, Points)

Students (Student_Id, Name, City)

Grades (Course_Id, Student_Id, Semester, Term, Grade)

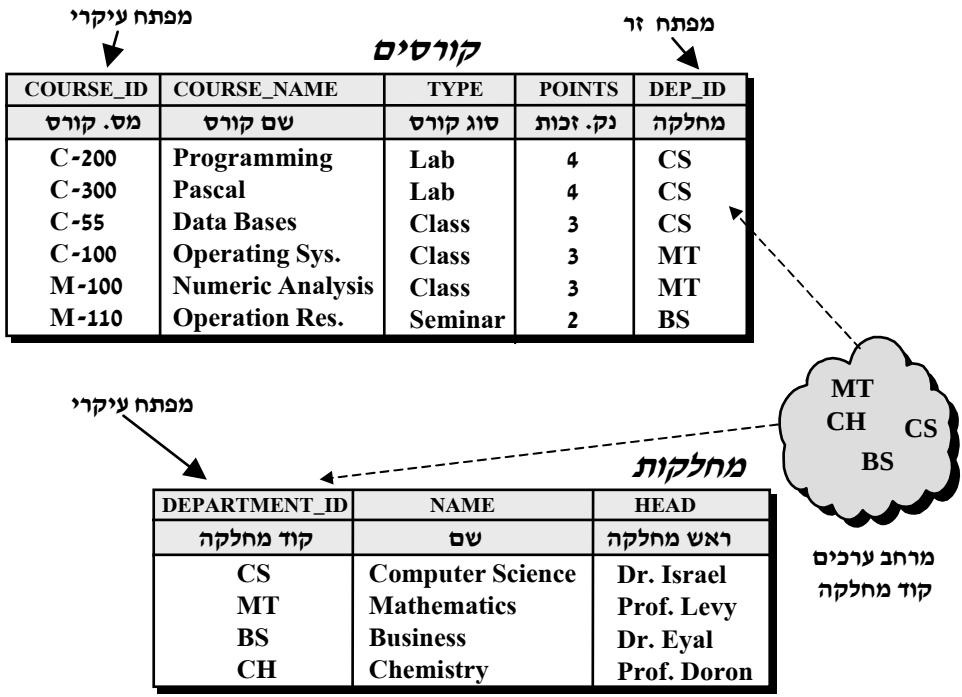
כפי שניתן לראות, אותו שם של עמודה חוזר על עצמו בטבלאות שונות. כדי לפנות לעמודה מספר קורס בטבלה ציונים נרשום: Grades.Course_Id.

ייצוג קשרים במודל הטבלאי

בסיס הנתונים מורכב מאוסף של טבלאות והקשרים ביניהן. במודל ההיררכי והרשתי, הקשרים ממומשים על ידי שימוש במצביעים ליצירת מבני נתונים מורכבים ומקושרים. לעומתם, המודל הטבלאי אינו מאפשר שימוש במצביעים ברמה הלוגית. כיצד אם כן, נממש את הקשרים במודל הטבלאי?

קשר בין שתי טבלאות נוצר ברגע ששתי עמודות, המופיעות בשתי טבלאות שונות, מקבלות את הערכים שלהן מאותו מרחב ערכים. נתבונן בדוגמה שבתרשים 4.15 שבו מוצג קשר ביו הטבלה מחלקות לבין הטבלה קורסים.

הקשר נוצר על ידי **ערכים** זהים המופיעים בעמודות שונות בשתי הטבלאות. לדוגמה, בעמודה קוד מחלקה בטבלת מחלקות מופיע הקוד CS ובעמודה קוד מחלקה בטבלת קורסים מופיע גם כן הערך CS. מכיון ששני ערכים אלה באים מאותו מרחב ערכים, המשמעות היא שאלה קורסים המוצעים על ידי המחלקה מדעי המחשב. לעמודה קוד מחלקה בטבלה קורסים נקרא בשם **מפתח זר**.



תרשים 4.15: קשר בין שתי טבלאות במודל הטבלאי.

מפתח זר **Foreign Key**	**נסמן ב- t_i שורה i בטבלה R וב- s_i נסמן שורה j בטבלה P. אוסף עמודות FK בטבלה R יקרא *מפתח זר*, אם:** **מרחב הערכים של FK זהה למרחב הערכים PK המהווה מפתח עיקרי בטבלה P.** **מתקיים $t_i[FK] \in s_j[PK]$, כלומר הערך בעמודות FK בטבלה R בשורה i כלשהי קיים בעמודות PK בטבלה P בשורה j כלשהי.**

מפתח זר יכול לפנות גם אל מפתח עיקרי של אותה טבלה, כלומר יכול להתקיים R = P. המודל הטבלאי מאפשר קישור נוח בין הטבלאות השונות, **ואין צורך בהגדרה מראש של הקשרים**. כשמוסיפים טבלה חדשה לבסיס הנתונים, היא נקשרת באופן לוגי לכל הטבלאות שיש להן עמודות הנובעות ממרחב ערכים זהה. אין צורך להפעיל תוכניות שירות מיוחדות לבניית המצביעים לכל הטבלאות הרלוונטיות. מבחינה מעשית, הקישור החדש אינו דורש ביצוע פעולה כלשהי, כי הוא יבוא לידי ביטוי רק בעת הפעלת האופרטורים של שפת SQL.

אמינות במודל הטבלאי (Relational Integrity)

המודל הטבלאי מבוסס על שני אילוצי אמינות: אילוץ המתייחס ל**מפתח העיקרי** ואילוץ המתייחס ל**מפתחות זרים**.

אמינות הישות (Entity Integrity)

אילוץ זה מתייחס ל**מפתח העיקרי** של כל טבלה.

אמינות המפתח **Entity Integrity**	**נניח ש-K הוא אוסף עמודות המרכיבות את המפתח העיקרי. אף עמודה ב-K אינה יכולה להכיל את הערך Null.**

המפתח העיקרי משמש לזיהוי שורה בטבלה, ולכן הוא לא יכול להכיל ערך Null.

אמינות הקשר (Referential Integrity)

בין הטבלאות מתקיימים קשרים שונים. אמינות הקשר (Referential Integrity), או RI כפי שהיא נקראת בקיצור, מגדירה אילוץ הקובע את צורת ניהול הקשרים האלה.

אמינות הקשר **Referential Integrity**	**אם טבלה מכילה מפתח זר, הערכים המופיעים בעמודה זו חייבים להופיע במפתח העיקרי של טבלה אחרת, או להכיל את הערך Null.**

כפי שראינו, קשר בין שתי טבלאות מבוסס על העובדה ששתי עמודות המופיעות בשתי טבלאות שונות, הן למעשה בעלות אותה משמעות סמנטית, ולכן נובעות מאותו מרחב ערכים. לדוגמה, העמודה קוד מחלקה מופיעה הן בטבלה מחלקות והן בטבלה קורסים. ההבדל בין שני המופעים של העמודה בטבלאות השונות הוא בתפקיד שהעמודה ממלאת בכל טבלה. בטבלה מחלקות העמודה מספר מחלקה משמשת כ**מפתח עיקרי** (Primary Key או PK בקיצור) ואילו בטבלה קורסים העמודה מספר מחלקה מוגדרת כ**מפתח זר** (Foreign Key או FK בקיצור). נשים לב שלמרות ששתי העמודות מקבלות את הערכים מאותו מרחב ערכים, מספרי מחלקות, על המפתח הזר קיים אילוץ נוסף – הערכים שלו יכולים להיות רק כאלה שמופיעים במפתח העיקרי, או להיות Null. דרך נוספת להבהיר זאת היא לומר שמרחב הערכים של המפתח הזר הוא תת-קבוצה של ערכי עמודת המפתח העיקרי. כלומר, כל ערך המופיע במפתח הזר חייב להופיע במפתח העיקרי, אבל לא להיפך.

האילוץ מתייחס לשתי נקודות זמן שונות. על כן עולות שאלות אלו: מה תהיה ההתנהגות בעת הוספת שורה חדשה לטבלת הבן, מה תהיה ההתנהגות בעת ביטול שורה מתוך טבלת האב, ומה תהיה ההתנהגות בעת עדכון שורה בטבלת האב או בטבלת הבן.

- **אילוץ בעת הוספה:** אילוץ זה קובע שמפתח זר יכול לקבל ערך מתוך הערכים הקיימים בטבלת האב.
- **אילוץ בעת ביטול:** אילוץ זה מגדיר כיצד יש להתנהג אם מבטלים שורה בטבלת האב, ומה יש לעשות עם השורות בטבלת הבן. מכיון שקיים קשר דו-סטרי בין שתי הטבלאות, ביטול של שורה בטבלת האב יכולה לגרום את ניתוק הקשר ולמעשה – להפרה של האילוץ.
- **אילוץ בעת עדכון:** עדכון המפתח הזר בטבלת הבן, או עדכון המפתח העיקרי בטבלת האב, יכולה לגרום לניתוק הקשר.

אלגברה טבלאית (Relational Algebra)

האלגברה הטבלאית היא שפה **תיאורטית**, שמטרתה להוכיח כי ניתן לשלוף כל אלמנט של מידע מתוך המודל הטבלאי, תוך שימוש באוסף מסוים של אופרטורים. נדגיש שהבנת האלגברה הטבלאית תורמת להבנת שפת SQL, אותה נציג בהמשך.

כל שפה טבלאית מתאפיינת בכך שהיא מורכבת מאופרטורים הפועלים על טבלאות ויוצרים טבלאות חדשות. זהו הבדל מהותי לעומת שפות לא-טבלאיות, שהן פרוצדורליות במהותן. אופרטור לא-טבלאי (כגון FIND NEXT במודל הרשתי) פועל על סוג רשומה מסוים ומחזיר כתשובה רשומה בודדת (זהו אופרטור מסוג One-Record-at-a-Time). בגלל אופי זה של האופרטורים הלא-טבלאיים, על תוכנית היישום לנווט בבסיס הנתונים מרשומה לרשומה לשליפת הנתונים המבוקשים.

לעומת המודלים האחרים, מארגן המודל הטבלאי את כל הנתונים במבנה טבלאות ומגדיר שפה לטיפול בנתונים טבלאיים (Relational Data Language) המורכבת מאופרטורים, המאפשרים בניית **טבלאות חדשות** מתוך **טבלאות קיימות** (One-Set-at-a-Time). שפות נתונים אלו מוגדרות כשפות לא-פרוצדורליות, מכיון שהן אינן מאפשרות למשתמש להתייחס לאלמנטים הקשורים למיקום התוכנית בבסיס הנתונים (כגון, קרא את הרשומה הבאה, בדוק האם הגעת לסוף וכד').

האלגברה הטבלאית מהווה כלי להוכחת היכולת לשלוף **כל** אלמנט של מידע מבסיס נתונים טבלאי. שפה טבלאית בעלת תכונה כזאת תיקרא שפה שלמה מבחינה טבלאית (Relationally Complete). למרות יכולת קיומה של שפה שלמה מבחינה טבלאית, אין זו שפה אמיתית המשמשת מערכת טבלאית מסחרית כשפת נתונים. חשיבותה של האלגברה הטבלאית היא רק כבסיס להבנת שפות טבלאיות אמיתיות, כדוגמת שפת SQL.

בסעיף הבא נסקור את כל האופרטורים של האלגברה הטבלאית.

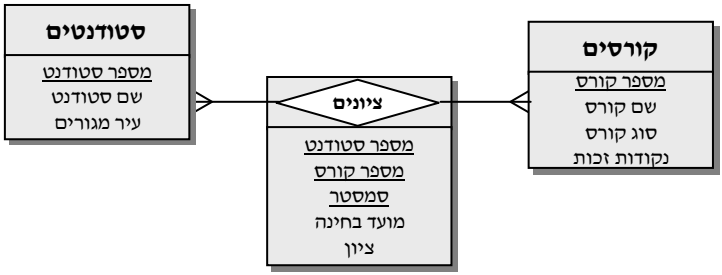
האופרטורים של האלגברה הטבלאית

חלק מהאופרטורים של האלגברה הטבלאית לקוחים מעולם תורת הקבוצות, וכאן באים לידי ביטוי גם השוני של השפה לעומת שפות אחרות וגם העוצמה של השפה.

ניתן לסווג את האופרטורים של האלגברה הטבלאית לשתי קבוצות:

- **אופרטורים אונאריים:** אופרטורים הפועלים על טבלה אחת בלבד. לקטגוריה זו שייכים אופרטור הבחירה (Select) ואופרטור ההיטל (Project).
- **אופרטורים בינאריים:** אופרטורים הפועלים על שתי טבלאות ובונים מהן טבלת תוצאה אחת. לקטגוריה זו שייכים אופרטור הצירוף (Join), אופרטור האיחוד (Union), אופרטור החיתוך (Intersect) ואופרטור החיסור (Minus).

להדגמת האלגברה הטבלאית, נשתמש בבסיס נתונים טבלאי המורכב משלוש טבלאות.



תרשים 4.16: תרשים ישויות-קשרים.

תרשים זה מציג את מימוש המודל הטבלאי על ידי שלוש טבלאות:

Courses (Course_Id, Course_Name, Type, Points)

Students (Student_Id, Name, City)

Grades (Student_Id, Course_Id, Semester, Term, Grade)

תרשים 4.17 מציג מצב נתון של שלוש הטבלאות:

Courses *קורסים*

COURSE_ID	COURSE_NAME	TYPE	POINTS
מס. קורס	שם קורס	סוג קורס	נק. זכות
C-200	Programming	Lab	4
C-300	Pascal	Lab	4
C-55	Data Bases	Class	3
C-100	Operating Sys.	Class	3
M-100	Numeric Analysis	Class	3
M-110	Operation Res.	Seminar	2

Students *סטודנטים*

STUDENT_ID	NAME	CITY
מס. סטודנט	שם	עיר
105	Moshe	Haifa
210	Dan	Tel Aviv
107	David	Tel Aviv
110	Ran	Haifa
245	Yair	Jerusalem
240	Lea	Haifa
310	Tova	Tel Aviv

Grades *ציונים*

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE
מס. סטודנט	מס. קורס	סמסטר	מועד	ציון
105	M-100	SUM1997	A	78
105	C-55	AUT1999	B	85
105	C-100	AUT1999	A	95
210	C-100	AUT1999	A	75
210	C-300	SUM1998	B	82
210	C-200	SUM1998	A	70
245	M-100	SUM1997	A	90
245	C-55	AUT1999	B	80
245	C-300	SUM1998	A	58
310	M-100	SUM1997	A	75

תרשים 4.17: תוכן בסיס נתונים לדוגמה.

בחירת שורות (Select)

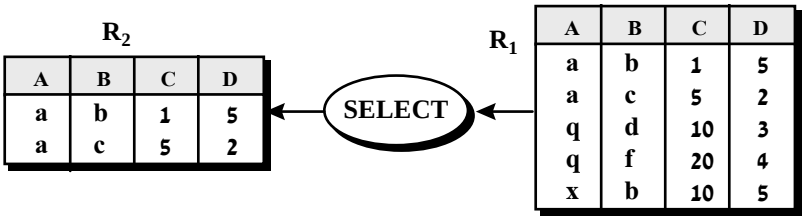
אופרטור הבחירה מאפשר שליפת שורות מסוימות מתוך טבלה ובונה טבלה חדשה, המכילה רק את השורות העונות על תנאי שליפה מוגדר כלשהו.

מבנה הפקודה:

SELECT *Table_Name*
WHERE *Predicate*
GIVING *Result_Table_Name*

דוגמה כללית:

SELECT R_1 **WHERE** A = *'a'* **GIVING** R_2



תרשים 4.18: אופרטור הבחירה.

דוגמה א': הצג את כל הסטודנטים שנבחנו במועד ב' בסמסטר סתיו 1999.

SELECT *GRADES*
WHERE *SEMESTER = "AUT1999'*
AND TERM = 'B'
GIVING *RESULT*

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE
מס. סטודנט	מס. קורס	סמסטר	מועד	ציון
105	C-55	AUT1999	B	85
245	C-55	AUT1999	B	80

דוגמה ב': הצג את כל הסטודנטים אשר קיבלו ציון גבוה מ-70 בקורס M-100.

SELECT *GRADES* **WHERE** *GRADE > 70 AND*
COURSE_ID = 'M-100' **GIVING** *RESULT*

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE
מס. סטודנט	מס. קורס	סמסטר	מועד	ציון
105	M-100	SUM1997	A	78
245	M-100	SUM1997	A	90
310	M-100	SUM1997	A	75

בחירת עמודות – היטל (Project)

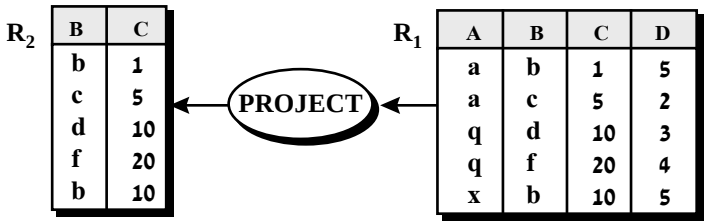
אופרטור ההיטל מאפשר שליפת עמודות מסוימות ובונה טבלה חדשה המכילה רק את העמודות המבוקשות. האופרטור אינו מאפשר קיום שורות כפולות בטבלת התוצאה. סדר העמודות בטבלת התוצאה נקבע על פי סדר הופעת שמות העמודות באופרטור.

מבנה הפקודה:

PROJECT *Table_Name*
OVER *Column_Name_1, Column_Name_2, ...*
GIVING *Result_Table_Name*

דוגמה כללית:

PROJECT R_1 **OVER** (*B,C*) **GIVING** R_2



תרשים 4.19: אופרטור ההיטל.

דוגמה א': הצג את רשימת הקורסים הנלמדים.

PROJECT *COURSES* **OVER** (*NAME*)
GIVING *RESULT*

COURSE_NAME
שם קורס
Programming
Pascal
Data Bases
Operating Sys.
Numeric Analysis
Operation Res.

דוגמה ב': הצג את הסמסטרים בהם נבחנו סטודנטים במועד א' וקיבלו ציון גבוה מ-60.

SELECT *GRADES* **WHERE** *TERM = 'A'* **GIVING** *TEMP*
PROJECT *TEMP* **OVER** (*SEMESTER*) **GIVING** *RESULT*

SEMESTER
סמסטר
SUM1997
AUT1999

זוהי דוגמה להפעלת שני אופרטורים ברצף, זה אחר זה. נשים לב שכל השורות הזהות בוטלו ורק מופע אחד של כל שורה מוצג בטבלה התוצאתית.

צירוף טבלאות (Join)

אופרטור הצירוף בונה טבלה חדשה מתוך שתי טבלאות, על פי עמודה אחת או יותר המשותפות לשתי הטבלאות. זהו האופרטור המאפשר את ניצול הקשרים הטבעיים בין הטבלאות בבסיס הנתונים. כפי שנראה בהמשך, קיימים מספר סוגי צירוף נוספים.

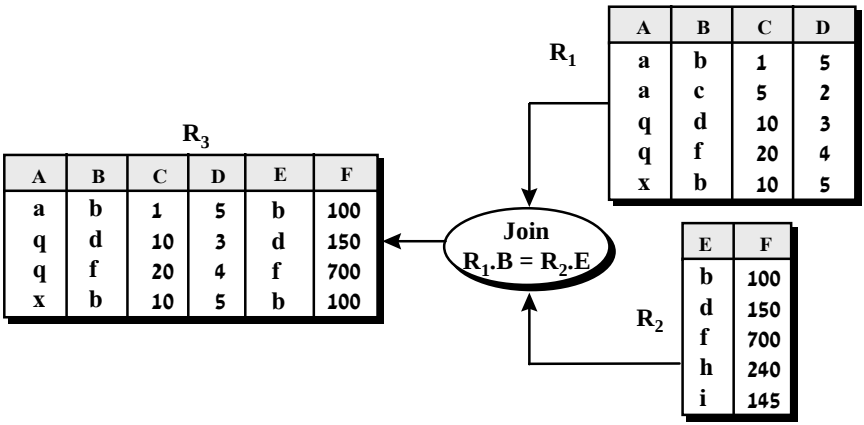
מבנה הפקודה:

JOIN *Table_Name_1, Table_Name_2*
WHERE *Column_Name_1 Condition Column_Name_2*
GIVING *Result_Table_Name*

לצירוף המבוסס על תנאי השוואה כלשהו, כמו <, >, <=, = , NOT או כל תנאי לוגי אחר, מקובל לקרוא בשם Theta Join. למקרה הפרטי, שבו התנאי הלוגי הוא שוויון, מקובל לקרוא בשם Equi Join.

דוגמה כללית:

JOIN R_1, R_2 **WHERE** R_1.B = R_2.E **GIVING** R3

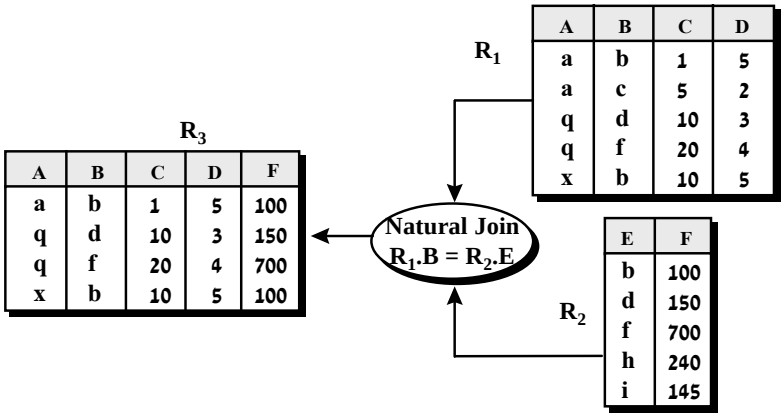


תרשים 4.20: אופרטור הצירוף.

כפי שניתן לראות מהתרשים, זהו צירוף מסוג Equi Join, שבו אופרטור הצירוף מצרף שורות מטבלה R_1 ומטבלה R_2, שבהן מתקיים התנאי:

הערך בעמודה B שווה לערך בעמודה E

בטבלת התוצאה R_3 העמודות B ו-E מכילות ערכים זהים. מכיון שמיותר להציג שתי עמודות המכילות ערכים זהים, מקובל להשמיט אחת מהן. לצירוף שבו התנאי הלוגי הוא שוויון, ורק אחת מהעמודות המשתתפות בתנאי מוצגת בטבלת התוצאה, נקרא בשם Natural Join. מכיון שזהו הצירוף הנפוץ ביותר, מקובל להשתמש באופרטור JOIN במשמעות של "צירוף טבעי".



תרשים 4.21: אופרטור הצירוף הטבעי.

דוגמה א': הצג את שמות הסטודנטים שלמדו בקורס M-100. בדוגמה זו, הנתונים הדרושים מופיעים בשתי טבלאות. רשימת הסטודנטים שלמדו בקורס M-100 מופיעה בטבלה ציונים, ושמות הסטודנטים מופיעים בטבלה סטודנטים. לקבלת המידע המבוקש עלינו לצרף את שתי הטבלאות, בהתבסס על שוויון בערכים המופיעים בעמודה המשותפת מספר סטודנט.

SELECT *GRADES* **WHERE** *GRADES.COURSE_ID = 'M-100'* **GIVING** *TEMP*
JOIN *TEMP, STUDENTS* **WHERE** *GRADES.STUDENTS_ID =*
STUDENTS.STUDENT_ID **GIVING** *TEMP1*
PROJECT *TEMP1* **OVER** (*NAME*) **GIVING** *ANSWER*

NAME
שם
Moshe
Yair
Tova

דוגמה ב': הצג את שמות הסטודנטים ומספר נקודות הזכות שקיבלו כל הסטודנטים שעיר המגורים שלהם "חיפה" ונבחנו במועד "ב'".

SELECT *GRADES* **WHERE** *TERM = 'B'* **GIVING** *TEMP1*
SELECT *STUDENTS* **WHERE** *CITY = 'HAIFA'* **GIVING** *TEMP2*
JOIN *TEMP1, TEMP2* **WHERE** *TEMP1.STUDENTS_ID =*
TEMP2.STUDENT_ID **GIVING** *TEMP3*
JOIN *TEMP3, COURSES* **WHERE** *TEMP3.COURSE_ID =*
COURSES.COURSE_ID **GIVING** *TEMP4*
PROJECT *TEMP4* **OVER** (*COURSE_NAME, POINTS*) **GIVING** *ANSWER*

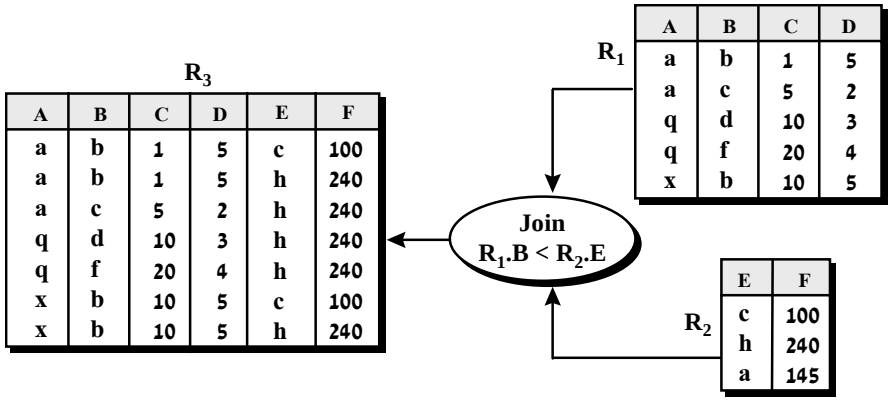
בשלב זה נראה כאילו השפה מסובכת מדי, מפני שלביצוע שליפה פשוטה יחסית נדרש להפעיל מספר רב של אופרטורים בסדר מסוים. אך יש להדגיש שזו הצגה תיאורטית בלבד, ובהמשך נראה שאת כל הפעולות האלו ניתן לבטא במשפט אחד בשפת SQL.

סוגי צירוף נוספים

בנוסף לצירוף הטבעי, קיימים מספר סוגי צירוף נוספים.

צירוף כללי (Theta Join)

בדרך כלל מקובל להשתמש בצירוף המבוסס על תנאי שוויון בין שתי עמודות. יחד עם זאת, אופרטור הצירוף תומך גם בכל תנאי לוגי אחר. לדוגמה, ניתן להשתמש בתנאי לוגי כגון <, >, =<, =>, NOT = וכד'.

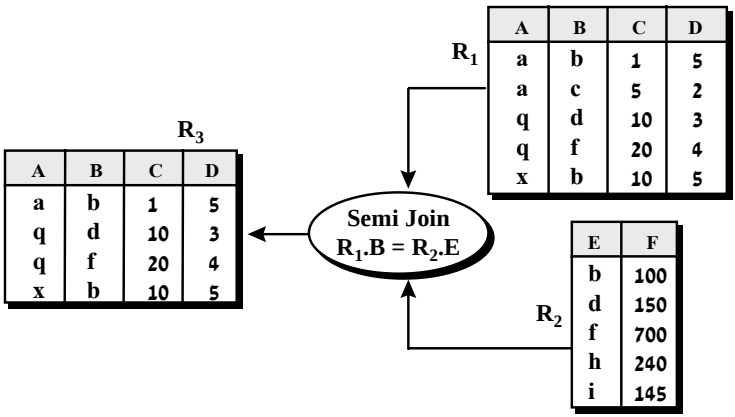


תרשים 4.22: אופרטור הצירוף הכללי.

צירוף למחצה (Semi Join)

על פי צירוף זה, טבלת התוצאה מכילה את העמודות של אחת הטבלאות בלבד. למעשה, ניתן לבצע Semi Join על ידי צירוף רגיל ולאחריו היטל על העמודות של אחת הטבלאות.

SEMI JOIN R_1,R_2 **WHERE** $R_1.B = R_2.E$
GIVING R_3



תרשים 4.23: צירוף למחצה.

דוגמה : הצג את מספר הסטודנט, שם הסטודנט ועיר המגורים לכל הסטודנטים שנבחנו בקורס כלשהו במועד ב'.

SELECT *GRADES* **WHERE** *TERM = 'B'* **GIVING** *TEMP*
SEMI JOIN *STUDENTS, TEMP*
WHERE *STUDENTS.STUDENTS_ID = TEMP.STUDENT_ID*
GIVING *RESULT*

STUDENT_ID	NAME	CITY
מס. סטודנט	שם	עיר
105	Moshe	Haifa
210	Dan	Tel Aviv
245	Yair	Jerusalem

תרשים 4.24: טבלת תוצאה של צירוף למחצה.

כאן מתבצע צירוף רגיל, אולם בטבלת התוצאה מופיעות רק העמודות של הטבלה הראשונה בצירוף. בדוגמה זו, אלו העמודות של הטבלה סטודנטים. ניתן כמובן לממש Semi Join גם על ידי פעולת היטל.

SELECT *GRADES* **WHERE** *TERM = 'B'* **GIVING** *TEMP1*
JOIN *STUDENTS, TEMP1*
WHERE *STUDENTS.STUDENTS_ID = TEMP.STUDENT_ID*
GIVING *TEMP2*
PROJECT *TEMP2* **OVER (***STUDENT_ID, NAME, CITY)*
GIVING *RESULT*

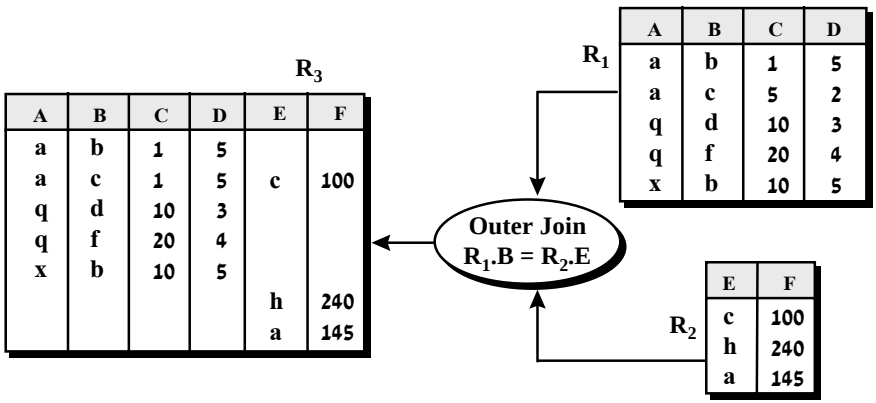
צירוף חיצוני (Outer Join)

תוצאת צירוף זה מכילה את השורות המקיימות את תנאי הצירוף ובנוסף להן – גם את השורות מאחת או משתי הטבלאות אשר **לא** ענו על תנאי הצירוף. בדוגמה זו, אם נרשום:

OUTER JOIN R_1,R_2 **WHERE** R_1.B = R_2.E

GIVING R_3

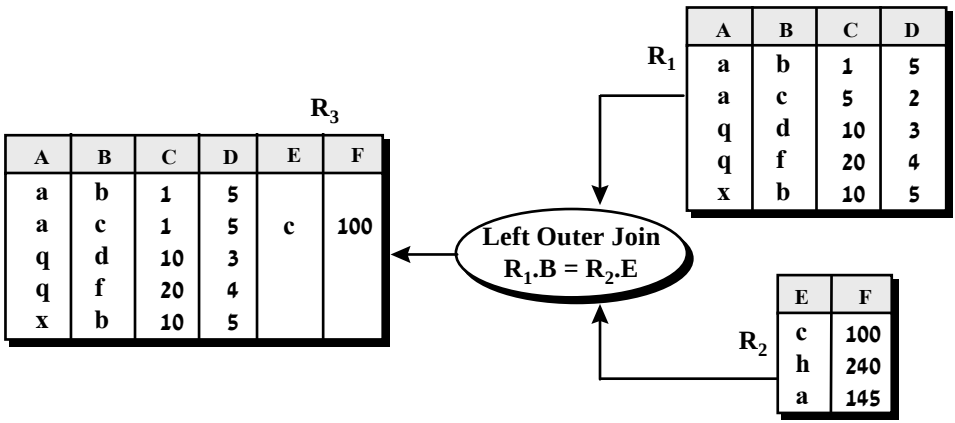
נקבל את הטבלה הבאה :



תרשים 4.25: צירוף חיצוני.

ניתן להגדיר גם צירוף חיצוני, שבו תופענה רק השורות השייכות לאחת משתי הטבלאות המשתתפות בצירוף ואינן מקיימות את תנאי הצירוף. במקרה זה יש לקבוע אם רוצים להציג את השורות שאינן מקיימות את התנאי מהטבלה השמאלית או הימנית.

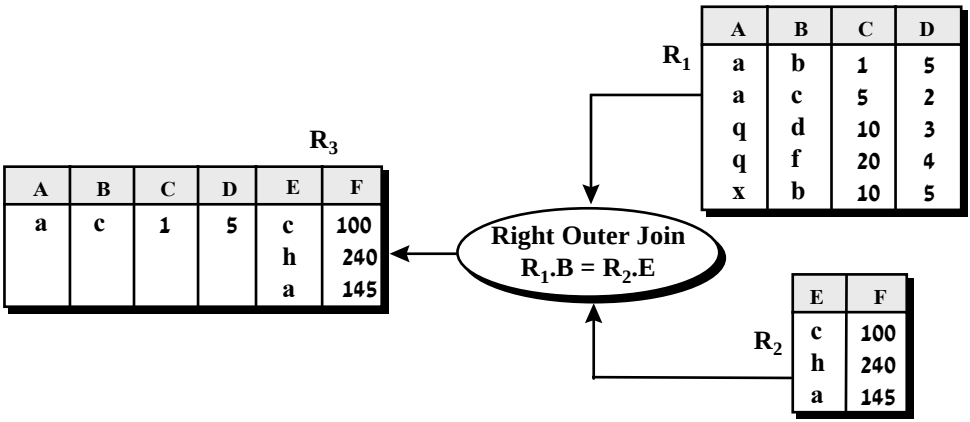
LEFT OUTER JOIN R_1,R_2 **WHERE** $R_1.B = R_2.E$
GIVING R_3



תרשים 4.26: צירוף חיצוני שמאלי.

במקרה זה מופיעות בטבלת התוצאה רק השורות השייכות לטבלה השמאלית המשתתפת בצירוף.

RIGHT OUTER JOIN R_1,R_2 **WHERE** $R_1.B = R_2.E$
GIVING R_3



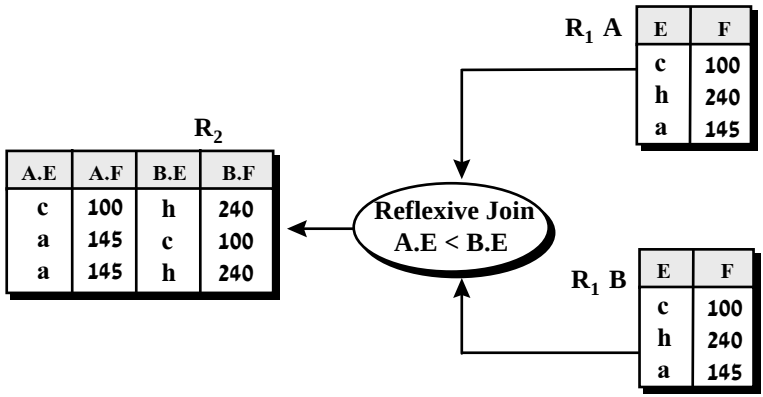
תרשים 4.27: צירוף חיצוני ימני.

במקרה זה מופיעות בטבלת התוצאה רק השורות השייכות לטבלה הימנית המשתתפת בצירוף.

צירוף עצמי (Reflexive Join)

בצירוף זה מצרפים שורות של טבלה מסוימת אל שורות של אותה טבלה, על פי תנאי כלשהו.

JOIN R_1.A, R_1.B **WHERE** A.E = B.E
GIVING R_2



תרשים 4.28: צירוף רפלקסיבי.

מכיון שמצרפים טבלה אל עצמה, יש לתת לכל טבלה כינוי נוסף (Alias), כדי שניתן יהיה לזהות לאיזה עמודה אנו מתכוונים. בדוגמה זו קבענו את השמות A ו-B לשני המופעים השונים של אותה טבלה, טבלת R_1.

איחוד טבלאות (Union)

אופרטור האיחוד מחבר שתי טבלאות ובונה מהן טבלת תוצאה אחת, על פי כללי האיחוד הלקוחים מתורת הקבוצות. השורות בטבלת התוצאה יילקחו משתי הטבלאות, אך שורות זהות תופענה בה פעם אחת בלבד.

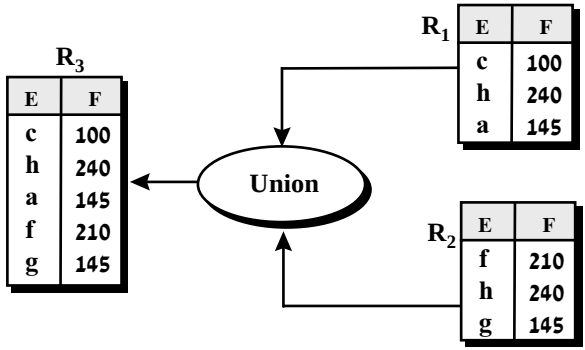
כדי שתהיה משמעות לפעולת האיחוד, אופרטור האיחוד פועל על שתי טבלאות זהות מבחינה סמנטית ומבחינת המבנה שלהן; כלומר, יש להן אותן עמודות הנובעות מאותם מרחבי ערכים. טבלאות המקיימות תנאים אלה נקראות **טבלאות תואמות איחוד** (Union Compatible). אין חשיבות לסדר הופעת הטבלאות באופרטור האיחוד.

מבנה הפקודה:

Table_Name_1 **UNION** *Table_Name_2*
GIVING *Result_Table_Name*

דוגמה כללית:

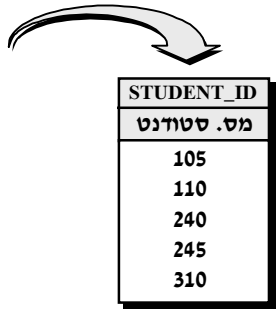
R_1 **UNION** R_2 **GIVING** R_3



תרשים 4.29: אופרטור האיחוד.

דוגמה : הצג את רשימת מספרי הסטודנט שעיר המגורים שלהם היא חיפה **או** נבחנו בקורס M-100, **או שניהם.**

SELECT *STUDENTS* **WHERE** *CITY = 'Haifa'* **GIVING** *TEMP1*
PROJECT *TEMP1* **OVER (***STUDENT_ID***) GIVING** *TEMP2*
SELECT *GRADES* **WHERE** *COURSE_ID = 'M-100'* **GIVING** *TEMP3*
PROJECT *TEMP2* **OVER (***STUDENT_ID***) GIVING** *TEMP4*
TEMP2 **UNION** *TEMP4* **GIVING** *RESULT*



STUDENT_ID
מס. סטודנט
105
110
240
245
310

חיסור טבלאות (Minus)

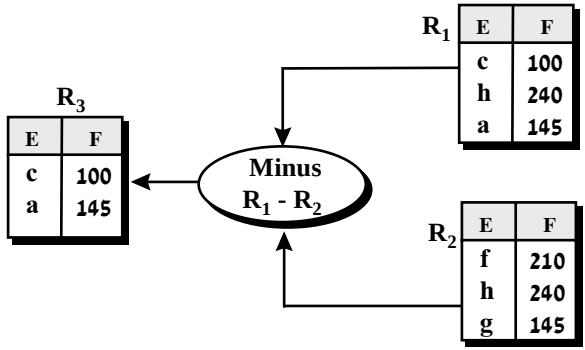
אופרטור החיסור מאפשר להחסיר מטבלה אחת את כל השורות הזהות המופיעות בטבלה השנייה. רק השורות הנותרות מועברות לטבלת התוצאה. כדי שתהיה לתוצאה משמעות כלשהי, הטבלאות חייבות להיות תואמות איחוד. סדר הופעת הטבלאות בפקודה חשוב, בדיוק כמו בפעולת חיסור מתמטית.

מבנה הפקודה :

Table_Name_1 **MINUS** *Table_Name_2*
GIVING *Result_Table_Name*

דוגמה כללית :

R_1 **MINUS** R_2 **GIVING** R_3



תרשים 4.30: אופרטור החיסור.

דוגמה: הצג רשימת מספרי הסטודנט שנבחנו בקורס M-100 ועיר המגורים שלהם אינה חיפה. נבנה את אותן טבלאות כמו בדוגמה הקודמת, ונחסר אותן זו מזו.

SELECT *STUDENTS* **WHERE** *CITY = 'Haifa'* **GIVING** *TEMP1*
PROJECT *TEMP1* **OVER** (*STUDENT_ID*) **GIVING** *TEMP2*
SELECT *GRADES* **WHERE** *COURSE_ID = 'M-100'* **GIVING** *TEMP3*
PROJECT *TEMP2* **OVER** (*STUDENT_ID*) **GIVING** *TEMP4*
TEMP2 **MINUS** *TEMP4* **GIVING** *RESULT*

STUDENT_ID
מס. סטודנט
245
310

חיתוך טבלאות (Intersection)

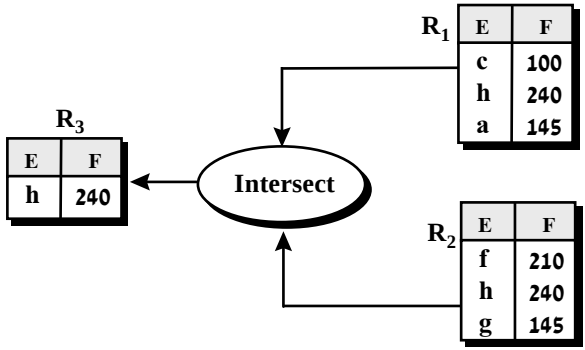
אופרטור החיתוך מאפשר חיתוך בין שתי טבלאות, כך שלטבלת התוצאה יועברו רק שורות המופיעות **בשתי** הטבלאות גם יחד. הטבלאות חייבות להיות תואמות איחוד. אין חשיבות לסדר הופעת הטבלאות משני צידי האופרטור.

מבנה הפקודה :

Table_Name_1 **INTERSECT** *Table_Name_2*
GIVING *Result_Table_Name*

דוגמה כללית:

R_1 **INTERSECT** R_2 **GIVING** R_3



תרשים 4.31: אופרטור החיתוך.

דוגמה: הצג רשימת מספרי הסטודנט שעיר המגורים שלהם חיפה **וגם** נבחנו בקורס M-100. נבנה טבלאות כמו בדוגמה של האיחוד, ונבצע חיתוך ביניהן.

SELECT *STUDENTS* **WHERE** *CITY = 'Haifa'* **GIVING** *TEMP1*
PROJECT *TEMP1* **OVER** (*STUDENT_ID*) **GIVING** *TEMP2*
SELECT *GRADES* **WHERE** *COURSE_ID = 'M-100'* **GIVING** *TEMP3*
PROJECT *TEMP2* **OVER** (*STUDENT_ID*) **GIVING** *TEMP4*
TEMP2 **INTERSECT** *TEMP4* **GIVING** *RESULT*

STUDENT_ID
מס. סטודנט
105

מכפלה קרטזית (Product)

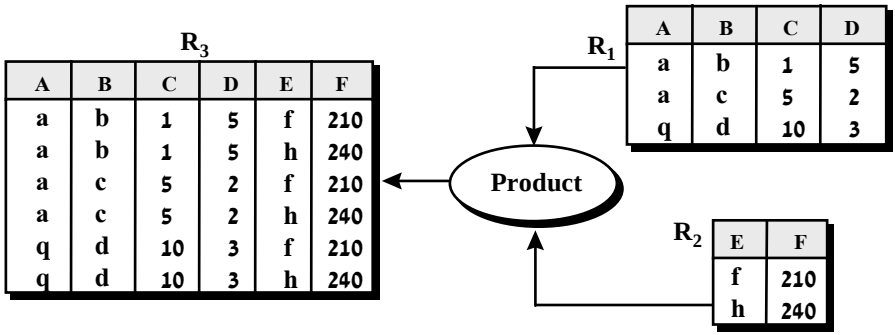
מכפלה קרטזית בין שתי טבלאות פירושה, שכל שורה של טבלה R_1 תצורף לכל אחת מהשורות של טבלה R_2. אם לטבלה R_1 יש n שורות ולטבלה R_2 יש m שורות, לטבלת התוצאה תהינה n x m שורות. הטבלאות לא חייבות להיות תואמות איחוד.

מבנה הפקודה :

Table_Name_1 **PRODUCT** *Table_Name_2*
GIVING *Result_Table_Name*

דוגמה כללית:

R_1 **PRODUCT** R_2 **GIVING** R_3



תרשים 4.32: אופרטור המכפלה.

חילוק טבלאות (Division)

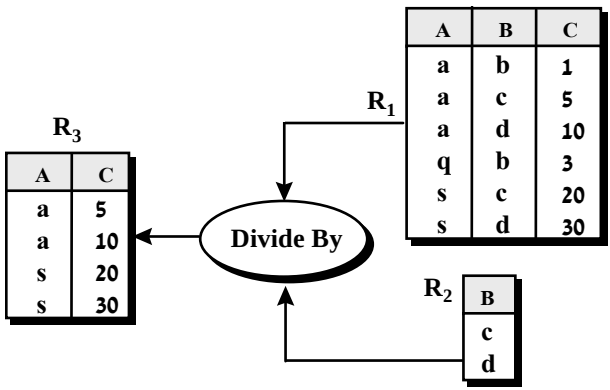
אופרטור החילוק מוצא בטבלה מסוימת (טבלת המונה) את כל השורות שמופיעות גם בטבלה השנייה (טבלת המכנה). שתי הטבלאות לא חייבות להיות תואמות איחוד. יש חשיבות לסדר הופעת הטבלאות, כמו בפעולת חילוק רגילה.

מבנה הפקודה:

Table_Name_1 **DIVIDE BY** *Table_Name_2*
GIVING *Result_Table_Name*

דוגמה כללית:

R_1 **DIVIDE BY** R_2 **GIVING** R_3



תרשים 4.33: אופרטור החילוק.

סיכום

בפרק זה הצגנו את המודל הטבלאי, הבסיס התיאורטי של רוב המערכות המסחריות המודרניות לניהול בסיסי נתונים. מודל זה מצטיין בפשטותו, כי הוא משתמש במונחים המובנים לכל – **טבלאות**. הראינו שניתן לתרגם כל מודל תפישתי לאוסף מתאים של טבלאות.

ניתן להגדיר אוסף של **אופרטורים טבלאיים** שיאפשרו ביצוע פעולות שונות ב**בסיס נתונים טבלאי**. Codd הוכיח ששפה זו **שלמה** (Relational Complete); כלומר, ניתן לשלוף באמצעותה כל פריט מידע מתוך בסיס הנתונים.

האלגברה הטבלאית הינה שפה תיאורטית בלבד, שמטרתה להדגים את הרעיונות של טיפול בטבלאות באמצעות אופרטורים מסוג חדש, לעומת האופרטורים המוכרים השולפים כל פעם רשומה (שורה) אחת. האלגברה הטבלאית מהווה את התשתית התיאורטית שעליה מתבססות שפות טבלאיות רבות עוצמה, כגון שפת SQL שתוצג בהרחבה בחלק ד' של הספר.

ניתן להגיע אל אותה תוצאה על ידי הפעלת האופרטורים השונים בסדר שונה (למשל, להפעיל תחילה PROJECT ולאחר מכן SELECT, או להיפך). לסדר ביצוע הפעולה יכולה להיות השפעה רבה על זמן הביצוע של השאילתה.

לסיכום נדגיש, שכדי לבצע כל פעולה בבסיס הנתונים, יש צורך רק בחמישה אופרטורים בסיסיים מבין שמונת האופרטורים שהוצגו (Select, Product, Project, Union, Minus). ניתן לבצע את הפעולות של שאר האופרטורים על ידי שילוב של האופרטורים הבסיסיים בלבד. לדוגמה, פעולת האופרטור Join דומה לשילוב בין Select- Product.

שאלות חזרה ותרגילים

שאלות חזרה

1. הסבר בקצרה את המושגים הבאים: יחס, סכימה טבלאית, שורה, עמודה, מפתח עיקרי, מפתח זר.
2. הסבר את שני אילוצי האמינות העיקריים של המודל הטבלאי ומדוע יש חשיבות בקיום אילוצים אלה.
3. הסבר אילו פעולות עדכון ניתן לבצע על טבלה ואיזה מאילוצי האמינות ניתן להפר כתוצאה מפעולות אלו.
4. הסבר את האופרטורים של האלגברה הטבלאית ואת המטרה של כל אחד מהם.
5. האם יחס וטבלה הם שני מושגים חופפים? אם לא, הסבר את ההבדלים ביניהם.
6. הסבר כיצד שלושת סוגי הקשרים 1:1, 1:N ו N:M מיוצגים במודל הטבלאי.
7. הסבר את סוגי ה-Join השונים.
8. קרא את המשפטים הבאים וענה בכן/לא:

א. במודל הטבלאי הקשרים מיוצגים על ידי טבלאות מיוחדות ולא על ידי מצביעים פיסיים.

ב. המפתח העיקרי הוא אחד מבין המפתחות הזרים הקיימים בטבלה.

ג. מפתח זר משמש ליצירת קשר בין שתי טבלאות.

ד. מפתח זר אחד יכול לשמש להגדרת מספר קשרים עם טבלאות שונות.

ה. המכפלה הקרטזית מוגדרת כקבוצה חלקית של יחס.

ו. יחס מוגדר כקבוצה חלקית של מכפלה קרטזית.

ז. לכל טבלה יש מפתח זר אחד בלבד.

ח. סכימה טבלאית מוגדרת כאוסף ההגדרות של כל הטבלאות בבסיס הנתונים.

ט. האופרטור Join מחייב ששתי הטבלאות תהיינה תואמות איחוד.

י. האופרטור Project חייב להכיל גם את עמודת המפתח העיקרי, מאחר ואסור שבטבלה התוצאתית תהיינה שורות כפולות.

יא. האופרטור Select בוחר שורות מתוך טבלה המקיימות תנאי לוגי כלשהו.

יב. אלגברה טבלאית היא שפת השאילתות של מערכות RDBMS מסחריות.

יג. אם נפרק טבלה לשתי טבלאות באמצעות האופרטור Project, תמיד נוכל לשחזר את הטבלה המקורית על ידי האופרטור Join.

יד. המכפלה הקרטזית לא ניתנת ליישום באלגברה טבלאית, מאחר ואין אופרטור התומך בפעולה זו.

תרגילים

1. נתונות שלוש הקבוצות הבאות:

   I. $A = \{a, b, c\}$

   II. $B = \{x, y, z\}$

   III. $C = \{h, f\}$

בנה את קבוצת המכפלה הקרטזית.

2. בנה את הסכימה הטבלאית המתאימה למודל ישויות-קשרים של תרגיל 1 בפרק 3. הסבר כיצד טיפלת בהיררכיה הסמנטית של סוגי הסרטים השונים. האם יש רק פתרון אחד למיפוי זה? אם יש מספר פתרונות, הצג אותם ונתח את היתרונות והחסרונות של כל אחד מהם.

3. בנה את הסכימה הטבלאית המתאימה למודל ישויות-קשרים של תרגיל 5 בפרק 3.

4. נתונה הסכימה הטבלאית הבאה:

   א. **מלון** (קוד מלון, שם מלון, כתובת מלון, מספר טלפון, מספר חדרים).

   ב. **חדר במלון** (קוד מלון, מספר חדר, סוג חדר, מספר מיטות, מחיר).

   ג. **אורח במלון** (מספר זהות או דרכון, שם אורח, כתובת, ארץ).

   ד. **הזמנת חדר** (קוד מלון, מספר חדר, מתאריך, עד תאריך, צורת תשלום).

בצע את המטלות הבאות המתייחסות לסכימה הזו:

   א. הגדר את המפתחות העיקריים והזרים של כל טבלה.

   ב. הצג את מודל ישויות-קשרים המתאים לסכימה זו.

   ג. בנה שאילתה באלגברה טבלאית המציגה את רשימת כל החדרים עם שתי מיטות לפחות במלונות שיש בהם מעל 100 חדרים.

   ד. בנה שאילתה באלגברה טבלאית המציגה את כל ההזמנות לחדרים מסוג מסוים שצורת התשלום עבורם בחודשיים האחרונים היתה במזומן.

   ה. בנה שאילתה באלגברה טבלאית המציגה את רשימת שמות האורחים המתארחים כרגע במלון המלך דוד או הילטון ירושלים.

5. חנות מוסיקה החליטה להקים מועדון לקוחות כך שכל לקוח יוכל להזמין את האלבומים המבוקשים דרך אתר האינטרנט ולקבל אותם לביתו. להלן נתוני הבעיה:

א. בחנות אוסף גדול מאוד של אלבומים.

ב. החנות מנהלת אתר אינטרנט ובו פרטים על האלבומים. לכל תקליט מוצג המידע הבא: המספר הקטלוגי של האלבום, שם האלבום, שם הזמר או הלהקה, שנת הוצאה לאור, מחיר האלבום, שם החברה שהפיקה את האלבום, סוג האלבום (פופ, קלאסי, ג'אז, בלוז, מעורב וכד'), מספר השירים באלבום, סוג המדיה האפשרית (CD, קסטה, תקליט). אלבום יכול להימכר במספר מדיות שונות. על פי בקשת הלקוח יש לאפשר גם הדפסה של שמות השירים באלבום ומה אורך השיר בדקות ושניות.

ג. לכל לקוח החנות מנהלת נתונים כגון: מספר לקוח, שם לקוח, כתובת, מספר טלפון, כתובת דאר אלקטרוני, צורת התשלום המועדפת (כרטיס אשראי, משלוח המחאה) ותאריך הצטרפות למועדון הלקוחות.

ד. הלקוח יכול להזמין ישירות דרך אתר האינטרנט את האלבומים המבוקשים ואלה ישלחו ישירות לביתו. עליו למלא טופס הזמנה המכיל את הנתונים הבאים: מספר הזמנה, תאריך הזמנה, מספר לקוח, שם לקוח, מועד משלוח מבוקש, מספר קטלוגי של האלבום המבוקש וכמות מבוקשת. ניתן להזמין בהזמנה אחת מספר רב של אלבומים שונים.

בצע את המטלות הבאות המתייחסות לבעיה זו:

- בנה את מודל ישויות-קשרים המתאים לבעיה.
- הצג את הסכימה הטבלאית המתאימה.
- בנה שאילתה באלגברה טבלאית המציגה את רשימת כל הזמנות בהן מופיע אלבום מסוים ושמועד המשלוח המבוקש הוא 10.2.1999.
- עדכן את הסכימה הטבלאית כך שנוכל לטפל גם באלבומי אוסף, כלומר אלבום המבוצע על ידי מספר זמרים או להקות שונות.
- לכל לקוח יש לדעת כמה אלבומים הוא הזמין. כיצד והיכן תנהל מידע זה? מה משמעות הכנסת כפילות נתונים זו למודל הנתונים?

פרק 5

נירמול נתונים (Data Normalization)

1. מבוא
2. מהי מטרת תהליך נירמול נתונים
3. תלות פונקציונלית (Functional Dependency)
4. חוקי ההיסק של ארמסטרונג
5. מבנה ברמת נירמול ראשונה (First Normal Form)
6. מבנה BCNF (Boyce Codd Normal Form)
7. פירוק טבלאות (Decomposition)
8. תלות רב-ערכית ומצב 4NF
9. שאלות חזרה ותרגילים

מבוא

כפי שהראינו בפרק 3, ניתן לייצג כל קבוצת ישות במודל התפישתי באמצעות טבלה, ולכן ניתן להמיר את המודל התפישתי למודל הטבלאי.

מודל הנתונים הטבלאי מגדיר אילו טבלאות משתתפות, כלומר: את הסכימה הטבלאית. אחת השאלות המעניינות היא: האם כל צירוף של **עמודות** לטבלה מסוימת טוב באותה מידה כמו צירוף אחר, או שמא קיימים צירופים טובים מאחרים.

תהליך הנירמול, אותו נציג בפירוט בהמשך, פותח במקור על ידי Codd [CE72b] ושוכלל על ידי חוקרים נוספים. מטרת התהליך היא להבטיח שבמעבר בין המודל התפישתי לבין המודל הטבלאי ייבנו טבלאות "טובות".

בהמשך יוצג הנושא בדרך אינטואיטיבית ולא בדרך פורמלית-מתמטית. הקורא המעוניין להרחיב את ידיעותיו ולהכיר הוכחות פורמליות, מופנה לספרות הענפה בנושא שמופיעה ברשימה הביבליוגרפית [DC90, MD83, UW97].

בפרק זה

- נציג את מטרת תהליך נירמול הנתונים, סילוק תופעות לא רצויות מבסיס הנתונים.
- נציג את המושג "תלות פונקציונלית". זהו מושג בסיסי להבנת הקשרים הקיימים בין עמודות הטבלה.
- נציג את תרשים התלויות הפונקציונליות, המאפשר הצגה גרפית של כל התלויות הפונקציונליות הקיימות בין העמודות של טבלה.
- נגדיר את כלל BCNF לבדיקת מצב הטבלה ולאיתור כפילויות הנתונים.
- נסביר מהו פירוק תקין ופירוק לא תקין של טבלה לטבלאות אחרות.

מהי מטרת נירמול נתונים

נתחיל את הדיון בתהליך הנירמול באמצעות דוגמה. נניח, שבשלב בניית המודל התפישתי החליט המעצב לבנות את קבוצת הישות קורסים, כך שניתן לייצג אותה על ידי הטבלה הבאה:

קורסים (מספר קורס, שם קורס, קוד מחלקה, שם מחלקה)

התדפיס שבתרשים 5.1 מציג תוכן אפשרי של טבלה זו. ניתן לראות בתדפיס שיש **כפילות נתונים** (Data Redundancy), כי שם מחלקה חוזר על עצמו עבור כל אחד מהקורסים הניתן על ידי אותה מחלקה. מעבר לעובדה שזהו בזבוז של מקום אחסון, ניתן לומר שקיים כאן פוטנציאל לבעיות בבסיס הנתונים. לדוגמה, שינוי שם המחלקה הופך לתהליך מורכב המחייב עדכון של מספר רב של שורות, במקום עדכון חד-פעמי.

COURSE_ID	COURSE_NAME	DEP_ID	DEP_NAME
מס. קורס	שם קורס	מחלקה	שם מחלקה
C-200	Programming	CS	Computer Science
C-300	Pascal	CS	Computer Science
C-55	Data Bases	CS	Computer Science
C-100	Operating Sys.	MT	Mathematics
M-100	Numeric Analysis	MT	Mathematics
M-110	Operation Res.	BS	Business

תרשים 5.1: טבלת קורסים עם כפילות נתונים.

מתוך דיון קצר זה ניתן להבין, שלא כל צירוף של עמודות לטבלה מסוימת הוא טוב. לתופעות הלא רצויות הנוצרות כתוצאה מכפילות הנתונים מקובל לקרוא בשם **אנומליות** (Anomalies). מקובל להבחין באנומליות הבאות:

- **אנומליית ההוספה (Insertion Anomaly):** אנומליה זו נוצרת כאשר רוצים להוסיף לטבלה את העובדה שנפתחה מחלקה חדשה שבשלב זה עדיין אינה מציעה אף קורס שהוא. מכיון שמפתח הטבלה הוא מספר הקורס, לא ניתן להוסיף שורה חדשה עד שלא יוגדר הקורס הראשון שהמחלקה מציעה.
- **אנומליית הביטול (Deletion Anomaly):** אנומליה זו נוצרת כאשר רוצים לבטל שורה בטבלה ויוצרים תופעת לוואי של איבוד נתונים נוספים. בדוגמה שלנו, אם נבטל את השורה של קורס M-110 נאבד באותה הזדמנות גם את המידע שהקוד של המחלקה למנהל עסקים הוא BS, מכיון שזהו הקורס היחיד שמחלקה זו מציעה.
- **אנומליית העדכון (Update Anomaly):** אנומליה זו נוצרת כאשר מבקשים לעדכן נתון מסוים. מכיון שבטבלה יש כפילות נתונים, יכול להיווצר מצב שבשורות מסוימות הנתון מעודכן ובשורות אחרות הוא אינו מעודכן. בדוגמה שלנו, אם נעדכן את שם המחלקה "מדעי המחשב" בשורה הראשונה בלבד ל- Computer Engineering, נקבל מצב שקוד מחלקה CS מתייחס לשני שמות מחלקה שונים.

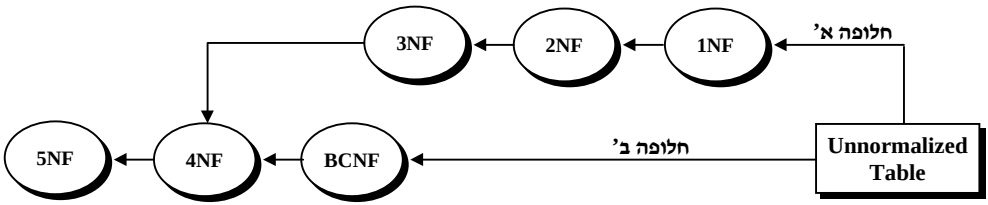
האנומליות האלו נוצרו כתוצאה משילוב נתונים של שתי ישויות שונות, קורסים ומחלקות. בדוגמה שהצגנו, הבעיה בולטת לעין וסביר להניח שמעצב נתונים מנוסה יגלה מצב זה מיידית ויגדיר מבנה נתונים נכון. הבעיה היא כמובן, שבמציאות יתכנו מצבים מורכבים בהרבה ולכן הסתמכות על הניסיון והאינטואיציה של מעצב בסיס הנתונים אינה מתכון מומלץ. כאן נכנס לתמונה תהליך נירמול הנתונים, שתפקידו לגלות מצבים אלה על ידי תהליך שיטתי ופורמלי, ולא על ידי ניחוש ואינטואיציה.

נירמול נתונים **Data Normalization**	***נירמול נתונים*** **הינו תהליך שמטרתו לייצר אוסף של טבלאות המייצגות את המודל התפישתי בצורה מדויקת ואמינה, ושאינן כוללות תופעות לא רצויות הנובעות מכפילות נתונים.**

תהליך הנירמול הוצג לראשונה על ידי E.F. Codd בשנת 1972 [CE72b]. התהליך מורכב ממספר שלבי פעולה ובדיקות המאפשרות לבחון האם אוסף הטבלאות מקיים אילוצים מסוימים, כפי שהוגדרו באמצעות המושג **מבנה מנורמל** (Normal Form).

Codd הציג במאמר שלושה מבנים מנורמלים וכינה אותם בשם **מבנה מנורמל ברמה ראשונה - 1NF** (First Normal Form), **מבנה מנורמל ברמה שנייה - 2NF** (Second Normal Form) **ומבנה מנורמל ברמה שלישית - 3NF** (Third Normal Form). ככל שמתקדמים ברמת הנירמול, נפטרים מתופעות לא רצויות רבות יותר במבנה הנתונים והטענה היתה, שאם מבנה הנתונים הינו מבנה מנורמל ברמה שלישית, הוא משוחרר מכל התופעות הלא רצויות הנובעות מכפילות נתונים. בשנת 1974 פרסם Codd יחד עם החוקר R. Boyce הגדרה טובה יותר של המבנה המנורמל, לאחר שהתברר שגם במבנה מנורמל ברמה שלישית עדיין יתכנו תופעות לא רצויות. ההגדרה החדשה נקראת על שם שני החוקרים שפיתחו אותה, Boyce-Codd Normal Form או בקיצור **BCNF**.

בשנים מאוחרות יותר פורסמו מספר מאמרים שהציעו רמות נירמול גבוהות יותר. Fagin הציע בשנת 1978 [FR78] מבנה מנורמל ברמה רביעית – **4NF** (Fourth Normal Form) ובשנת 1979 מבנה מנורמל ברמה חמישית – **5NF** (Fifth Normal Form). למרות הדיוק הפורמלי העומד מאחורי מבנים נורמלים אלה, הם מטפלים במצבים נדירים יחסית, ולכן חשיבותם היא יותר במישור הפורמלי-מתמטי מאשר במישור המעשי. רמת נירמול BCNF היא הרמה המעשית שבה מעצבי בסיסי נתונים טבלאיים משתמשים, ולכן נציג אותה בהמשך הפרק.



תרשים 5.2: שתי חלופות לתהליך נירמול נתונים.

כפי שניתן לראות מתרשים 5.2, קיימות שתי חלופות לביצוע נירמול. שתי החלופות מתחילות מטבלה הנמצאת במצב לא מנורמל וממנה, על ידי בדיקה שיטתית של קיום אילוצים מסוימים מתקדמים לקראת רמות נירמול גבוהות יותר. חלופה א' היא השיטה הישנה יותר, ואילו חלופה ב' היא המקובלת כיום ומבוססת על מבנה BCNF.

כפי שנראה מייד, ניתן לזהות את האנומליות ואת התופעות הלא רצויות במבנה הנתונים בעזרת ניתוח **התלות הפונקציונליות** בין העמודות השונות של הטבלה.

תלות פונקציונלית
(Functional Dependency)

נתחיל את הצגת נירמול הנתונים בהצגת המושג המרכזי – **תלות פונקציונלית**.

תלות פונקציונלית **Functional Dependency**	***תלות פונקציונלית*** **קיימת בין עמודה B לבין עמודה A, אם בכל נקודת זמן כל ערך של A קשור לערך אחד של B לכל היותר. ניתן גם לומר, ש-A מגדיר את B.**

התלות הפונקציונלית קובעת שקיימת פונקציה F כלשהי, שהינה אוסף מסודר של זוגות <a,b>, כך שהערך a נלקח מתוך מרחב הערכים של A ו-b נלקח מתוך מרחב הערכים של B, ובכל נקודת זמן עבור כל ערך a קיים ערך b אחד לכל היותר.

$$F = \{<a,b> \mid a \in A,\ b \in B\}$$

משמעות הדבר היא שבכל שורה בטבלה שבה מופיע הערך a בעמודה A, יופיע הערך b בעמודה B. אם נסמן ב- t_i את השורה i ביחס R ואם השורות i ו-j הן שורות ביחס R, כלומר מתקיים $t_i, t_j \in R$, אזי:

כל פעם שמתקיים $t_i[a] = t_j[a]$ מתקיים גם $t_i[b] = t_j[b]$

מבחינה גרפית נסמן את התלות הפונקציונלית הזו על ידי חץ, $A \rightarrow B$. משמעות הסימון היא ש-B תלוי תלות פונקציונלית ב-A, או לחילופין: A מגדיר את B.

C	B	A
c_1	b_4	a_1
c_4	b_2	a_2
c_3	b_4	a_1
c_9	b_7	a_3
c_2	b_4	a_1
c_1	b_7	a_3

תרשים 5.3: טבלה המקיימת תלות פונקציונלית $A \rightarrow B$.

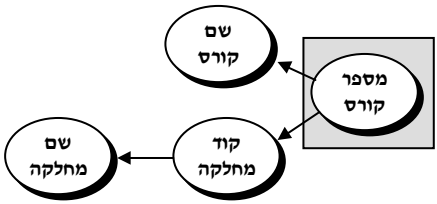
כפי שניתן לראות מהתרשים, בכל פעם שבעמודה A מופיע הערך a_1, בעמודה B מופיע הערך b_4. ניתן להרחיב את הגדרת התלות הפונקציונלית, כך שאוסף עמודות אחד יהיה תלוי באוסף עמודות אחר. נקבל עכשיו:

$$[A_1\ A_2\ A_3\ \ldots A_n] \rightarrow [B_1\ B_2\ B_3 \ldots\ B_m]$$

תלות פונקציונלית תקרא **טריוויאלית**, אם $B_i \subseteq A_j$; כלומר, אוסף העמודות B מוכַל בתוך אוסף העמודות A.

בדוגמה הקודמת קיימת תלות פונקציונלית בין שם קורס לבין מספר קורס; כלומר, כל ערך של מספר קורס מגדיר ערך אחד של שם קורס. באופן דומה, קיימת תלות פונקציונלית בין קוד מחלקה לבין מספר קורס. האם נכון יהיה לומר שקיימת תלות פונקציונלית בין שם מחלקה לבין מספר קורס? אכן קיימת תלות פונקציונלית, אולם היא אינה ישירה. מדויק יותר לומר, שקיימת תלות פונקציונלית בין שם מחלקה לבין קוד מחלקה ובין קוד מחלקה לבין מספר קורס; כלומר, אין כאן תלות פונקציונלית ישירה, אלא עקיפה.

ניתן להציג את התלויות הפונקציונליות שבין העמודות, על ידי **תרשים התלויות הפונקציונליות** (Functional Dependencies Diagram). את המפתח העיקרי של הטבלה נקיף בריבוע.



תרשים 5.4: תרשים תלויות פונקציונליות.

נבנה עתה תרשים תלויות פונקציונליות של דוגמה מורכבת יותר. נניח שבתרשים ישויות-קשרים מופיעה קבוצת הישות הזו:

תרשים 5.5: קבוצה ראשונית לפני ניתוח תלויות.

ניתן לבצע המרה של קבוצה זו לטבלה זו:

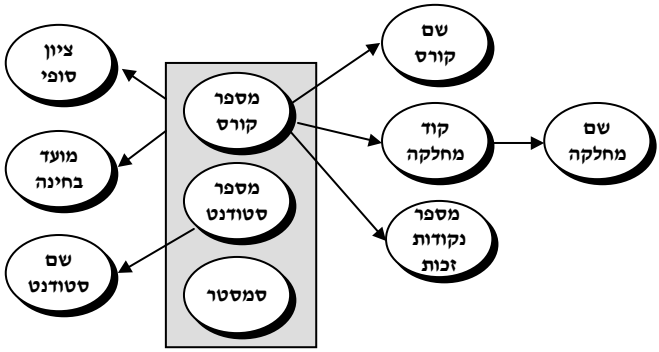
הישגים (מספר קורס, מספר סטודנט, סמסטר, שם קורס, שם סטודנט, קוד מחלקה, שם מחלקה, מספר נקודות זכות, ציון סופי, מועד בחינה).

מתוך ניתוח התלויות הפונקציונליות ניתן להגיע למסקנות אלו:

- מספר הקורס מגדיר את שם הקורס, את מספר נקודות הזכות ואת קוד המחלקה.
- מספר הסטודנט מגדיר את שם הסטודנט.

- קוד המחלקה מגדיר את שם המחלקה.
- מספר הקורס יחד עם מספר הסטודנט והסמסטר מגדירים את הציון הסופי ואת מועד הבחינה.

ניתן לרשום את המסקנות באופן גרפי, בתרשים תלויות פונקציונליות כמודגם בתרשים:



תרשים 5.6: תרשים תלויות פונקציונליות.

חלק מהתלויות הפונקציונליות מתייחסות לעמודה בודדת (כמו במקרה של שם קורס ומספר קורס) וחלקן מתייחסות לאוסף עמודות (כמו במקרה של ציון סופי התלוי בצירוף שבין מספר קורס לבין מספר סטודנט ולסמסטר).

הקורא מוזמן לבנות טבלה המבוססת על הדוגמה שלעיל, וליצור מספר מצבים אפשריים, כמו למשל: אותו סטודנט לומד במספר קורסים שונים, אותה מחלקה מציעה מספר קורסים שונים, מספר סטודנטים שונים לומדים באותו קורס וכד'. הכפילויות שנוצרו בטבלה ודאי בולטות לעין.

חוקי ההיסק של ארמסטרונג

חוקי היסק התלויות מאפשרים להסיק קיום של תלות פונקציונלית אחת מתוך קיום של תלויות פונקציונליות אחרות. לדוגמה, אם מתקיים מספר קורס ← מספר מחלקה, ומתקיים מספר מחלקה ← שם מחלקה, ניתן להסיק גם, שמתקיים מספר קורס ← שם מחלקה. חוקי ההיסק הוצגו כאן באמצעות המונחים הפורמליים: **יחס** (Relation) ו**תכונה** (Attribute).

להלן חוקי ההיסק הבסיסיים:

- **החוק הרפלקסיבי** (Reflexive Rule): אם X ו-Y הן אוסף של תכונות ביחס R, ו-Y מוכלת בתוך X, כלומר: $Y \subseteq X$, אזי קיימת תלות פונקציונלית $X \rightarrow Y$.
- **חוק ההרחבה** (Augmentation Rule): אם X, Y ו-W הן אוסף של תכונות ביחס R ומתקיימת התלות הפונקציונלית $X \rightarrow Y$, אזי תתקיים גם התלות הפונקציונלית $WX \rightarrow WY$. חוק זה קובע שהוספת תכונה בשני צידי התלות הפונקציונלית, יוצרת תלות פונקציונלית חוקית אחרת.

- **החוק הטרנסיטיבי** (Transitive Rule): אם X, Y ו-W הן אוסף של תכונות ביחס R ומתקיימת התלות הפונקציונלית $X \rightarrow Y$, אזי תתקיים גם התלות הפונקציונלית $X \rightarrow W$.
- **חוק האיחוד** (Union Rule): אם X, Y ו-W הן אוסף של תכונות ביחס R ומתקיימות התלויות הפונקציונליות $X \rightarrow Y$ וגם $X \rightarrow W$, אזי תתקיים גם התלות הפונקציונלית $X \rightarrow WY$. חוק זה קובע שניתן להוסיף תכונה לאגף ימין של התלות הפונקציונלית ועדיין לקבל תלות פונקציונלית חוקית. משמעות הדבר היא, שנוכל לבנות תלות מהסוג $X \rightarrow [A1, A2,..An]$ מתוך אוסף תלויות אלו: $X \rightarrow A1$, $X \rightarrow A2$, $X \rightarrow An$.
- **חוק הפירוק** (Decomposition Rule): אם X, Y ו-W הן אוסף של תכונות ביחס R ומתקיימת התלות הפונקציונלית $X \rightarrow WY$, אזי מתקיימות גם התלויות הפונקציונליות $X \rightarrow Y$ ו- $X \rightarrow W$. חוק זה קובע שניתן לסלק תכונה מאגף ימין של התלות הפונקציונלית ועדיין לקבל תלות פונקציונלית חוקית. משמעות הדבר היא, שנוכל לפרק תלות מהסוג $X \rightarrow [A1, A2,..An]$ לאוסף תלויות אלו: $X \rightarrow A1$, $X \rightarrow A2$, $X \rightarrow An$.
- **החוק הטרנסטיבי החלקי** (Pseudo Transitive Rule): אם X, Y, W ו-Z הן אוסף של תכונות ביחס, ומתקיימות התלויות הפונקציונליות $X \rightarrow Y$, $WY \rightarrow Z$ - אזי מתקיימת גם התלות הפונקציונלית $WX \rightarrow Z$.

שלושת החוקים הראשונים מבין השישה שמפורטים למעלה הוצגו על ידי **ארמסטרונג** (Armstrong) [AD80] ומהם ניתן לפתח את שלושת החוקים הנוספים. משמעות הדבר היא שאם נתון יחס R ואוסף של תלויות F, ניתן להסיק את כל התלויות המתקיימות בין תכונות היחס תוך שימוש בשלושת החוקים הראשונים. לאוסף כל התלויות הפונקציונליות שניתן להסיק מחוקים אלה מקובל לקרוא בשם: **הסגור של F** (Closure of F). נסמן אוסף זה ב- F^+. החוקים הנוספים מפשטים את תהליך ההסקה.

חשיבות חוקי היסק התלויות היא בכך שניתן להתחיל מאוסף מסוים של תלויות פונקציונליות הנובעות מהסמנטיקה של היחס, ולפתח באופן שיטתי את קבוצת הסגור, כלומר, את כל התלויות הנובעות מהאוסף הראשוני.

אם נתונה קבוצה של תכונות X ביחס R ונתונה קבוצה F של תלויות פונקציונליות בקבוצה זו, ניתן לפתח בקלות אלגוריתם שמאפשר להסיק את אוסף כל התכונות שתלויות פונקציונלית בקבוצה X תחת F. לקבוצת הסגור של X תחת אוסף התלויות F (Closure of X under F) נקרא "**X^+ תחת F**".

```
X+ := X;
REPEAT UNTIL (oldX+ = X+)
        oldX+ := X+;
        FOR EACH Functional Depenedency Y → Z in F
                DO
                  IF Y ⊆ X+ THEN X+ := X+ ∪ Z ;
                END DO ;
END REPEAT;
```

תרשים 5.7: אלגוריתם לבניית קבוצת הסגור של X תחת F.

האלגוריתם מתחיל בהכנסת כל התכונות של X בקבוצת הסגור X^+. חוקי ההיסק מאפשרים להוסיף תכונות לקבוצה X^+ כתוצאה מקיום התלויות הפונקציונליות. נמשיך להוסיף תכונות בלולאה שתסתיים כאשר אין יותר תכונות להוספה אל הקבוצה X^+.

דוגמה : נניח שאוסף התלויות הפונקציונליות F בטבלה הוא :

- מספר הקורס ← שם הקורס, מספר נקודות הזכות, קוד המחלקה.
- מספר הסטודנט ← שם הסטודנט.
- קוד המחלקה ← שם המחלקה.

נגדיר את אוסף כל העמודות התלויות במספר קורס :

X^+ = {קוד קורס}

על סמך התלות הפונקציונלית הראשונה נקבל :

X^+ = {קוד קורס, שם קורס, מספר נקודות זכות, קוד מחלקה}

על סמך התלות הפונקציונלית השנייה נקבל :

X^+ = {קוד קורס, שם קורס, מספר נקודות זכות, קוד מחלקה, שם סטודנט}

על סמך התלות הפונקציונלית השלישית :

X^+ = {קוד קורס,שם קורס,מספר נקודות זכות,קוד מחלקה,שם סטודנט,שם מחלקה}

אחד השימושים המקובלים לחוקי ההיסק הוא הבדיקה אם המפתח של הטבלה אכן מגדיר את כל יתר העמודות בטבלה.

מבנה ברמת נירמול ראשונה (First Normal Form)

טבלה לא מנורמלת **Unnormalized Table**	**טבלה לא מנורמלת היא טבלה שבה יותר מערך אחד קשור לעמודה מסוימת של הטבלה; כלומר, קיימות עמודות מרובות ערכים בטבלה.**

הדרך הנוחה ביותר להבין הגדרה זו היא באמצעות דוגמה :

מספר עובד	שם העובד	שם ילד
123	משה	אילנה, נתן
145	דן	צבי, עדנה, רותי
258	מרים	דב, שלמה
265	צילה	

תרשים 5.8: טבלה לא מנורמלת.

טבלה זו אינה מנורמלת, מפני שלעמודה שם ילד קשורים מספר ערכים שונים עבור אותה שורה, שהרי ייתכן שלעובד אחד יהיו מספר ילדים. בדוגמה זו שם הילד חוזר על עצמו מספר פעמים, כי זו תכונה מרובת ערכים (Multiple Valued Attribute). במצב זה קשה מאוד לבצע פעולות בבסיס הנתונים. לדוגמה, אחזור כל העובדים שיש להם ילד בשם נתן הוא פעולה מורכבת, כי צריך לסרוק מספר רב של מחרוזות באורך בלתי ידוע ופעולה זו מאוד לא יעילה. ביטול שם של ילד הוא בעייתי, כי צריך לצמצם את המחרוזת ועוד.

כדי לפשט את תהליכי עדכון בסיס הנתונים, ובמיוחד במקרה של בסיס נתונים טבלאי המשתמש בשפת SQL, יש להעביר את הטבלה הלא-מנורמלת למצב ראשוני, שהוא **מבנה מנורמל ברמה ראשונה** (1NF - First Normal Form).

טבלה במצב 1NF	**בטבלה במצב 1NF כל עמודה מכילה ערך אטומי אחד בלבד, ולא אוסף של ערכים.**

המעבר למבנה מנורמל ברמה ראשונה יבוצע על ידי סילוק העמודות מרובות הערכים, במידה וקיימות כאלו. נדגיש שבמעבר זה עדיין לא נלקחים בחשבון שיקולי תלות פונקציונלית, אלא רק הרצון לקבל טבלה דו-מימדית פשוטה, שבה כל ערך קשור לעמודה אחת בלבד.

הטבלה המופיעה בתרשים הקודם איננה במצב 1NF. כדי להעביר אותה למצב הרצוי, נסלק את העמודה מרובת הערכים על ידי פירוק הטבלה לשתי טבלאות נפרדות, טבלת עובדים וטבלת ילדי עובדים, כמודגם בתרשים הבא.

עובדים

מספר עובד	שם העובד
123	משה
145	דן
258	מרים
265	צילה

ילדי עובדים

מספר עובד	שם ילד
123	אילנה
123	נתן
145	צבי
145	עדנה
145	רותי
258	דב
258	שלמה

עובדים (מספר עובד , שם עובד)
ילדי עובדים (מספר עובד , שם ילד)

תרשים 5.9: פירוק טבלה לא-מנורמלת לשתי טבלאות 1NF.

נשים לב למפתח של טבלת ילדי עובדים הכולל מספר עובד ושם ילד.

מבנה BCNF
(Boyce Codd Normal Form)

מתוך ניתוח התלויות הפונקציונליות מתברר, שניתן להגדיר מצב מסוים שבו לא תתקיימנה תופעות לא רצויות בבסיס הנתונים.

מבנה BCNF	**טבלה נמצאת במצב BCNF אם כל פעם שקיימת תלות פונקציונלית $X \rightarrow Y$ ו-Y אינו כלול ב-X, X מכיל מפתח אפשרי (Candidate Key).**

ההגדרה קובעת, שכל פעם שעמודה (או אוסף עמודות) מגדירה עמודה אחרת, עליה להיות מפתח אפשרי של הטבלה; ואם לא, תיווצרנה תופעות בלתי רצויות. הדרך לטפל בתופעות לא רצויות אלו היא על ידי פירוק הטבלה לאוסף אחר של טבלאות, כך שכל אחת מהן תהיה במצב BCNF.

לדוגמה, נעיין בטבלה הישגים שהצגנו קודם.

הישגים (מספר קורס, מספר סטודנט, סמסטר, שם קורס, שם סטודנט, קוד מחלקה, שם מחלקה, מספר נקודות זכות, ציון סופי, מועד בחינה).

נרשום את כל העמודות המגדירות שמצאנו:

מספר קורס ← שם קורס, מספר נקודות זכות, מספר מחלקה
מספר מחלקה ← שם מחלקה
מספר סטודנט ← שם סטודנט
מספר קורס, מספר סטודנט, סמסטר ← ציון סופי, מועד מבחן

מבין כל העמודות המגדירות, רק האוסף מספר קורס, מספר סטודנט, סמסטר מכיל מפתח. כל העמודות המגדירות האחרות אינן מכילות מפתח. מכאן נובע, הטבלה אינה במצב BCNF.

כפי שהוסבר, Codd הגדיר תהליך נירמול שמתחיל מטבלה הנמצאת במצב 1NF. על ידי סילוק כל התלויות החלקיות (כלומר עמודות שתלויות רק בחלק מהעמודות המרכיבות את המפתח) נקבל מבנה 2NF. על ידי סילוק כל התלויות העקיפות (כלומר עמודה תלויה בעמודה אחרת הקשורה למפתח) נקבל מבנה 3NF. בשלב מאוחר יותר הסתבר, שבמצבים מסוימים טבלאות הנמצאות במבנה 3NF עדיין מכילות כפילויות וכך נוצר מבנה BCNF. לכל צורך מעשי, ההגדרה של BCNF היא פשוטה יותר והפירוק של טבלה במבנה 1NF לאוסף טבלאות BCNF נעשה בצעד אחד, ללא צורך במספר רב של צעדי ביניים.

פירוק טבלאות (Decomposition)

סילוק תופעות לא-רצויות מבוצע על ידי פירוק טבלה שמכילה אנומליות ואינה במצב BCNF, לאוסף של טבלאות קטנות יותר. שיטת הפירוק מבוססת על שני עקרונות אלה:

- כל הטבלאות המתקבלות צריכות להיות במצב BCNF.
- כל הטבלאות המתקבלות צריכות לייצג באופן נאמן את המידע המופיע בטבלה המקורית.

לא כל פירוק של טבלה למספר טבלאות הוא פירוק טוב. מטרת פירוק "טוב" היא לשַמֵר את הנתונים או המידע, ולא לאבד אותם, כדי שהתוכן הסמנטי יישמר. פירוק כזה ייקרא **פירוק משמר תלויות**.

פירוק משמר תלויות **Loseless Join Decomposition**	**פירוק של טבלה T לשתי טבלאות R ו-S יהיה *פירוק משמר תלויות*, אם ניתן לשחזר את טבלה T מתוך טבלאות R ו-S תוך שימוש באופרטור Join. משמעות הדבר היא שניתן לקבל את התלויות הפונקציונליות של T על ידי איחוד התלויות הפונקציונליות של S ושל R.**

ניתן לומר, שתמיד קיים פירוק של T ל- R ול- S המשמר תלויות. אסטרטגיית הפירוק שבה ננקוט היא, שכל פעם שנמצא בטבלה T תלות פונקציונלית לא טריוויאלית: $A_1, A_2, \ldots A_n \rightarrow B_1, B_2, \ldots B_m$ המפרה את כלל BCNF, נצרף לאגף ימין של התלות את כל התכונות התלויות בתכונות הנמצאות באגף שמאל ונבנה טבלה חדשה שבה $A_1, A_2, \ldots A_n$ הוא המפתח.

לדוגמה, נתונה הטבלה סטודנטים הבאה:

סטודנטים (<u>מספר סטודנט</u>, שם סטודנט, כתובת)

התלויות הפונקציונליות בטבלה הן מספר סטודנט ← שם הסטודנט וכתובת הסטודנט. ניתן לפרק טבלה זו לשתי הטבלאות הבאות:

שמות סטודנטים (<u>מספר סטודנט</u>, שם סטודנט)
כתובות (<u>כתובת</u>)

זהו פירוק שאינו משמר את התלויות הפונקציונליות, כי התלות בין כתובת לבין מספר סטודנט נעלמה. לא ניתן לשחזר את הטבלה סטודנטים המקורית מתוך שתי הטבלאות החדשות שמות סטודנטים ו-כתובות. אם רוצים לפרק טבלה זו באופן שהתלויות תשמרנה, נעשה זאת כך:

שמות סטודנטים (<u>מספר סטודנט</u>, שם סטודנט)
כתובות (<u>מספר סטודנט</u>, כתובת)

מכאן נובע, שפירוק טבלה שאינה במצב BCNF לאוסף טבלאות אחר, שכל אחת מהן תהיה במצב BCNF, יכול להיעשות על ידי בניית טבלאות חדשות, כך שכל עמודה מגדירה תהיה מפתח.

נחזור לדוגמה של טבלת הישגים:

הישגים (מספר קורס, מספר סטודנט, סמסטר, שם קורס, שם סטודנט, קוד מחלקה, שם מחלקה, מספר נקודות זכות, ציון סופי, מועד בחינה).

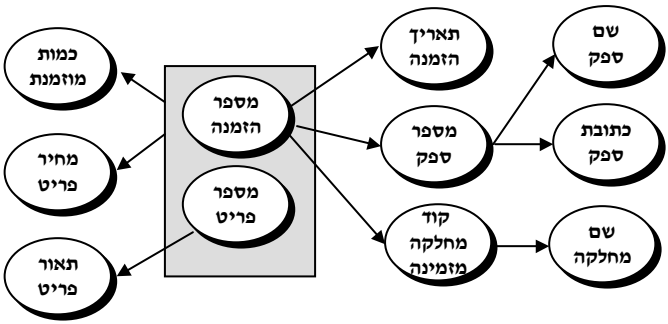
על סמך ניתוח התלויות ועל פי הגדרת מצב BCNF נוכל לפרק את הטבלה לטבלאות אלו:

סטודנטים (מספר סטודנט , שם סטודנט)
קורסים (מספר קורס , שם קורס, מספר נקודות זכות, מספר מחלקה)
מחלקות (מספר מחלקה , שם מחלקה)
הישגים (מספר קורס , מספר סטודנט , סמסטר , ציון סופי, מועד בחינה)

נציג דוגמה נוספת לניתוח תלויות ופירוק משמר תלויות לטבלאות BCNF. נניח, שבשלב המרת המודל התפישתי בנה המעצב את הטבלה הזמנות עם העמודות האלו:

הזמנות (מספר הזמנה, תאריך הזמנה, מספר ספק, שם ספק, כתובת הספק, קוד מחלקה מזמינה, שם מחלקה מזמינה, מספר פריט, תיאור פריט, כמות מוזמנת, מחיר פריט)

המפתח העיקרי של הטבלה, הוא הצירוף של מספר הזמנה עם מספר פריט, מכיון שכל הזמנה יכולה להכיל מספר פריטים. התרשים הבא מציג את תרשים התלויות הפונקציונליות של הטבלה.



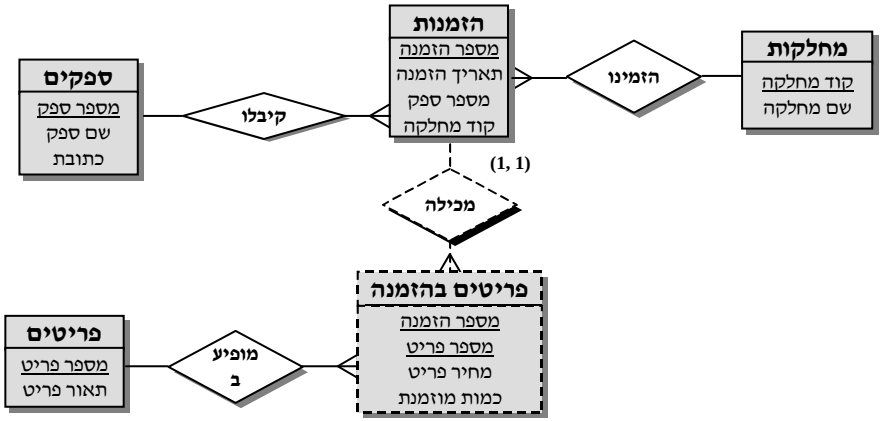
תרשים 5.10: תרשים תלויות פונקציונליות.

התכונות המגדירות שהתקבלו: מספר הזמנה + מספר פריט, מספר הזמנה, מספר פריט, מספר ספק, קוד מחלקה מזמינה. מבין כל העמודות המגדירות האלו, רק מספר הזמנה + מספר פריט הן מפתח. מכאן ברור שהטבלה אינה במצב BCNF ולכן יש לפרקה למספר טבלאות.

פירוק משמר תלויות יהיה:

פריטים (מספר פריט, תיאור פריט)
מחלקות (קוד מחלקה, שם מחלקה)
ספקים (מספר ספק, שם ספק, כתובת ספק)
הזמנות (מספר הזמנה, תאריך הזמנה, מספר ספק, קוד מחלקה מזמינה)
פריטים בהזמנה (מספר הזמנה, מספר פריט, מחיר פריט, כמות מוזמנת)

התרשים הבא מציג תרשים ישויות-קשרים המעודכן המתקבל אחרי הפירוק לטבלאות BCNF.



תרשים 5.11: תרשים ישויות-קשרים.

תלות רב-ערכית ומצב 4NF

טבלה במצב BCNF עלולה להכיל כפילויות. תופעה זו התגלתה על ידי Fagin והביאה להגדרת **תלות פונקציונלית רב-ערכית** (Multi Valued Dependency). כדי להסביר את הנושא נשתמש בדוגמה. נניח, שנתונה הטבלה הזו:

ספרים-מרצים (מספר מרצה, מספר קורס, שם ספר)

בקורס אחד יכולים ללמד מספר מרצים שונים (כל מרצה מלמד קבוצת לימוד אחרת), ובאותו קורס יכולים להשתמש במספר ספרי לימוד. כיון שכל העמודות בטבלה ספרים-מרצים משתתפות במפתח, הטבלה במצב BCNF. התלות הפונקציונלית הרגילה אינה תופסת כאן, מאחר והיא מתייחסת לכך **שערך אחד** של תכונה A מגדיר **קבוצת ערכים** של תכונה B.

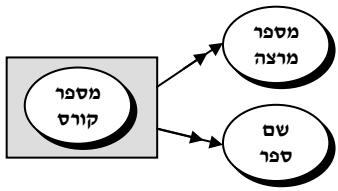
עיון בטבלה שבתרשים 5.12 מבהיר שקיימת כפילות, למרות שבאופן פורמלי הטבלה נמצאת במצב BCNF.

מספר מרצה	מספר קורס	שם ספר
125	E-123	מבוא לכלכה
125	E-123	כלכלה למתחילים
125	E-123	עקרונות הכלכלה
245	E-123	מבוא לכלכה
245	E-123	כלכלה למתחילים
245	E-123	עקרונות הכלכלה

תרשים 5.12: טבלה במצב BCNF עם כפילויות.

תלות פונקציונלית רב-ערכית **Multi Valued Dependency**	***תלות פונקציונלית רב-ערכית*** **קיימת בין עמודה B לבין עמודה A, אם בכל נקודת זמן כל ערך של A קשור לאוסף כלשהו של ערכים של B, באופן בלתי תלוי בעמודה C כלשהי.**

מבחינה גרפית נסמן את התלות הרב-ערכית על ידי $A \rightarrow\rightarrow B$. עבור טבלת ספרים-מרצים נקבל את תרשים התלויות הפונקציונליות הזה:



תרשים 5.13: תרשים תלויות פונקציונליות עם תלות רב-ערכית.

מצב 4NF	**טבלה נמצאת במצב *4NF* כאשר היא נמצאת במצב BCNF, ולא קיימות בה תלויות רב-ערכיות.**

בדוגמה שלנו, כדי להביא את הטבלה למצב 4NF, נבצע את הפירוק הזה:

קורסים-מרצים (מספר קורס, מספר מרצה)
קורסים-ספרים (מספר קורס, שם ספר)

למען הדיוק פותחו גם מבנים ברמת נירמול גבוהה יותר מ-4NF. מצב 5NF מטפל במצב מיוחד במינו ונקרא גם בשם "Project - Join Normal Form". מקובל לומר שטבלה נמצאת במצב 5NF אם לא ניתן לפרק אותה לטבלאות קטנות יותר ואחר כך לצרף אותה חזרה לטבלה המקורית מבלי לאבד את הנתונים ואת משמעותם. טבלה הנמצאת במצב 5NF נמצאת גם במצב 4NF.

Fagin הגדיר גם מצב DK/NF (Domain Key / Normal Form) והוכיח שלא ייתכנו מבנים ברמות נירמול גבוהות יותר. מכיון שלמבנים אלה אין משמעות מעשית, הם לא מוצגים כאן.

שאלות חזרה ותרגילים

שאלות חזרה

1. מה מטרת תהליך הנירמול? האם תוכל לחשוב על מצבים בהם יש גם חסרונות לתהליך הנירמול? האם יש חשיבות לתהליך זה גם בבואנו לבנות מערכת מידע בסביבת FMS?
2. מדוע לסלק כפילויות מבסיס הנתונים? האם תמיד כדאי לסלק את כל הכפילויות מבסיס הנתונים?
3. הסבר את הרעיון של **התלות הפונקציונלית**. מדוע התלות הפונקציונלית חשובה בהקשר של בניית מודלים של נתונים?
4. האם ניתן להסיק את התלויות הפונקציונליות מתוך התבוננות על שורה בודדת של טבלה?
5. מה ההבדל בין פירוק משמר תלויות לפירוק שאינו משמר את התלויות?
6. קרא את המשפטים הבאים וענה בכן/לא:
   א. אם כל התכונות תלויות פונקציונלית במפתח, אין צורך לבצע פירוק נוסף.
   ב. מצב BCNF משוחרר לחלוטין מעל בעיות ולכן יש תמיד לשאוף למצב זה.
   ג. בסיס נתונים במצב BCNF הינו בעל זמני תגובה טובים יותר מבסיס נתונים לא מנורמל.
   ד. תרשים ישויות-קשרים משמש לאותן המטרות כמו תרשים התלויות הפונקציונליות.
   ה. תרשים תלויות פונקציונליות מאפשר איתור בעיות במבנה הנתונים.

תרגילים

1. נתונה הטבלה הבאה המציגה את יומן הפגישות של סוכני ביטוח עם לקוחותיהם:

מס. סוכן	שם סוכן	מס' לקוח	שם לקוח	תאריך פגישה	שעת פגישה	מספר פוליסה
1245	משה כהן	3452	רון עדי	25.10.99	12:30	P-1554
1245	משה כהן	2544	יעל לוי	25.10.99	20:00	P-1554
3452	דן לוי	9234	דינה זיו	25.10.99	09:00	C-2543
6534	איל בונה	6543	שלי בגיר	26.10.99	08:00	F-6543
3452	דן לוי	3452	רון עדי	26.10.99	13:30	C-2543
6534	איל בונה	2544	יעל לוי	27.10.99	08:00	F-6543

בנה את תרשים התלויות הפונקציונליות הנובע מטבלה זו. השתמש בכללי הנירמול כדי לבנות סכימה טבלאית במצב BCNF.

2. נתונות התלויות הפונקציונליות הבאות:

I. $A \rightarrow R, L, X$

II. $B \rightarrow C, D$

א. $D \rightarrow F$

I. $A, B \rightarrow H, Y$

II. $Y \rightarrow Z$

צייר את תרשים התלויות המתאים. פרק את המבנה לאוסף טבלאות BCNF.

3. חברת התוכנה "ביטביט" מפתחת מספר מוצרי תוכנה. לחברה שלוש מחלקות עיקריות: מחלקת הפיתוח, מחלקת מכירות ומחלקת כספים ואדמיניסטרציה. נתונות התלויות הפונקציונליות הבאות:

א. קוד מחלקה ← שם מחלקה, תקציב, תעודת זהות מנהל המחלקה

ב. תעודת זהות מנהל המחלקה ← שם המחלקה

ג. קוד לקוח ← שם לקוח, כתובת לקוח, מספר טלפון

ד. תעודת זהות ← שם עובד, כתובת עובד, שכר, תאריך תחילת עבודה, קוד מחלקה

ה. תעודת זהות, תאריך ← קוד התמחות

ו. קוד התמחות ← שם התמחות

ז. מס' הזמנה ← שם מוצר תוכנה, תאריך הזמנה, סכום הזמנה, תעודת זהות של איש המכירות, קוד לקוח, גירסה שנקנתה

ח. שם מוצר תוכנה ←← מערכת הפעלה בה פועל, גירסה אחרונה (שים לב, זוהי תלות רב-ערכית).

בנה אוסף טבלאות BCNF מתאים לבעיה זו.

4. נתונות שתי הקבוצות הבאות של תלויות פונקציונליות:

I. $R = \{A \rightarrow C, AC \rightarrow D, E \rightarrow H\}$

II. $S = \{A \rightarrow CD, E \rightarrow AH\}$

השתמש בכללי Armstrong כדי לבדוק אם שתי הקבוצות הללו הן זהות.

5. נתון יחס עם התכונות A, B, C, D, E ו-F. נניח שידועות התלויות הפונקציונליות הבאות המתקיימות ביחס זה:

א. $AB \rightarrow C$

I. $BC \rightarrow AD$

II. $D \rightarrow E$

III. $CF \rightarrow B$

בנה את קבוצת הסגור של $\{A, B\}^+$ כלומר את $\{A, B\}$.

6. נתון היחס הבא המתייחס לעובדים המשמשים כנציגי מכירות וללקוחות שקנו מהם פריט כלשהו:

עובדים (מספר עובד, שם פרטי, שם משפחה, מספר מחלקה, קוד תפקיד, מספר קורס, שם תפקיד, תאריך, שכר, שם ילד, מספר לקוח, שם לקוח, מספר פריט, כמות, תיאור פריט).

נתונות התלויות הפונקציונליות האלו:

א. מספר עובד ← שם פרטי, שם משפחה

ב. מספר עובד ← קוד תפקיד, שם תפקיד

ג. מספר עובד ← מספר קורס

ד. מספר עובד, תאריך ← שכר

ה. מספר עובד, מספר לקוח, מספר פריט ← כמות

ו. מספר פריט ← תיאור פריט

ז. קוד תפקיד ← תיאור תפקיד

ח. מספר לקוח ← שם לקוח

ט. מספר עובד ←← שם ילד

י. מספר עובד ←← תאריך, שכר

העבר את היחס המוצג בתרגיל זה למבנה 4NF וצייר את תרשים הישויות-קשרים המתאים.

7. נתונות התלויות הפונקציונליות הבאות:

I. $A \rightarrow B$

II. $A \rightarrow C$

III. $A \rightarrow\rightarrow D$

IV. $A \rightarrow\rightarrow E$

V. $B \rightarrow L$

VI. $B \rightarrow C$

VII. $F \rightarrow G$

VIII. $H \rightarrow F$

IX. $A, F \rightarrow K$

X. $A, F \rightarrow H$

נתונים היחסים הבאים:

I. R1 (A , B, F , G)

II. R2 (A , B, C)

III. R3 (A , B, D , E)

IV. R4 (A , C, G)

V. R5 (A , F , H, K)

VI. R6 (B , L)

איזה מהיחסים שלמעלה נמצא במצב 4NF על פי התלויות הפונקציונליות הנתונות?

פרק 6

מתודולוגיה לעיצוב בסיס הנתונים (Database Design Methodology)

1. מבוא
2. תיאור כללי של המתודולוגיה
3. המרת מודל תפישתי למודל טבלאי
4. דוגמה לתהליך עיצוב בסיסי נתונים
5. סיכום
6. שאלות חזרה ותרגילים

מבוא

תפקידו של תהליך עיצוב בסיס הנתונים, הוא לבנות ולעצב מודל של המציאות הרלוונטית, שהוא כמובן פישוט של המציאות, ועם זאת הוא מספק את צרכי המידע של הארגון.

תהליך עיצוב בסיס הנתונים מתחיל בעיצוב המודל התפישתי ועובר להמרת המודל שהתקבל למודל טבלאי. תהליך זה הינו מורכב יחסית, ובדרך כלל צורך זמן רב. התהליך מחייב מגע עם המשתמשים, הבנת דרישות המידע, ניסיון לחזות דרישות עתידיות, התייחסות לדרישות משותפות ולדרישות ייחודיות ובחירת חלופות עיצוב מתוך מיגוון אפשרויות.

ניתן לחלק את אוסף המתודולוגיות לבניית מודלים תפישתיים לשתי קטגוריות עיקריות:

- **עיצוב מעלה-מטה** (Top-Down): מתודולוגיות אלו מתחילות מזיהוי קבוצות, זיהוי התכונות של הקבוצות וזיהוי הקשרים ביניהן. בשנים האחרונות שולבו במתודולוגיות אלו גם תהליכים לנירמול הקבוצות לקבלת מבנה נכון ויציב.
- **עיצוב מטה-מעלה** (Bottom-Up): מתודולוגיות אלו מתחילות מזיהוי האלמנטים הבסיסיים ביותר, התכונות, וזיהוי מפתחות. אחר כך הן בונות את המודל התפישתי תוך שימוש בטכניקות של ניתוח תלויות, כלומר הקבצת תכונות סביב המפתח והגדרת קבוצה מול כל מפתח.

המתודולוגיה שתוצג בהמשך, שואבת את רעיונותיה משתי הקטגוריות של המתודולוגיות לעיצוב המודל התפישתי גם יחד. היתרונות העיקריים של מתודולוגיה זו הם היכולת לאתר במהירות יחסית את קבוצות הישות העיקריות, לנתח את הקשרים ביניהן – ולאחר מכן לעדכן את המודל המתקבל באופן איטרטיבי עד לקבלת רמת הפירוט המבוקשת. המתודולוגיה משתמשת בכלים גרפיים להצגת המודל התפישתי.

עיצוב בסיסי הנתונים אינו מדע מדויק ולכן ניתן לראות את המתודולוגיה כאוסף של כללים מנחים, ולא כאלגוריתם מוגדר היטב. תהליך העיצוב מחייב הבנה של הסביבה והיישום, יכולת הפשטה ואינטרפרטציה של העולם הממשי עד לבניית המודל המתאים.

הניסיון המקצועי והאינטואיציה של מנתח המערכת ומעצב בסיס הנתונים, עדיין מהווים גורם חשוב, למרות הרצון להתייחס לתהליך העיצוב כאל תהליך מובנה ומדויק. מכאן ברור, שמטרת המתודולוגיה המוצגת היא להוות מעין מסגרת **ורשימת תיוג** (Checklist) אשר תבטיח טיפול מלא ועקבי בכל ההיבטים עד לקבלת מבנה נכון ויציב.

בפרק זה

- נסקור את השלבים העיקריים של המתודולוגיה.
- נסקור כיצד ניתן לתרגם את המודל התפישתי, מודל ישויות-קשרים לאוסף טבלאות.
- נציג דוגמה מפורטת של תהליך עיצוב בסיס נתונים טבלאי.

תיאור כללי של המתודולוגיה

המתודולוגיה מחולקת לשלושה שלבים עיקריים:

שלב א' – עיצוב מודל-על ראשוני

בשלב זה המעצב סוקר את כל החומר הרלוונטי ומזהה את קבוצות הישות העיקריות, המפתחות שלהן, התכונות העיקריות והקשרים העיקריים. מודל העל המתקבל מאפשר קבלת תחושה מהירה "מי הם השחקנים ומה גבולות המגרש". תחושה זו חשובה בעיקר בבעיות אמיתיות שבהן המעצב נתקל במושגים שאינם ברורים לו מלכתחילה והוא בונה את הידע וההבנה שלו באופן הדרגתי. מודל-על זה מהווה את השלד הראשוני של **המודל התפישתי הגלובלי** – **GV** (Global View).

שלב ב' – ניתוח נקודות מבט מקומיות ושילובן במודל הגלובלי

שלב זה הינו איטרטיבי במהותו ובמסגרתו מתבצע ניתוח פרטני של כל **מבט מקומי** – **LUV** (Local User View). כל מבט מקומי נגזר מתוך מסמך ניתוח המערכת (כגון DFD) והוא מייצג את הנתונים הדרושים לביצוע תהליך כלשהו (טופס דוח, תהליך ארגוני וכד'). לאחר הבנת כל פרטי המידע, התלויות, הקשרים והאילוצים, מתבצע **תהליך שילוב מבט מקומי** (View Integration) אל תוך מודל העל הכולל, כלומר **המודל התפישתי הגלובלי**.

שלב ג' – המרת המודל התפישתי למודל טבלאי

שלב זה עוסק בהמרת מודל הישויות-קשרים שהתקבל לאוסף מקביל של טבלאות. כל הטבלאות המתקבלות הן במצב BCNF המבטיח כפילות נתונים מינימלית.

תרשים 6.1: שלבים עיקריים במתודולוגיית עיצוב בסיס הנתונים

התרשים 6.2 מציג בפירוט את שלושת שלבי המתודולוגיה.

תיאור מפורט של המתודולוגיה

שלב א': עיצוב מודל-על ראשוני

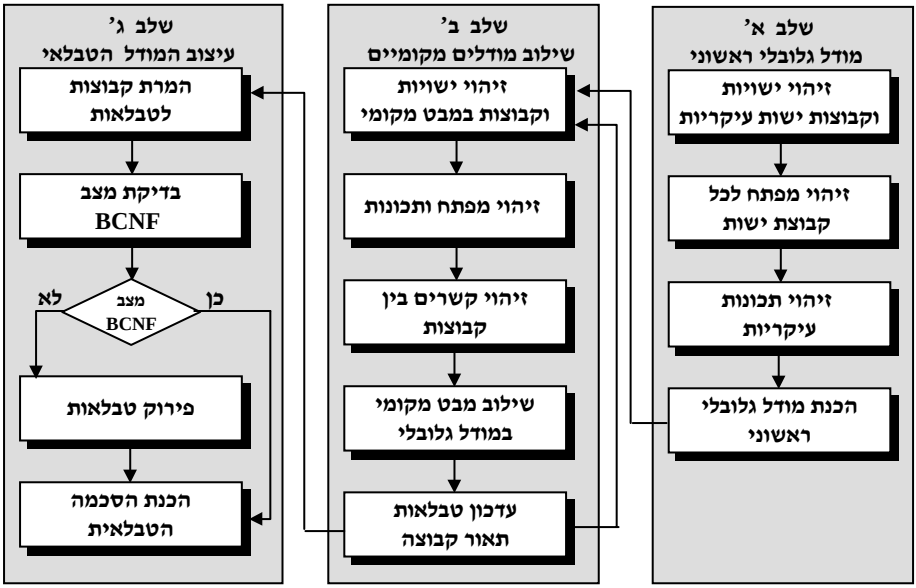
א. סקירה של כל החומר הרלוונטי (מסמכי דרישות, תרשימי DFD וכד') וזיהוי הישויות העיקריות. בשלב זה לא ניתן לאתר את הקשרים בין הישויות, וגם לא ניתן לזהות את סוג הקשר.

ב. סיווג הישויות לקבוצות ישות.

ג. זיהוי המפתח והתכונות העיקריות של כל קבוצת ישות. עדיין לא ניתן לזהות את כל התכונות של כל קבוצה.

ד. הכנה של טבלת תיאור קבוצה, עבור כל קבוצה שמזוהה ורישום של התכונות, המפתח וכל אילוץ מיוחד שזוהה כבר בשלב זה.

ה. הכנת תרשים של קבוצות הישות העיקריות ורישום המפתחות שלהן. תרשים זה יהיה המודל הגלובלי הראשוני (Initial Global View) ויעודכן באופן איטרטיבי בשלב ב'.



תרשים 6.2: תיאור שלבי המתודולוגיה.

שלב ב': ניתוח נקודות מבט מקומיות

בשלב זה עובר מעצב בסיס הנתונים על **כל** דרישות המידע במסמכי הניתוח. דרישת מידע יכולה להיות מאגר לוגי בתרשים DFD, מבנה של דוח, טופס קלט וכד'. כל דרישת מידע כזאת מוגדרת כנקודת מבט מקומית – LUV (Local User View). עבור כל נקודת מבט מקומית, יש לבצע את הצעדים הבאים:

א. זיהוי הישויות וסיווגן לקבוצות או תת-קבוצות.

ב. זיהוי התכונות השייכות לכל קבוצה.

ג. זיהוי המפתח העיקרי של הקבוצה.

ד. זיהוי הקשרים של הקבוצה לקבוצות אחרות. יש לזהות בכל קשר את הקרדינליות שלו וסוג המיפוי.

ה. שילוב המבט המקומי המנורמל במודל הגלובלי – GV. את המודל הגלובלי יש לנהל בצורה גרפית, בתרשים ישויות-קשרים. יש לרשום על גבי התרשים את אילוצי הקרדינליות הנובעים מהמבט המקומי ששולב. במהלך השילוב יש לפתור בעיות של סתירות (לדוגמה, במבט מקומי אחד מכנים את המונח "מספר קורס" ובמבט מקומי אחר מכנים את המונח "קוד קורס").

ו. הקמה או עדכון של טבלת תיאור קבוצה לכל קבוצה. יש לרשום בטבלה זו את כל האילוצים ואת מרחבי הערכים הידועים.

שלב ג': המרת המודל התפישתי למודל טבלאי

בשלב זה מתבצע תהליך המרת המודל התפישתי למודל הטבלאי. כל קבוצת ישות עוברת המרה לטבלה, מתבצעת בדיקה שכל הקשרים הקיימים במודל התפישתי באים לידי ביטוי במודל הטבלאי ומתבצעת בדיקה שהטבלאות המתקבלות הן במצב BCNF. אם לא, מתבצע תהליך של פירוק הטבלה לאוסף מקביל של טבלאות עד שכל אחת מהן תהיה במצב BCNF. התהליך מורכב מהפעולות הבאות:

א. המרת כל קבוצת ישות מקבילה. ההמרה תתבצע על פי הכללים שנציג מייד בהמשך. יש לוודא שכל קשר המופיע במודל התפישתי יתורגם לאוסף מתאים של מפתחות עיקריים ומפתחות זרים.

ב. מעבר שיטתי על טבלה שהתקבלה. בדיקה אם הטבלה במצב 1NF. אם הטבלה אינה במצב 1NF, יש לסלק ממנה את העמודות מרובות הערכים על ידי פירוק הטבלה למספר טבלאות.

ג. ניתוח תלויות של עמודות הטבלה והעברתה למצב BCNF. אם הטבלה אינה במצב BCNF, יש לפרק אותה למספר טבלאות בהתאם לכללי הנירמול.

המרת מודל תפישתי למודל טבלאי

מודל הנתונים התפישתי מיוצג על ידי **תרשים ישויות-קשרים** מתאים. בסעיף זה נעסוק בכללים להמרת המודל התפישתי למודל טבלאי. נסקור את אוסף הכללים המשמשים להמרת מודל הנתונים התפישתי למודל הנתונים הטבלאי.

המרת קבוצת ישות רגילה

- לכל קבוצת ישות רגילה יש ליצור טבלה אחת.
- עבור כל תכונה של הקבוצה תוגדר עמודה בטבלה.
- אם לקבוצה יש תכונה מורכבת, יש להחליט האם חשוב לנהל בטבלה עמודה אחת המכילה את כל המרכיבים או שיש לנהל עמודה נפרדת לכל אחד מהמרכיבים של התכונה. למשל, את התכונה כתובת הסטודנט ניתן לנהל כעמודה אחת או שניתן לנהל אותה במספר עמודות שונות כגון – עיר, רחוב, מספר בית, מיקוד. החלטה זו תלויה בשימוש הצפוי.
- אם לקבוצה יש מספר מפתחות אפשריים, יש לבחור באחד מהם כמפתח עיקרי. הבחירה מבוססת בדרך כלל על שיקולים כגון יציבות ואורך העמודה.

נציג דוגמה להמרה של קבוצת ישות סטודנטים:

סטודנטים (מספר סטודנט, שם סטודנט, עיר, שם רחוב, מספר בית, מיקוד, תאריך לידה, מספר זהות)

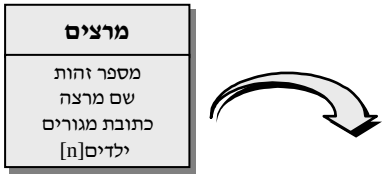
תרשים 6.3: המרת קבוצת ישות פשוטה לטבלה.

נשים לב שבדוגמה זו בחרנו לפרק את התכונה המורכבת כתובת מגורים לעמודות נפרדות ובחרנו את מספר הסטודנט כמפתח עיקרי, למרות שגם מספר הזהות הוא מפתח עיקרי.

המרת קבוצת ישות עם תכונה מרובת מופעים

אם לקבוצה יש תכונה מרובת ערכים, יש לבנות טבלה חדשה שתכיל מפתח עיקרי מורכב הבנוי מהמפתח העיקרי של קבוצת הישות העיקרית יחד עם התכונה שחוזרת על עצמה.

מהדוגמה בתרשים 6.4 ניתן לראות שהתכונה מרובת המופעים ילדים הועברה לטבלה נפרדת – טבלת הילדים של המרצים. לטבלה חדשה זו נקבע מפתח מורכב המכיל את מספר הזהות של המרצה ומספר הזהות של הילד.



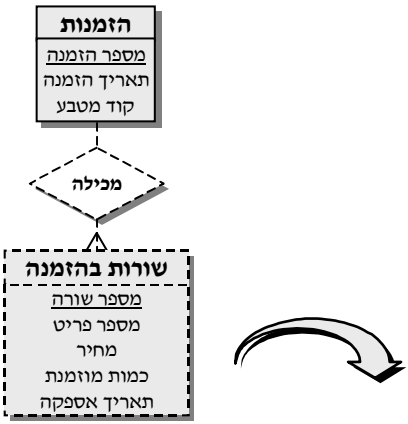
מרצים (מספר זהות, שם מרצה, כתובת מגורים)
ילדים של מרצה (מספר זהות מרצה, מספר זהות ילד, שם ילד, תאריך לידה)

תרשים 6.4: המרת תכונה מרובת מופעים.

המרת קבוצת ישות חלשה

- לכל קבוצת ישות חלשה יש ליצור טבלה.
- הטיפול בתכונות בדומה לקבוצת ישות רגילה.
- המפתח העיקרי של הטבלה יורכב מהמפתח של טבלת האב והמפתח של טבלת הבן.

נציג דוגמה להמרת קבוצת ישות חלשה:



הזמנות (מספר הזמנה, תאריך הזמנה, קוד מטבע)
ילדים של מרצה (מספר הזמנה, מספר שורה, מספר פריט, כמות מוזמנת, תאריך אספקה)

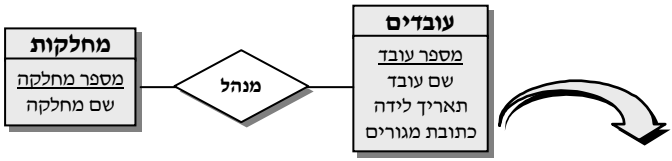
תרשים 6.5: המרת קבוצת ישות חלשה.

נשים לב שהמפתח העיקרי של טבלת הזמנות נגרר אל הישות החלשה והפך לחלק מהמפתח העיקרי של טבלת שורות בהזמנה, שהיא ישות חלשה.

המרת קשר חד-חד-ערכי

אם קיים קשר חד-חד-ערכי בין קבוצת ישות E_1 וקבוצת ישות E_2, נבחר את אחת מהטבלאות, למשל E_2, ונוסיף לה כמפתח זר את המפתח העיקרי של הטבלה המייצגת את קבוצה E_1.

לדוגמה, נניח שבמודל התפישתי שתי קבוצות ישות, עובדים ומחלקות וביניהן קשר חד-חד-ערכי, בעל משמעות מנהל המחלקה.



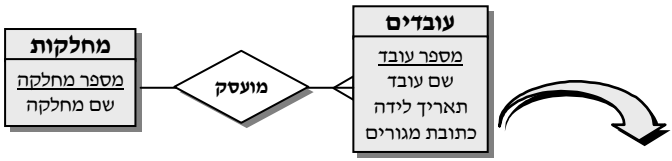
עובדים (מספר עובד, שם עובד, תאריך לידה, כתובת מגורים)
מחלקות (מספר מחלקה, שם מחלקה, מספר עובד של מנהל מחלקה)

תרשים 6.6: קשר חד-חד-ערכי בין שתי קבוצות.

נשים לב שבמקרה זה בחרנו להוסיף את העמודה מספר עובד של מנהל המחלקה לטבלת המחלקות כדי לממש את הקשר החד-חד-ערכי. הקורא שואל את עצמו מדוע לא הוספנו גם את מספר המחלקה בטבלת עובדים. ובכן, כפי שנראה בהמשך, שפת SQL תאפשר לנו בקלות למצוא מי המנהל של כל מחלקה ללא צורך להוספת עמודה נוספת זו.

המרת קשר חד-רב-ערכי

אם קיים קשר חד-רב-ערכי בין קבוצת ישות E_1 אל קבוצת ישות E_2, נוסיף לטבלה המייצגת את הקבוצה E_2 מפתח זר, שהוא המפתח העיקרי של הטבלה המייצגת את הקבוצה E_1.



עובדים (מספר עובד, שם עובד, תאריך לידה, כתובת מגורים, מספר מחלקה)
מחלקות (מספר מחלקה , שם מחלקה)

תרשים 6.7: קשר חד-רב-ערכי בין שתי קבוצות.

ניתן לראות שהוספנו את המפתח העיקרי מטבלת המחלקות כמפתח זר לטבלת העובדים.

קשר רב-רב-ערכי

אם קיים קשר רב-רב-ערכי בין קבוצת ישות E_1 עם קבוצת ישות E_2, נוסיף טבלה חדשה שיוצרת את הקשר בין שתי הקבוצות. המפתח העיקרי של הטבלה החדשה יהיה מורכב מהמפתח העיקרי של הטבלה המייצגת את הקבוצה E_1 והמפתח העיקרי של הטבלה המייצגת את הקבוצה E_2. שיטה זו תופעל הן במקרה שהקשר נושא מידע או שהוא אינו נושא מידע.

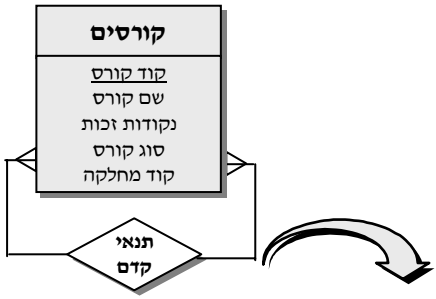
מרצים (מספר זהות, שם מרצה, כתובת מגורים)
קורסים (קוד קורס, שם קורס, נקודות זכות, סוג קורס, קוד מחלקה)
מרצה בקורס (מספר זהות , קוד קורס , סוג מרצה, שעות שבועיות)

תרשים 6.8: קשר רב-רב-ערכי בין שתי קבוצות.

בדוגמה זו, בנינו טבלה מרצה בקורס הקושרת בין שתי הקבוצות של מרצים וקורסים. המפתח העיקרי של טבלת מרצה בקורס הוא מפתח מורכב המכיל שני מפתחות זרים.

המרת קשר רב-רב-ערכי רפלקסיבי

אם קיים קשר רב-רב-ערכי בין ישויות של אותה קבוצה, למשל E_1, נבנה טבלה חדשה שתכיל את המפתח העיקרי של הקבוצה בשני תפקידים שונים.



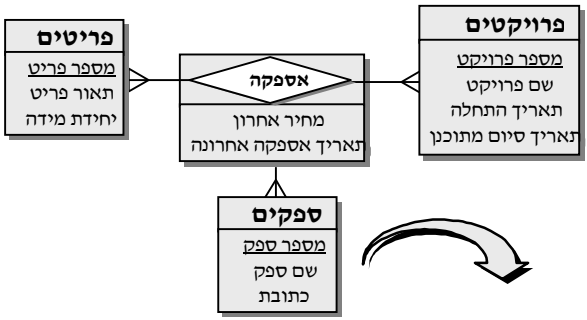
קורסים (קוד קורס, שם קורס, נקודות זכות, סוג קורס, קוד מחלקה)
תנאי קדם לקורס (קוד קורס ראשי , קוד קורס קדם , ציון מינימלי)

תרשים 6.9: קשר רב-רב-ערכי רפלקסיבי.

בדוגמה זו בנינו טבלה חדשה, תנאי קדם לקורס, המכילה פעמיים את קוד הקורס: פעם קוד הקורס הראשי ופעם שנייה קוד הקורס שמהווה תנאי קדם לקורס הראשי. כשצריך ניתן לנהל בטבלת תנאי קדם לקורס את הציון המינימלי שעל סטודנט לקבל בקורס, המהווה תנאי קדם להרשמה לקורס. כך ניתן לבטא כל קשר רפלקסיבי בגמישות רבה.

המרת קשר רב-רב-ערכי בדרגה גבוהה

אם קיים קשר רב-רב-ערכי בדרגה גבוהה בין מספר קבוצות ישות, יש לבנות טבלה שתכיל את נתוני הקשר. המפתח של טבלה זו יהיה הצירוף של המפתחות העיקריים של כל קבוצות הישות המשתתפות בקשר.



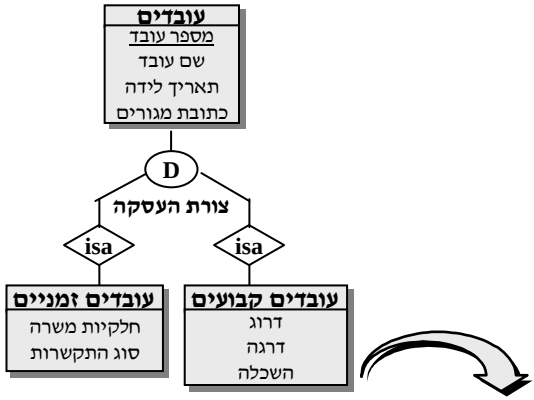
פריטים (מספר זהות, תאור פריט, יחידת מידה)
פרויקטים (מספר פרויקט, שם פרויקט, תאריך התחלה, תאריך סיום מתוכנן)
ספקים (מספר ספק, שם ספק, כתובת)
אספקה (מספר ספק, מספר פריט, מספר פרויקט, מחיר אחרון, תאריך אספקה אחרונה)

תרשים 6.10: קשר רב-רב-ערכי בדרגה גבוהה.

המרת היררכיה סמנטית

להיררכיות סמנטיות אין ייצוג ישיר במודל הטבלאי ולכן יש להפוך גם אותן לאוסף של טבלאות. כמובן שבצורה זו, העובדה שאנו מכניסים קשר **isa** לסכימה אינה מקבל ביטוי כלשהו וזו אחריות המשתמש בבסיס הנתונים להבין שהוא מטפל בהיררכיה סמנטית.

כפי שניתן לראות בתרשים 6.11, ההיררכיה הסמנטית תורגמה לשלוש טבלאות, כל אחת עם הערכים המיוחדים שלה. לכל הטבלאות אותו מפתח, מספר עובד. מאחר ואין תמיכה בהיררכיה סמנטית, גם אין כאן הורשה של תכונות מקבוצת העל, עובדים, אל שתי תת-הקבוצות, עובדים קבועים ועובדים זמניים. התכונה המגדירה צורת העסקה מופיעה בטבלת עובדים ומאפשרת להחליט באיזו משתי הטבלאות האחרות נמצאים שאר הנתונים של העובד. הפתרון לכך הוא שימוש במנגנון הטבלה המדומה (View) של מערכות RDBMS כדי לקבל לכל אחד מסוגי העובדים את כל התכונות.



עובדים (מספר עובד, שם עובד, תאריך לידה, כתובת מגורים, צורת העסקה)
עובדים קבועים (מספר עובד, דרוג, דרגה, השכלה)
עובדים זמניים (מספר עובד, חלקיות משרה, סוג התקשרות)

תרשים 6.11: היררכיה סמנטית.

נשים לב שניתן לייצג את ההיררכיה הסמנטית בצורות נוספות: למשל, להחליט שכל קבוצות הישות בהיררכיה תיוצגנה על ידי טבלה אחת המכילה את כל התכונות של כל הקבוצות או לחילופין לייצג את ההיררכיה בשתי טבלאות. אם למשל רוב העובדים במפעל הם קבועים, נוכל לבנות טבלה אחת עבור עובדים קבועים המכילה את התכונות של שתי הקבוצות ועוד טבלה אחת עבור היוצאים מן הכלל, עובדים זמניים. בצורה זו, עבור רוב הגישות לטבלת עובדים קבועים, ניתן לקבל את כל נתוני העובד מטבלה אחת ואין צורך לטפל בשתי טבלאות.

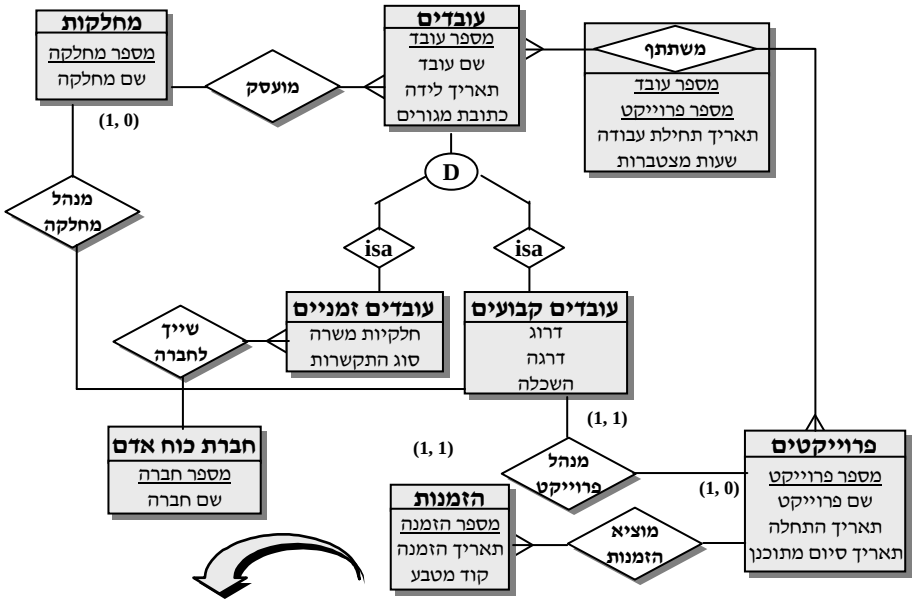
דוגמה להמרת מודל תפישתי למודל טבלאי

לסיכום נושא זה, נציג תרשים ישויות-קשרים כלשהו, ונתרגם אותו לסכימה הטבלאית המתאימה.

תרשים 6.12 מציג את הסכימה הטבלאית המתאימה למודל הישויות-קשרים. כפי שניתן לראות, למודל התפישתי יש יכולת ביטוי רבה יותר בגלל היותו גרפי, ובגלל תמיכתו הישירה בהיררכיות סמנטיות. לעומתו המודל הטבלאי הוא פשוט יותר, מאחר וכל הנתונים מיוצגים על ידי טבלאות פשוטות.

דוגמה לתהליך עיצוב בסיסי נתונים

נשתמש בדוגמה פשוטה כדי להציג את השימוש במתודולוגיה המשולבת לעיצוב בסיס הנתונים. בדוגמה נקים בסיס נתונים לניהול פעילות מכללה. סיפור המעשה דומה לסיפור המעשה המופיע בפרק המבוא, אולם הוא מפורט יותר ומכיל מספר שינויים שנועדו להדגים את המתודולוגיה.



עובדים (מספר עובד, שם עובד, תאריך לידה, כתובת מגורים, צורת העסקה)
עובדים קבועים (מספר עובד, דרוג, דרגה, השכלה)
עובדים זמניים (מספר עובד, חלקיות משרה, סוג התקשרות, מספר חברה)
חברות כוח אדם (מספר חברה, שם חברה)
מחלקות (מספר מחלקה, שם מחלקה, מספר עובד מנהל)
פרויקטים (מספר פרויקט, שם פרויקט, תאריך התחלה, תאריך סיום, מספר עובד מנהל)
משתתף (מספר עובד , מספר פרויקט, תאריך התחלת עבודה, שעות מצטברות)
הזמנות (מספר הזמנה, תאריך הזמנה, קוד מטבע, מספר פרויקט)

תרשים 6.12: תרשים ישויות-קשרים והמודל הטבלאי המתקבל.

רקע כללי ותיאור האירוע

המכללה לישראל החליטה למחשב את כל תהליכי המנהל האקדמי כדי לשפר ולייעל את השירות הניתן לסטודנטים וליחידות הארגוניות השונות. בהמשך מופיע תיאור כללי של תהליכי העבודה ורשימה של דרישות מידע מהמערכת החדשה.

<u>סיפור המעשה – "מכללת ישראל" המכללה ללימודים מתקדמים</u>

"מכללת ישראל" היא מכללה חדשה המשמשת כמוסד הוראה ללימודים אקדמיים. במכללה יש מספר מחלקות אקדמיות הרשאיות להעניק תארים אקדמיים, כמו למשל המחלקה למנהל עסקים, המחלקה למשפטים, המחלקה לכלכלה, המחלקה למדעי המחשב, המחלקה למתמטיקה וכד'. בראש כל מחלקה עומד ראש מחלקה שהינו אחד מחברי הסגל האקדמי. ראש המחלקה יכול להתחלף מעת לעת.

כל מחלקה מציעה מספר חוגים. למשל המחלקה למדעי המחשב מציעה את החוגים להנדסת מחשבים, הנדסת תוכנה, מערכות מידע ותיב"מ. כל חוג מוצע על ידי מחלקה אחת בלבד. כל מחלקה מזוהה על ידי קוד של שני תווים (למשל MT – מתמטיקה, CS – מדעי המחשב, HS – היסטוריה וכד'). כל מחלקה אקדמית רשאית להעניק תארים אקדמיים – תואר ראשון או תואר שני. אנשי הסגל האקדמי מלמדים בקורסים השונים. כל איש סגל אקדמי שייך למחלקה אקדמית כלשהי. איש סגל אקדמי יכול לעבוד במהלך תקופת עבודתו במספר מחלקות, אולם לא בו-זמנית.

סטודנט הלומד במכללה חייב להירשם לחוג ראשי כלשהו ולמספר חוגים משניים, גם כאלה הניתנים על ידי מחלקות שונות מהמחלקה המציעה את החוג הראשי.

כל סטודנט הרוצה להירשם לקורסים הניתנים על ידי חוג, חייב להיעזר ביועץ אקדמי של החוג הראשי אליו הוא נרשם. היועץ מאשר את תוכנית הלימודים של הסטודנט בכל סמסטר וחותם על גבי טופס הרישום לקורסים. היועץ הינו מרצה במחלקה המסוימת.

כל חוג מציע מיגוון קורסים. כל קורס מזוהה על ידי קוד בן חמישה תווים, שנים המציינים את קוד המחלקה ושלושה מציינים מספר רץ (למשל החוג להנדסת מחשבים מציעה את הקורסים CS102 – מבוא למחשבים, CS304 – שפת פסקל, CS223 – מערכות הפעלה ועוד). כל קורס מקנה מספר מסוים של נקודות זכות לסטודנט המסיים אותו בהצלחה.

המכללה פועלת בסמסטרים. כל סמסטר מזוהה על ידי קוד המציין את מספר הסמסטר והשנה הקלנדרית (למשל 1/1999 מציין סמסטר ראשון של שנת 1999). בכל סמסטר מציע כל חוג מיגוון חלקי בלבד מבין כל הקורסים שלו.

כל קורס ניתן במסגרת קבוצות לימוד. קורסים מבוקשים ניתנים במספר קבוצות לימוד והפחות מבוקשים ניתנים במסגרת קבוצת לימוד אחת בלבד. כל קבוצת לימוד מזוהה על ידי תו אחד. זיהוי מלא של קבוצת לימוד יהיה צירוף של קוד הקורס, קוד הסמסטר וקוד קבוצת הלימוד (למשל, CS102-2/1999-B מציין את קבוצת הלימוד השנייה של קורס "מבוא למחשבים" הניתן בסמסטר שני של שנת 1999). כל קבוצת לימוד ניתנת בימים ושעות מסוימות על ידי מרצה מסוים. למשל קבוצת הלימוד CS102-2/1999-B נפגשת ביום א' בין השעות 10:00 ל 12:00 וביום ג' בין השעות 13:00 ל 14:00.

בזמן תהליך ההרשמה, על הסטודנט לציין את הקוד המלא של קבוצת הלימוד אליה הוא רוצה להירשם. מאחר וייתכן שלא ניתן לשבץ את הסטודנט לקבוצה המבוקשת, עליו לציין גם עדיפויות משניות. בגמר תהליך ההרשמה והשיבוץ לקבוצות לימוד, מקבל כל מרצה את רשימת הסטודנטים ששובצה לקבוצת הלימוד בה הוא מלמד. על גבי רשימה זו חייב המרצה לרשום את הציונים שכל סטודנט קיבל בקורס. בגמר הסמסטר חייב המרצה להחזיר את הטופס למזכירות לרישום הציונים.

בגמר קליטת הציונים של כל הסמסטר, מפיקים דוח הישגים מצטבר לכל סטודנט. דוח זה מציג את ציוניו של הסטודנט בכל הקורסים בהם השתתף ואת סה"כ נקודות הזכות שלו.

המכללה מפעילה מספר מעונות סטודנטים ברחבי העיר. בכל מעון מספר חדרים מסוגים שונים (ליחיד ולזוג). מדי שנה מתבצע רישום ושיבוץ של הסטודנטים למעונות.

דרישות מידע

נסקור עכשיו את כל דרישות המידע כפי שהן מוצגות למעצב בסיס הנתונים.

דרישה א': אלפון סטודנטים

דוח המציג את רשימת הסטודנטים במכללה, ממוין לפי שם הסטודנט.

אלפון סטודנטים

שם סטודנט	מספר סטודנט	כתובת ראשית	כתובת משנית
לאור דוד	12345	הרצל 5, רמת גן	
שמשוני רן	55435	אלנבי 121, חיפה	טגור 15, תל אביב

תרשים 6.13: דוח אלפון סטודנטים.

דרישה ב': דוח הישגים מצטבר לסטודנט

הדוח המוצג בתרשים 6.14 מציג לכל סטודנט את כל ההישגים במהלך כל שנות לימודיו במכללה. לכל סמסטר מוצגת רשימת כל הקורסים בהם הסטודנט למד, הציון שקיבל ומספר הנקודות המשוקללות לתואר (מכפלה של מספר נקודות הזכות בכל קורס והציון).

דרישה ג': שאילתת מרצים לפי מחלקות

שאילתה זו מציגה את רשימת המרצים השייכים למחלקה מסוימת. הנתונים המוצגים בה: קוד המחלקה, שם המחלקה, שם ראש המחלקה, מספר המרצה, שם המרצה, כתובת פרטית, כתובת המשרד במכללה, מספר טלפון חיצוני, מספר טלפון פנימי, דרגה נוכחית, תאריך תחילת עבודה במחלקה (מרצה יכול לעבוד במשך הזמן במספר מחלקות שונות), דרגה התחלתית במחלקה.

שם סטודנט: לאור דוד
מספר סטודנט: 12345
עיר מגורים ראשית: רמת גן

רישום לחוגים

חוג ראשי: מערכות מידע מחלקה: מדעי המחשב תאריך הרשמה: 10.08.1998
חוג משני: חשבונאות מחלקה: מנהל עסקים תאריך הרשמה: 15.02.1998
חוג משני: מחלקה: תאריך הרשמה:

פירוט הישיגם מצטבר

סמסטר	קוד קורס	קב'	שם קורס	שם מחלקה	נק' זכות	ציון	נק' משוקלל
2/1998	CS102	ב	מבוא למחשבים	מדעי המחשב	4	90	3.6
	CS304	א	תכנות Java	מדעי המחשב	3	80	2.4
	MT045	ג	משוואות דיפ'	מתמטיקה	4	90	3.6
1/1999	BS205	א	חשבונאות א'	מנהל עסקים	3	70	2.1
	BS310	א	מבוא למימון	מנהל עסקים	3	90	2.7
	CS223	ב	מערכות הפעלה	מדעי המחשב	4	80	3.2

סה"כ נקודות זכות: 21 סה"כ נקודות משוקללות: 17.5

תרשים 6.14: דוח הישגים מצטבר.

דרישה ד': רשימת יועצים לסמסטר

הרשימה המוצגת בתרשים 6.15 מתפרסמת מדי סמסטר כדי לאפשר לסטודנטים לפנות ליועץ המתאים לכל חוג. לכל חוג יש מספר חברי סגל אקדמי המשמשים כיועצים בסמסטר מסוים.

דרישה ה': לוח זמנים לקבוצות לימוד

לוח הזמנים המוצג בתרשים 6.16 מתפרסם מדי סמסטר על לוחות המודעות ובידיעונים, ומציג את רשימת קבוצות הלימוד המוצעות לאותו סמסטר תוך פירוט הימים, שעות המפגש, הכתה שבה נפגשים וכן את שם המרצה של קבוצת הלימוד.

דרישה ו': אלפון קורסים

תרשים 6.17 מציג את רשימת הקורסים המוצעים על ידי כל חוג, את מספר נקודות הזכות שכל קורס מקנה ואת רשימת הקורסים המהווים תנאי קדם לקורס.

רשימת יועצים לסמסטר1/1999

שם מחלקה	קוד	שם חוג	שם יועץ	טל' פנימי
מדעי המחשב	CS	הנדסת מחשבים	ד"ר דוד גולן	6442
		הנדסת תוכנה	פרופ' זאב הירש	6123
		הנדסת תוכנה	ד"ר דן שמואלי	6253
		מערכות מידע	ד"ר אבי רוזן	6452
		תי"במ	פרופ' אלונה שמעוני	6255
מנהל עסקים	BS	חשבונאות	ד"ר דינה ירון	3224
		חשבונאות	פרופ' רות גביש	3542
		שיווק	ד"ר חיים לוי	3441
		מימון	פרופ' רפי לייבוביץ	3165

תרשים 6.15: רשימת יועצים לסמסטר.

לו" ז קבוצות לימוד לסמסטר 1/1999
המחלקה למדעי המחשב -החוג להנדסת תוכנה

שם קורס	קוד קורס	קבוצה	יום	משעה עד שעה	חדר בנין	שם מרצה
מבוא למחשבים	CS102	א	ב	10:00 12:00	204 שפירא	פרופ' זאב הירש
			ד	15:00 16:00	212 מעבדה א'	
תכנות Java	CS304	א	ה	17:00 19:00	301 מעבדה ג'	ד"ר דן שמואלי
				08:00 09:00	301 מעבדה ג'	
		ב	ג	10:00 13:00	310 וולפסון	ד"ר שמואל בלנק

תרשים 6.16: לוח זמנים לקבוצות לימוד.

דרישה ז': טופס בקשה לרישום לקבוצת לימוד

תרשים 6.18 מציג דוגמה של הטופס המשמש את הסטודנט לרישום קבוצות הלימוד, אליהן ברצונו להירשם לסמסטר הבא. בכל קורס עליו לרשום גם את קבוצת הלימוד בעדיפות שנייה (במידה ויש יותר מקבוצת לימוד אחת), וזאת כדי לאפשר למזכירות החוג לשבץ את הסטודנט לקבוצות בהתאם להעדפותיו. הטופס חייב להיות חתום על ידי היועץ של החוג הנותן את הקורס. על מערכת המידע לאחסן טופס זה בבסיס הנתונים כדי לאפשר בירורים ושינויים.

רשימת קורסים

המחלקה למדעי המחשב - החוג להנדסת תוכנה

סמסטר1/1999

שם קורס	קוד קורס	נק' זכות	תנאי קדם	רשות חובה	סוג קורס
מבוא למחשבים	CS102	3		חובה	שיעור
בסיסי נתונים	CS215	3	מבוא למחשבים	חובה	שיעור
מבני נתונים	CS227	2	מבוא למחשבים	חובה	שיעור
שפת SQL	CS445	2	בסיסי נתונים מבני נתונים	רשות	מעבדה
תכנות Java	CS304	2	מבוא למחשבים	רשות	רשיעור
Unix	CS620	3	מבוא למחשבים מבני נתונים שפת C++	חובה	סמינר

תרשים 6.17: אלפון קורסים.

דרישה ח': רשימת ראשי מחלקות

שאילתה זו תאפשר הצגה של רשימת ראשי מחלקה מסוימת. לכל מחלקה יש להציג את שם המחלקה, שם ראש המחלקה, תאריך תחילת מינוי, תאריך סיום מינוי.

דרישה ט': דוח שיבוץ סטודנטים למעון

דוח זה מציג את השיבוץ של הסטודנטים לחדרים במעון. הנתונים המוצגים בדוח הם: קוד מעון, שם מעון, מספר טלפון, מספר חדר, סוג חדר, מספר סטודנט, שם סטודנט, סמסטר, שנה, תעריף.

טופס הרשמה לקבוצת לימוד

שם סטודנט: _____

מספר סטודנט: _____

סמסטר: _____

	שם קורס	קוד קורס	קבוצה עדיפות א'	קבוצה עדיפות ב'
1.				
2.				
3.				
4.				
5.				
6.				
7.				
8.				
9.				

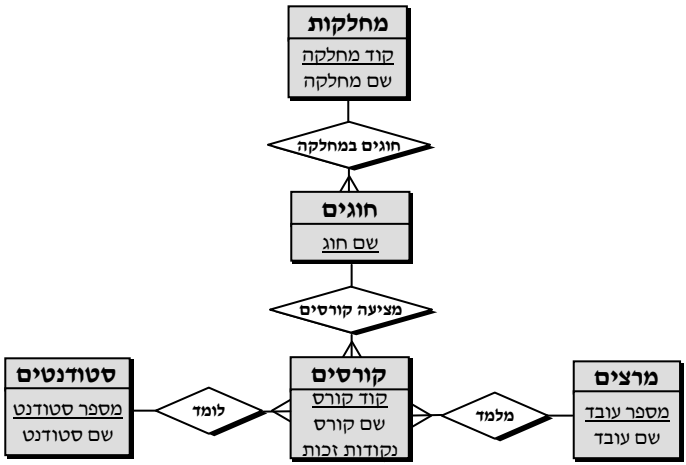
שם יועץ: _____ **תאריך יעוץ:** _____ **חתימה:** _____

תרשים 6.18: טופס בקשה לרישום.

תהליך עיצוב בסיס הנתונים

שלב א': מודל-על ראשוני

בשלב זה נגדיר הגדרה ראשונית של קבוצות הישות העיקריות המשתתפות ביישום. ברור שלא ניתן לזהות בשלב זה את כל הקבוצות והקשרים ביניהן. זיהוי זה יבוא באופן איטרטיבי תוך כדי ניתוח דרישות המידע. לכל קבוצת ישות נגדיר כבר בשלב זה את המפתח העיקרי. מתוך קריאת וסקירת דרישות המידע השונות נקבל מודל-על (מודל גלובלי) ראשוני:



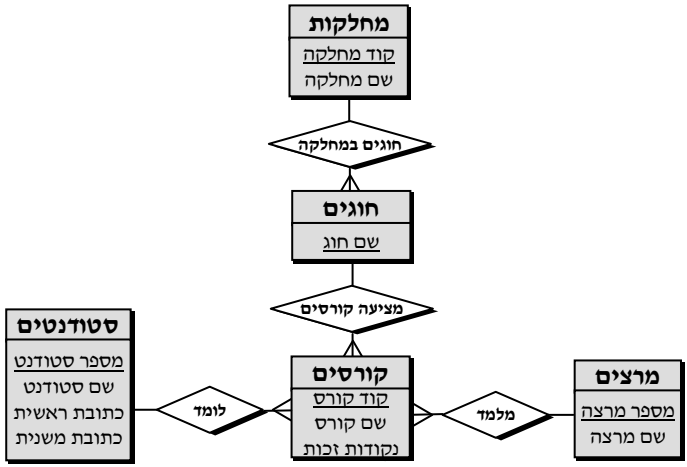
תרשים 6.19: מודל-על ראשוני.

שלב ב': שילוב מודלים מקומיים

בשלב זה ננתח כל אחת מדרישות המידע הפרטניות וכן נעבור מחדש על מסמך תיאור האירוע כדי לוודא שכל המידע המובא בו נלקח בחשבון.

דרישה א' – אלפון הסטודנטים

דרישה זו מוסיפה מספר תכונות לקבוצת סטודנטים. בשלב זה, לא נפרק את הכתובת של הסטודנט למרכיביה (עיר, רחוב, מספר וכד'). נקבל מודל גלובלי מעודכן, כמפורט בתרשים 6.20.

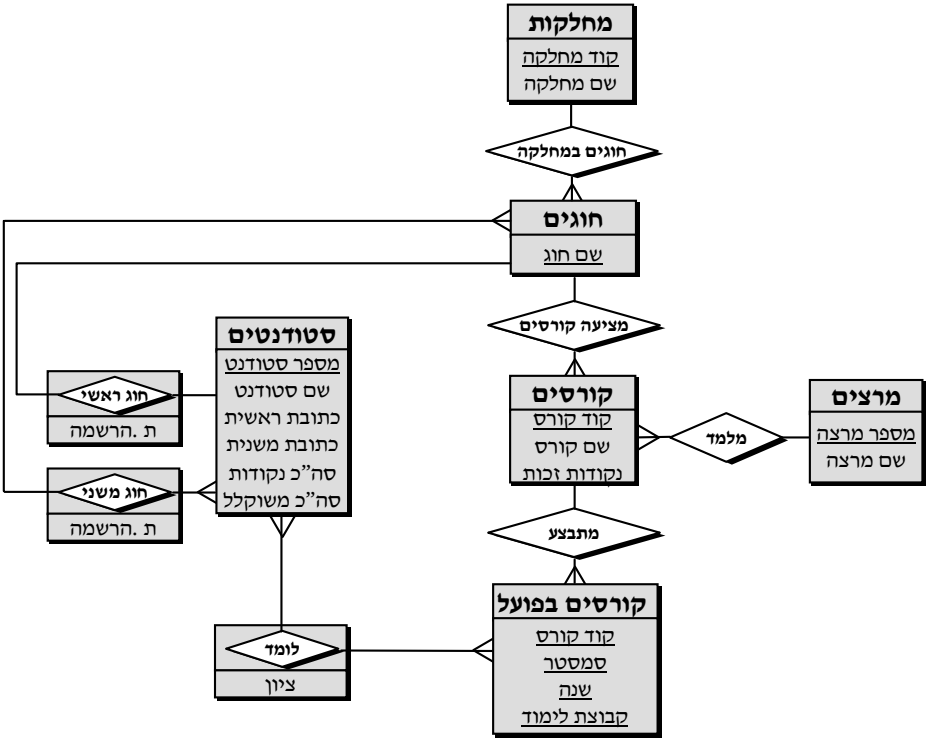


תרשים 6.20: מודל גלובלי לאחר שילוב דרישה א'.

דרישה ב' – דוח הישגים מצטבר לסטודנט

דרישה זו מציגה שני סוגי מידע בלתי תלויים: רשימת החוגים אליהם רשום הסטודנט ורשימת הקורסים וההישגים המצטברת של הסטודנט. כפי שניתן לראות מהדוח, לכל סטודנט יש חוג ראשי אחד ומספר חוגים משניים. בשלב זה צריך לעשות את האבחנה בין קורס לקורס בפועל. קורס מסוים, כגון תכנות Java, המקנה מספר מסוים של נקודות זכות חוזר על עצמו מדי סמסטר ולעיתים במספר קבוצות לימוד באותו סמסטר. את הסמסטר נפרק לשני מרכיביו, מספר סמסטר ושנה.

את התכונה נקודות משוקללות לא נכניס למודל מכיון שהוא ניתן לחישוב פשוט (מכפלת נקודות זכות בציון). לעומת זאת, את סה"כ נקודות הזכות המצטברות ואת סה"כ הנקודות המשוקללות כן נכניס למודל כדי לחסוך זמן חישוב. ברור, שזה מחייב תשומת לב בזמן תכנון היישום, מאחר ויש כאן כפילות נתונים. זוהי דוגמה לשיקול של העדפת ביצועים על סילוק כפילות נתונים. המודל המתקבל מפעולות אלו מוצג בתרשים 6.21.



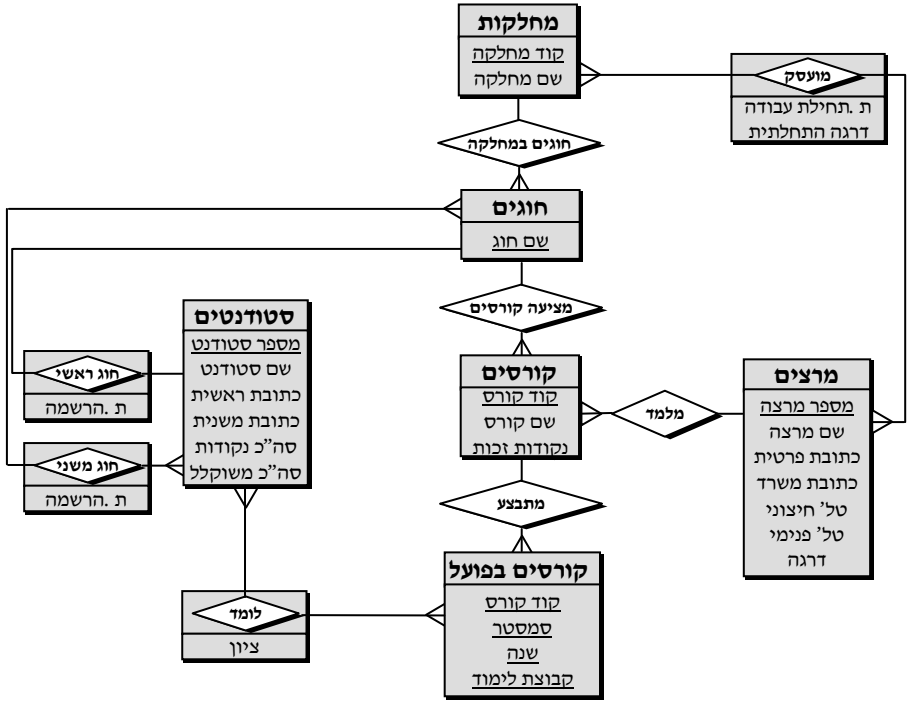
תרשים 6.21: מודל גלובלי לאחר שילוב דרישה ב'.

דרישה ג' – שאילתת מרצים לפי מחלקות

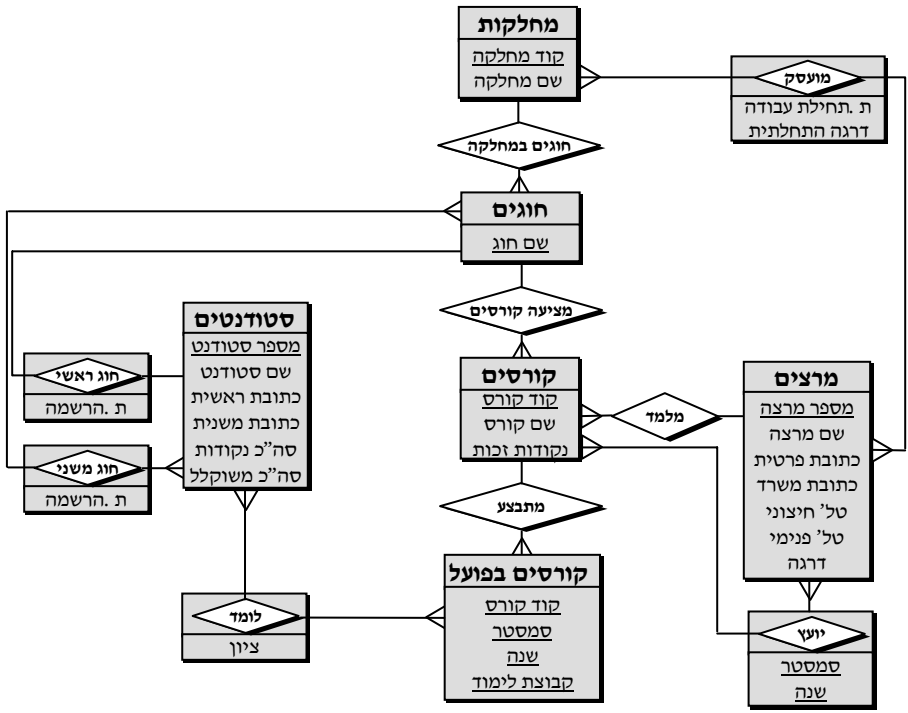
דרישה זו מוסיפה מספר תכונות חדשות לקבוצת המרצים ויוצרת קשר בין המרצים לבין המחלקות המעסיקות אותם. את המודל המתקבל מדרישה זו נוכל לראות בתרשים 6.22.

דרישה ד' – רשימת יועצים לסמסטר

מדרישה זו ניתן להבין שלכל קורס יש יועץ לכל סמסטר. ייתכן גם, שאותו מרצה ישמש כיועץ בסמסטרים שונים. ניצור קשר בין מרצים לבין קורסים. את המודל המתקבל מדרישה זו נוכל לראות בתרשים 6.23.



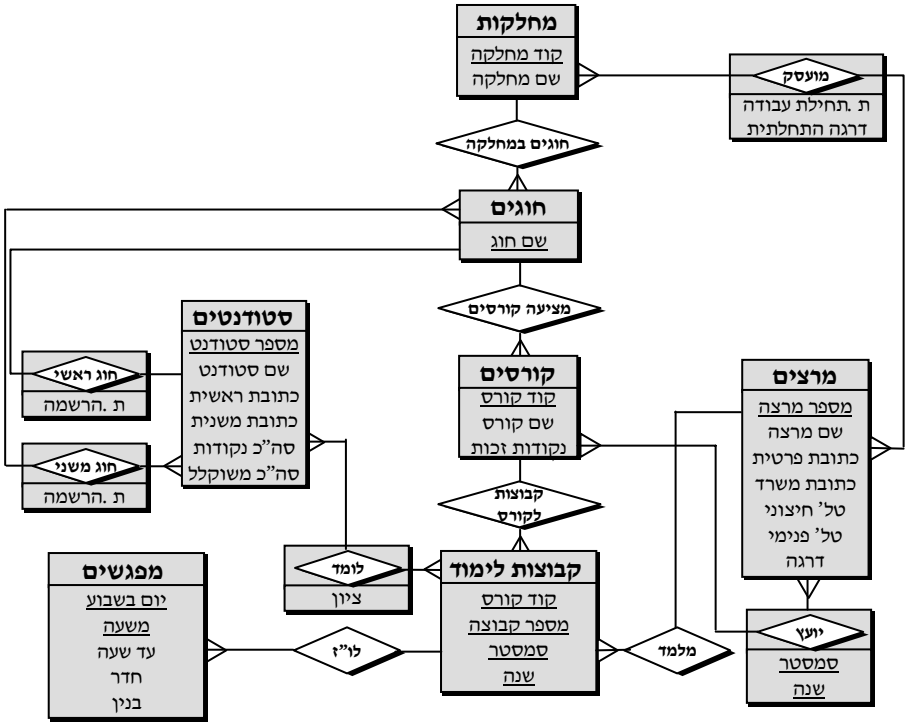
תרשים 6.22: מודל גלובלי לאחר שילוב דרישה ג'.



תרשים 6.23: מודל גלובלי לאחר שילוב דרישה ד'.

דרישה ה' – לוח זמנים לקבוצות לימוד

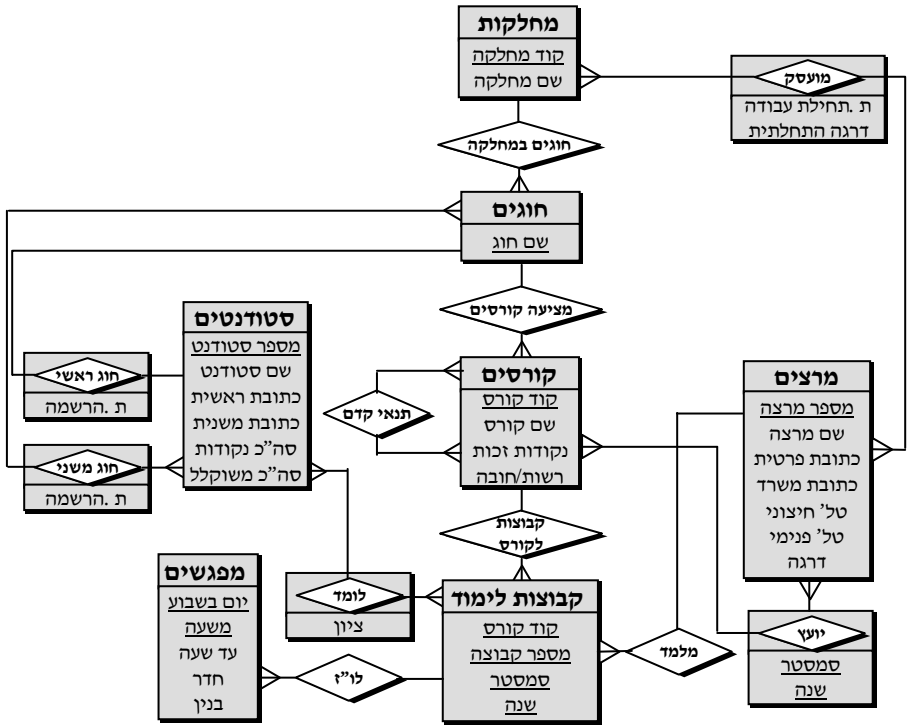
בשלב זה ברור שיש להוסיף מספר קבוצות חדשות. מתוך עיון בדוח המפגשים לקבוצות, ברור שקורס אחד יכול להנתן במספר קבוצות לימוד שונות בכל סמסטר ולכל קבוצת לימוד יש מספר מפגשים בשבוע. מרצה משמש כיועץ לקורס והוא מלמד קבוצת לימוד. את המודל המתקבל מדרישה זו נוכל לראות בתרשים 6.24.



תרשים 6.24: מודל גלובלי לאחר שילוב דרישה ה'.

דרישה ו' – אלפון קורסים

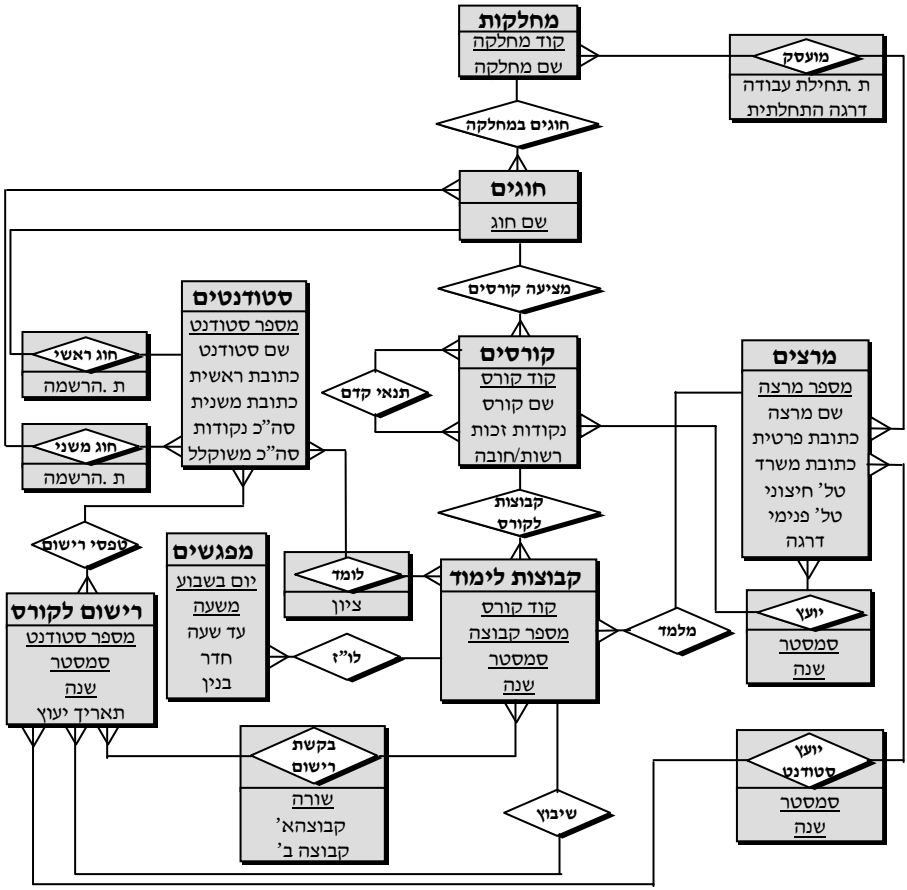
דרישת מידע זו מוסיפה את תנאי הקדם לקורס. נוסיף קשר רפלקסיבי לקורסים. כמו כן נוסיף תכונה חדשה לקורסים המציינת אם הקורס הוא רשות או חובה. את המודל המתקבל מדרישה זו נוכל לראות בתרשים 6.25.



תרשים 6.25: מודל גלובלי לאחר שילוב דרישה ו'.

דרישה ז' – טופס בקשה לרישום לקורס

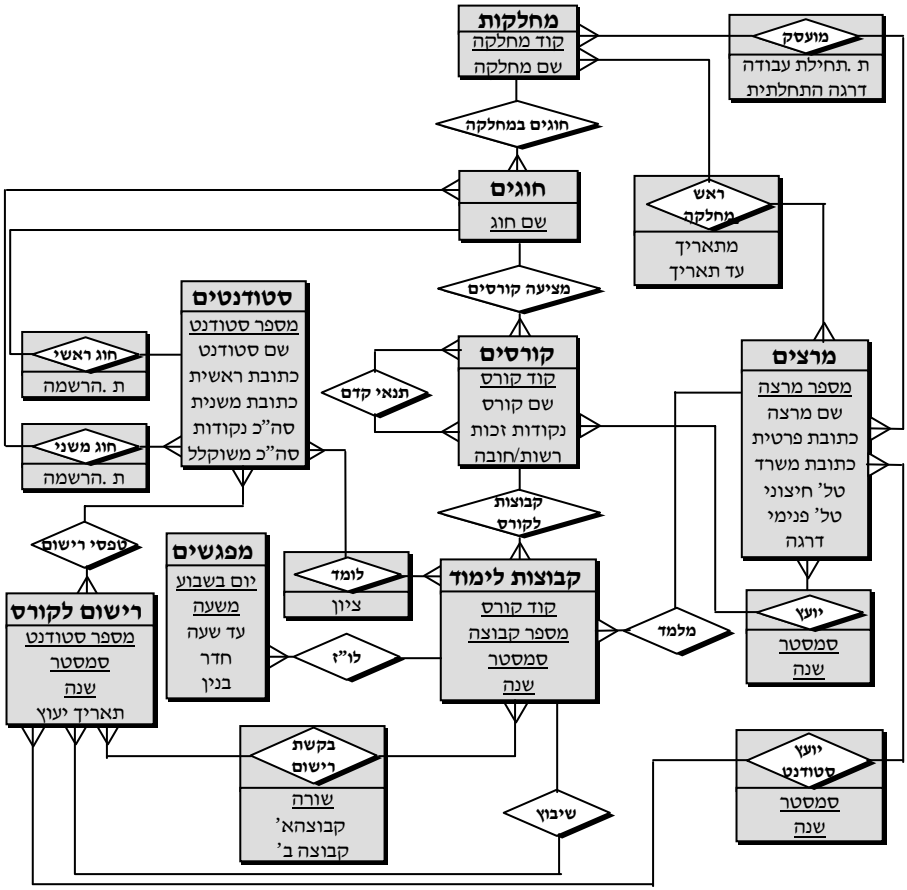
דרישה זו מציגה את טופס הרישום לקורסים המשמש בסיס להחלטות המזכירות האקדמית לשיבוץ סטודנטים לקבוצות לימוד. נוסיף קבוצה – רישום לקורס ומספר קשרים חדשים בין קבוצה זו לבין קבוצות אחרות. הקשר בקשת רישום לקורס מתעדת את בקשת הסטודנט והעדיפויות שלו לרישום לקבוצות לימוד. הקשר שיבוץ לקורס מתעד את החלטת המזכירות האקדמית והקשר יועץ אישי מתעד את המרצה שאישר את טופס הבקשה לשיבוץ לקבוצות אשר מכילה את בקשת הסטודנט ואת החלטות המזכירות האקדמית. את המודל המתקבל מדרישה זו נוכל לראות בתרשים 6.26.



תרשים 6.26: מודל גלובלי לאחר שילוב דרישה ז'.

דרישה ח' – רשימת ראשי מחלקות

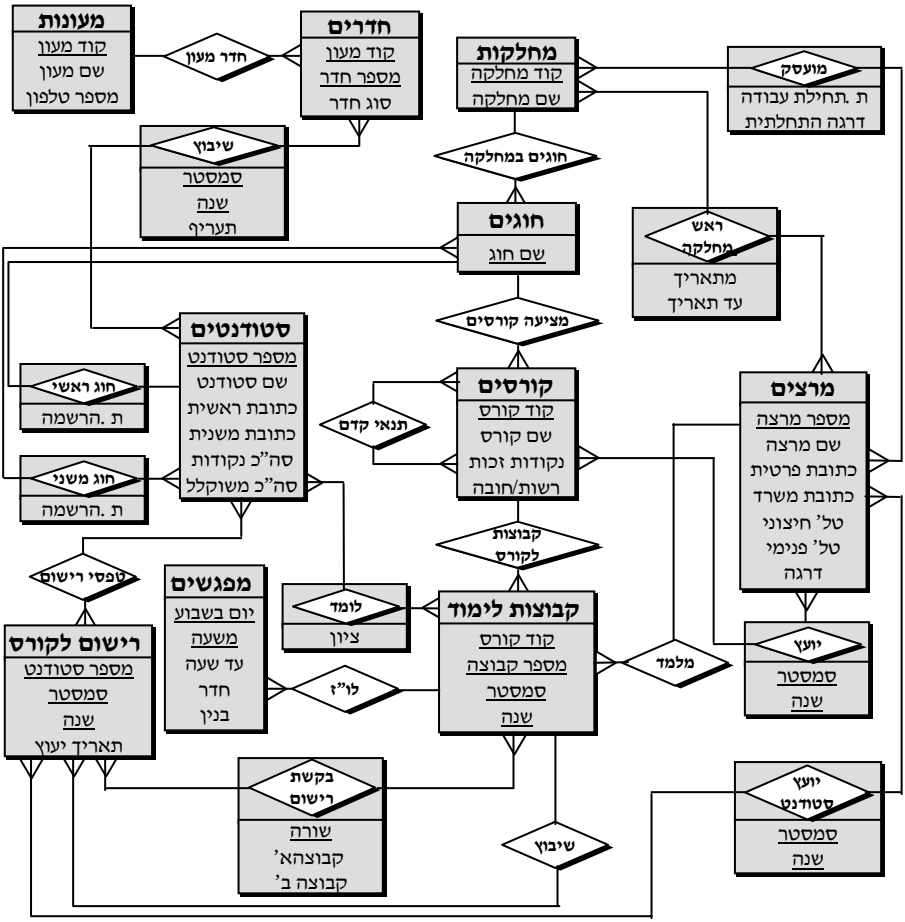
דרישת מידע זו מוסיפה קשר ראש מחלקה בין מרצים לבין מחלקות. את המודל המתקבל מדרישה זו נוכל לראות בתרשים 6.27.



תרשים 6.27: מודל גלובלי לאחר שילוב דרישה ח'.

דרישה ט' – רשימת שיבוץ סטודנטים למעונות

מדרישת מידע זו אנו מבינים שבכל מעון ישנם מספר חדרים, וסטודנטים משובצים לחדרים מדי סמסטר. נוסיף את קבוצת מעונות, קבוצת חדרים ואת השיבוץ של סטודנט לחדר. את המודל המתקבל מדרישה זו נוכל לראות בתרשים 6.28.



תרשים 6.28: מודל גלובלי לאחר שילוב דרישה ט'.

שלב ג': המרת המודל הגלובלי למודל טבלאי

בשלב זה נמיר באופן שיטתי את המודל הגלובלי שהתקבל בשלב הקודם לאוסף של טבלאות. נבדוק בכל טבלה אם היא מקיימת את כלל BCNF ואם לא, היא תפורק לאוסף אחר של טבלאות. נקבל את הסכימה הטבלאית שמוצגת בתרשים 6.29.

מחלקות (קוד מחלקה , שם מחלקה)
חוגים (שם חוג , קוד מחלקה)
מרצים (מספר מרצה, שם מרצה, כתובת פרטית, כתובת משרד, טל' חיצוני, טל' פנימי, דרגה)
ראשי מחלקות (קוד מחלקה , מספר מרצה , מתאריך , עד תאריך)
תעסוקה (קוד מחלקה , מספר מרצה , תאריך תחילת עבודה , דרגה התחלתית)
קורסים (קוד קורס , שם קורס, נקודות זכות, רשות/חובה, שם חוג)
תנאי קדם (קוד קורס , קוד קורס קדם)
קבוצות לימוד (קוד קורס , מספר קבוצה , סמסטר , שנה)
מפגשים (קוד קורס , מספר קבוצה , סמסטר , שנה , יום בשבוע , משעה , עד שעה, חדר, בנין)
סטודנטים (מספר סטודנט, שם סטודנט, כתובת ראשית, כתובת משנית, סה"כ נקודות, סה"כ משוקלל)
חוג ראשי (מספר סטודנט, שם חוג ראשי, תאריך הרשמה)
חוג משני (מספר סטודנט , שם חוג משני , תאריך הרשמה)
מלמד (מספר מרצה , קוד קורס , מספר קבוצה , סמסטר , שנה)
הישגים (מספר סטודנט , קוד קורס , מספר קבוצה , סמסטר , שנה , ציון)
יועץ לקורס (מספר מרצה , קוד קורס , סמסטר , שנה)
יועץ לסטודנט (מספר מרצה , קוד קורס , סמסטר , שנה , מספר סטודנט)
בקשת רישום לקבוצה (מספר סטודנט , קוד קורס , סמסטר , שנה , שורה, קבוצה א', קבוצה ב')
שיבוץ לקבוצה (מספר סטודנט , קוד קורס , סמסטר , שנה)
מעונות (קוד מעון , שם מעון, מספר טלפון)
חדרים במעון (קוד מעון , מספר חדר , סוג חדר)
שיבוץ למעונות (קוד מעון , מספר חדר , סמסטר , שנה , תעריף)

תרשים 6.29: סכימה טבלאית למודל התפישתי.

סיכום

בפרק זה הצגנו את המושגים העיקריים הקשורים לתהליך עיצוב מודל הנתונים התפישתי ותרגומו למודל טבלאי. השלבים העיקריים הם:

- הבנת דרישות המידע והעיבוד בהתבסס על המציאות הרלוונטית ליישום. מתוך הבנה זו נבנה מודל ישויות-קשרים ראשוני.
- בניית מודלים מקומיים. כל מודל כזה מבטא את נקודת המבט המקומית (של משתמש או של תוכנית יישום או של פעילות מסוימת) על הנתונים.
- שילוב המודלים המקומיים למודל ישויות-קשרים גלובלי אחד, תוך פתרון הסתירות השונות, במידה וקיימות.
- בניית המודל הטבלאי מתוך המודל התפישתי.

ניתן לראות את המודל התפישתי כשלב ביניים, שבו מנתח המערכת, או מעצב בסיס הנתונים, בונה את מודל הנתונים, כשהוא משוחרר מאילוצים כלשהם ומתמקד בהגדרה של הקבוצות, התכונות שלהן והקשרים ביניהן.

במובן זה המודל שמתקבל הוא "טהור וטבעי". היתרונות של בניית מודל תפישתי כשלב ראשון ומודל טבלאי כשלב שני הם:

- עבודת עיצוב בסיס הנתונים פשוטה ביותר בעקבות חלוקת המטלה לשלבים שונים בעלי פלט מוגדר.
- קל יותר לתכנן את המודל התפישתי מהמודל הטבלאי משום שהוא משוחרר מאילוצים ומשיקולי יעילות.
- המודל התפישתי מהווה מסמך חשוב המתעד את מודל הנתונים של הארגון. המודל הטבלאי מותאם למערכת RDBMS, ולעיתים קשה להסיק ממנו מה היה המבנה המקורי.
- קל יותר להסביר למשתמשים את המודל התפישתי ולקבל את הערותיהם.

שאלות חזרה ותרגילים

אירוע בעיצוב בסיס נתונים

הפעם, התרגיל שונה בסגנונו מהתרגילים בפרקים אחרים.

חברת **דירות להשכרה בע"מ** הינה אחת החברות הגדולות בארץ העוסקת בהשכרת דירות. לחברה סניפים בכל רחבי הארץ וברשותה מספר רב של דירות שהן בבעלותה ומושכרות. החברה מבקשת להקים מערכת מידע שתנהל את פעילותה. להלן תיאור הדרישות:

1. לחברה מספר סניפים ברחבי הארץ. לכל סניף יש לנהל את הנתונים הבאים: מספר סניף, שם סניף, כתובת הסניף הכוללת את הרחוב, מספר, עיר ומיקוד, מספר טלפון ראשי, מספר טלפון משני, מספר פקס וכתובת דאר אלקטרוני.

2. בכל סניף מועסקים מספר עובדים. לכל עובד יש לנהל את הנתונים הבאים: מספר עובד, שם משפחה ושם פרטי, כתובת העובד (הכוללת את הרחוב, מספר, עיר ומיקוד), מין העובד, מספר שלוחה פנימית, כתובת דאר אלקטרוני, תאריך לידה, תפקיד בסניף ותאריך תחילת עבודה בסניף. הסניף מעסיק את העובדים במספר סוגי תפקידים: מנהל סניף, אנשי תפעול, אנשי מכירות ואנשי מינהלה (מזכירות).

3. עובדים יכולים לעבור בין סניפים שונים. לכל עובד יש לנהל את ההיסטוריה של התעסוקה שלו. עבור כל שינוי במצב התעסוקה יש לנהל מידע כגון: קוד סניף, סוג השינוי (קידום בתפקיד, מעבר מסניף אחר וכד'), תאריך השינוי וסיבת השינוי. לגבי אנשי המינהלה בלבד יש לנהל מידע על הכישורים שלהם בהפעלת תוכנות משרדיות כגון Word, Excel ו- PowerPoint.

4. אנשי התפעול אחראים לתחזוקה השוטפת של הדירות השונות ולביצוע פעולות כגון צביעה, שרברבות וחשמל. אנשי התפעול מחולקים למספר צוותי עבודה. לכל צוות יש מנהל צוות המפקח ומתכנן את עבודת הצוות ומספר אנשי צוות. לכל צוות יש מזכירה אחת או יותר. מזכירה אחת יכולה לתת שירותי מזכירות למספר צוותים.

5. כל סניף אחראי על הבניינים והדירות באזור שלו. לכל בניין יש איש צוות תפעול האחראי עליו. איש צוות אחד יכול להיות אחראי על 20 בנינים לכל היותר.

6. עבור כל בניין יש לנהל את הנתונים הבאים: קוד רכוש, כתובת הבניין (הכוללת את הרחוב, מספר, עיר ומיקוד) וסוג הבניין (רב קומות, קוטג' וכד'). אם הבניין מכיל מספר דירות להשכרה, לכל דירה יש לנהל את הנתונים: מספר הדירה, סוג הדירה (רגילה, דופלקס), גודל הדירה במ"ר, קומה, יש/אין מעלית וסכום שכירות חודשי.

7. כל דירה מושכרת על ידי שוכר אחד בלבד. שוכר יכול להיות איש פרטי או חברה עסקית. אם השוכר הוא איש פרטי יש לנהל עליו את הנתונים הבאים: מספר זהות, שם משפחה ופרטי, מספר חשבון בנק, קוד בנק וקוד סניף בנק. אם השוכר הוא חברה, יש לנהל את הנתונים הבאים: מספר חברה, שם חברה, כתובת החברה, שם איש הקשר, טלפון איש קשר, פרטי בנק כגון קוד בנק, קוד סניף ומספר חשבון.

8. לכל דירה מושכרת מנוהל חוזה שכירות. המידע המנוהל בכל חוזה הוא מספר חוזה שכירות, קוד הבניין והדירה, פרטי השוכר, תאריך תחילת שכירות, תאריך סיום שכירות, סכום שכירות, סכום פיקדון, שיטת הצמדת דמי השכירות, צורת עדכון דמי השכירות שנקבעה (פעם בחודש, פעם בחצי שנה, פעם בשנה) ותאריך העדכון האחרון. החברה אינה נוהגת להשכיר את דירותיה לתקופות של פחות מחצי שנה ולא יותר משנתיים.

9. כל דירה פנויה עוברת תהליך של ביקורת תקופתית אחת לחצי שנה, לבחינת מצבה וקביעת פעולות התחזוקה שיש לבצע. את הביקורת מבצע איש הצוות שאחראי על הבניין. עליו למלא טופס ביקורת הכולל: קוד הבניין והדירה, קוד איש הצוות האחראי, תאריך ביקורת, שעת ביקורת, ממצאים עיקריים והערות.

10. כל פעולת תחזוקה המבוצעת בדירה כלשהי נרשמת. הנתונים שיש לרשום הם: קוד בניין ודירה, קוד איש הצוות האחראי, סוג פעולת התחזוקה (חשמל, מיזוג, צביעה, שרברבות וכד'), תאריך פעולת התחזוקה, תיאור הפעולה והערות.

11. החברה נוהגת לפרסם בעתונות הארצית והמקומית את הדירות הפנויות להשכרה. החברה עובדת באופן קבוע עם מספר עיתונים. לכל עיתון יש לנהל מידע על שם העיתון, תפוצה (מקומית, ארצית), תדירות פרסום העיתון (יומי, שבועי, חודשי), כתובת העיתון, מספר טלפון ושם איש קשר. לכל פרסום בעתון יש לנהל את הנתונים הבאים: שם העיתון, תאריך פרסום המודעה, גודל המודעה, מיקום המודעה (עמוד ראשי, עמוד אחורי וכד') ועלות פרסום המודעה.

12. לקוחות פוטנציאליים פונים לבירור אפשרויות השכרה בעקבות פרסום המודעה בעיתון, או על ידי פנייה ישירה לאחד מסניפי החברה. לכל לקוח פוטנציאלי שפונה לחברה נקבעת פגישה עם אחד מאנשי המכירות של הסניף. במסגרת הפגישה מקבל הלקוח הסבר על הדירה ועל תנאי ההשכרה. אם הלקוח מעוניין, מתבצע גם ביקור בדירה עצמה. לפעמים, מבקר הלקוח מספר פעמים באותה דירה או בדירות שונות עד לקבלת החלטה וחתימה על חוזה.

13. לכל לקוח פוטנציאלי יש לנהל את הפרטים הבאים: מספר לקוח, שם לקוח, כתובת, סוג לקוח (פרטי או חברה), תאריך פנייה ראשונה ושם איש המכירות שטיפל בלקוח. עבור כל ביקור בסניף יש לנהל מידע כגון: תאריך הביקור, שם איש המכירות שפגש את הלקוח, רשימת הבנינים והדירות שמעניינות את הלקוח והערות. עבור כל דירה שבה הלקוח ביקר יש לנהל מידע על תאריך הביקור, שם איש המכירות שהתלווה ללקוח והערות בעקבות הביקור.

תכנן ועצב את תרשים ישויות-קשרים ואת הסכימה הטבלאית המתאימה לבעיה המתוארת לעיל על פי המתודולוגיה המתוארת בפרק זה.

חלק ג'

שפת SQL

שפת SQL הינה כיום השפה הנפוצה ביותר לניהול וגישה לנתונים. היא נתמכת על ידי מיגוון גדול מאוד של מערכות מסחריות לניהול בסיסי נתונים טבלאיים (RDBMS). השפה הינה מצד אחד, פשוטה ומכילה מספר קטן יחסית של פקודות בעלות מספר רב של אפשרויות ומצד שני, היא בעלת עוצמה רבה. חלק זה של הספר סוקר את שפת SQL במספר פרקים נפרדים. כל פרק מתמקד בנושא מסוים.

- **פרק 7 – מבוא**: פרק זה מציג את הרקע להתפתחות השפה ממעבדות המחקר של חברת יבמ במסגרת פרויקט אבטיפוס למערכת ניהול בסיס נתונים טבלאי ועד להפיכתה לשפה נפוצה ומוגדרת על ידי אוסף של תקנים רשמיים. הפרק מציג את אפשרויות ההפעלה השונות של הפקודות – הפעלה אינטראקטיבית או הפעלה מתוך תוכניות יישום.

- **פרק 8 – פקודות לטיפול בנתונים:** פרק זה סוקר את הפקודות לטיפול בנתונים. הפרק מציג את ארבע פקודות היסוד של השפה, הפקודות SELECT, UPDATE, INSERT ו-DELETE ופקודות ייחודיות לשפת SQL: UNION ו- INTERSECT. דיון מיוחד מוקדש לצורת הטיפול של השפה ושל הפקודות לטיפול בנתונים בעלי ערכים חסרים (Null Values).

- **פרק 9 – פקודות להגדרת נתונים:** פרק זה סוקר את הפקודות להגדרת נתונים, כלומר להקמת אובייקטים חדשים בבסיס הנתונים. אובייקטים אלה יכולים להיות טבלאות חדשות, אינדקסים, אילוצים שונים וכד'. הפרק מציג את הפקודות CREATE TABLE להגדרת טבלה חדשה, ALTER TABLE לשינוי הגדרה של טבלה ו-DROP TABLE לביטול טבלה. דיון נפרד מוקדש לאחד הנושאים המרכזיים בהגדרת בסיס הנתונים, האמינות והשלמות של הנתונים (Data Integrity).

- **פרק 10 – טבלאות מדומות ושימושיהן (Views):** פרק זה מציג את המושג "טבלה מדומה", שאינה קיימת באופן ממשי בבסיס הנתונים אלא רק בצורת שאילתה הנשמרת בבסיס הנתונים. הטבלה המדומה הופכת לטבלה המוצגת למשתמש רק בעת הפעלת שאילתה המתייחסת אליה. הרעיון של הטבלה המדומה מספק רמה חדשה של גמישות הן מבחינת הגדרת נקודות מבט שונות על טבלאות בסיס הנתונים והן מבחינת ביטחון הנתונים. הפרק מציג את הפקודה CREATE VIEW המאפשרת את הגדרת הטבלה המדומה.

- **פרק 11 – בקרת גישה לבסיס הנתונים:** פרק זה עוסק בכל תפישת בקרת הגישה לנתונים בסביבת SQL. בקרת הגישה לנתונים מורכבת משילוב של שלושה מימדים שונים – הגדרת המשתמשים, הגדרת האובייקטים המוגנים והגדרת הזכויות של המשתמש על האובייקט. הפרק מציג את הפקודה GRANT להענקת זכויות גישה ואת הפקודה REVOKE לסילוק זכויות גישה.

- **פרק 12 – תכנות בסביבת SQL:** פרק זה מציג כיצד ניתן לשלב פקודות SQL בתוך תוכניות יישום הכתובות בשפת תכנות כלשהי. שפת תכנות זו, הנקראת **שפה מארחת**, יכולה להכיל פקודות SQL המשובצות באופן ישיר או יכולה לפנות לממשק תכנות המסופק על ידי מערכת RDBMS. הפרק מציג את השיטה לשיבוץ פקודות (Embedded SQL) ואת שיטת הפנייה דרך ממשק קריאה (Call Interface). יש הבחנה בין פקודות SQL סטטיות שבהן הפקודה מוגדרת בעת כתיבת התוכנית לבין פקודות SQL דינמיות, שבהן הפקודות נוצרות תוך כדי ריצת התוכנית.

- **פרק 13 – פרוצדורות בסיס נתונים (Stored Procedures and Triggers):** פרק זה מציג את נושא "פרוצדורות בסיס נתונים", אשר הופכות את בסיס הנתונים לאקטיבי, המאחסן בנוסף לנתונים גם לוגיקה עסקית. הפרק מציג "פרוצדורות שמורות" (Stored Procedure) – לוגיקה המופעלת על פי בקשת תוכנית היישום, ואת תפישת המזניק (Trigger) – לוגיקה המופעלת באופן ישיר על ידי מערכת RDBMS כתוצאה מאירוע כלשהו.

- **פרק 14 – תנועות בסביבת SQL:** פרק זה מציג את נושא עיבוד התנועות (Transactions), נושא בעל חשיבות מרובה בהפעלת יישומים מבוססי SQL. הפרק מציג את שתי הפקודות העיקריות של שפת SQL, הפקודה COMMIT המסמנת סיום מוצלח של תנועה והפקודה ROLLBACK המאפשר ביטול של כל העדכונים שהתנועה ביצעה. בנוסף למושג התנועה, מציג הפרק את יומן האירועים (Log File), המנגנון המאפשר למערכת RDBMS לבטל בעת הצורך את כל העדכונים שתנועה ביצעה. הפרק סוקר גם את הבעייתיות המיוחדת של עבודה בסביבה מרובת משתמשים המעדכנים את בסיס הנתונים בו-זמנית ואת מנגנוני הנעילות המשמשים את מערכות RDBMS לטיפול במצבים אלה.

- **פרק 15 – קטלוג המערכת:** פרק זה מציג את קטלוג המערכת, אוסף הטבלאות המשמשות את מערכת RDBMS ומכילות את ההגדרות של בסיס הנתונים. בנוסף למנהל בסיס הנתונים, המשתמש בטבלאות אלו כדי להבין את תכולת בסיס הנתונים, מאפשרת שפת SQL לגשת אל טבלאות אלו באמצעות פקודות SELECT רגילות, כדי לאפשר למשתמשים שונים לתחקר את מבנה בסיס הנתונים.

פרק 7

מבוא ומושגי יסוד בשפת SQL

1. מבוא – הרקע להתפתחות השפה
2. שפת SQL כשפה תקנית
3. אי-תלות בנתונים בשפת SQL
4. אופן הפעלת פקודות SQL
5. סוגי פקודות SQL
6. אבני הבניין של פקודת SQL
7. טיפוסי נתונים (Data Types)
8. שיטה לתיאור מבנה פקודת SQL
9. בסיס נתונים לדוגמה

מבוא – הרקע להתפתחות השפה

במהלך השנים הוצעו מספר רב של שפות לטיפול בטבלאות. מבין כל השפות האלו, שפת SQL נקלטה בצורה הטובה ביותר והפכה למעשה לשפה הנפוצה והסטנדרטית לניהול בסיסי נתונים טבלאיים. כל מערכות RDBMS המסחריות תומכות כיום בשפת SQL. בגלל התפוצה הגדולה של בסיסי הנתונים הטבלאיים, פותחו במשך השנים מספר רב של כלים המבוססים על שפת SQL וביניהם ניתן למנות כלים כגון מחוללי יישומים, מחוללי שאילתות (Query Generators), מחוללי דוחות, מערכות לניתוח מידע רב-מימדי (Multi Dimensional Analysis), מערכות להעתקת נתונים או העברתם, מערכות לשכפול נתונים (Data Replication) וכד'.

השפה עצמה פותחה במסגרת פרויקט במעבדות המחקר של חברת יבמ ב- San Jose שהחל בשנת 1974 ועסק בהקמת האבטיפוס של מערכת לניהול בסיס נתונים טבלאי, System/R. מערכת זו פותחה כדי לבחון את הישימות של הרעיונות שהוצגו לראשונה על ידי E.F. Codd במאמרו המפורסם משנת 1970 שבו הניח את התשתית הרעיונית לתפישת בסיסי הנתונים הטבלאיים – "A Relational Model of Data for Large Data Banks" [CE70]. בתחילת הדרך נקראה השפה Sequel, אולם בהמשך שונה שמה ל-**SQL** – **שפת שאילתות מובנית** (Structured Query Language), כדי למנוע בלבול עם מוצר אחר. גם כיום יש כאלה המבטאים את שם השפה כ- Sequel וכאלה המבטאים SQL. פרויקט System/R המתואר במאמר [AM76] הסתיים בשנת 1979 לאחר שחברת יבמ הגיעה למסקנה שבסיסי נתונים טבלאיים הם ברי יישום, והחלה בפיתוח המערכת המסחרית הראשונה המבוססת על מודל זה, מערכת SQL/DS ששוחררה ללקוחות בשנת 1981. במקביל למאמצי המחקר של חברת יבמ בוצעו מחקרים בנושאי בסיסי נתונים טבלאיים גם במעבדות אוניברסיטת קליפורניה בברקלי, שהביאו לפיתוחה של שפה מקבילה בשם Quel שהיוותה לאחר מכן בסיס למוצר מסחרי בשם Ingres. תקופה מסוימת היתה תחרות בין שתי השפות אולם בסופו של דבר ניצחה שפת SQL עם ההודעה על תמיכה בשפה זו גם על ידי מערכת Ingres בשנת 1986 ועם פרסום התקנים הראשונים של שפת SQL [AN86].

חשוב להדגיש שהשם SQL אינו משקף נכונה את המשמעות האמיתית מהסיבות הבאות:

- **SQL אינה שפת שאילתות**: שפת SQL היא שפת ניהול נתונים מלאה ועשירה ולא שפת שאילתות בלבד כפי שניתן להבין משמה – Structured ***Query*** Language. השפה מספקת את כל מיגוון הפעולות הנדרשות לניהול הנתונים בבסיס הנתונים משלב הגדרת הנתונים, דרך הטיפול בנתונים ועד לנושאים מורכבים של ניהול תנועות והרשאות גישה לנתונים. השפה תומכת באופן מלא בכל פעולות היסוד – קריאה, עדכון, ביטול והוספה ותומכת במיגוון נוסף של פעולות – **הגדרת הנתונים** (Data Definition), **ניהול תנועות** (Transaction Management) **ואבטחת הנתונים** (Data Security). מבחינה זו, שפת SQL הינה שפת הנתונים המקיפה ביותר ומשלבת בתוכה את כל מיגוון הפעולות הדרושות לניהול בסיס נתונים. עד להופעת שפת SQL התבססו כל המערכות לניהול בסיסי נתונים על שתי שפות נפרדות – **שפה להגדרת נתונים** (DDL) ששימשה בעיקר את מנהל בסיס הנתונים **ושפה לטיפול בנתונים** (DML) ששימשה את מפתחי היישומים.

- **SQL אינה שפת תכנות מלאה:** שפת SQL איננה שפת תכנות מלאה כפי שניתן להבין משמה – Structured Query ***Language*** – אלא רק שפת נתונים המאפשרת את הגדרת הנתונים ואת הטיפול בהם. למען הדיוק יש להתייחס אל שפת SQL כאל תת-שפה (Sub Language) המיועדת לניהול נתונים בלבד. שפת SQL אינה כוללת פקודות לביצוע לוגיקה אלא רק פקודות להגדרת וניהול של הנתונים. כדי לבצע עיבוד מלא של הנתונים יש צורך להשתמש בשפת תכנות כלשהי כגון Cobol, Pascal, Visual Basic, C++, Java או בשפות דור רביעי כגון PowerBuilder, Visual Age, המכילות את כל מיגוון הפעולות הדרושות לעיבוד הנתונים, מבנים כגון If-Then-Else, לולאות וכד'. שפת SQL יכולה להתארח בתוך שפת תכנות כלשהי כדי להשלים את השפה בכל הקשור לגישה וניהול הנתונים בבסיס נתונים טבלאי. המצב שבו שפת SQL מתארחת בשפת תכנות אחרת נקרא Embedded SQL ולשפה המארחת מקובל לקרוא Host Language.
- **SQL אינה שפה מובנית:** שפת SQL איננה שפה מובנית וקשיחה כפי שניתן אולי להבין משמה – Query Language ***Structured*** – אלא שפה מודרנית המאפשרת גמישות רבה ביותר. במקור המונח מובנה נולד מהרעיון של תמיכה בשאילתה המכילה תת-שאילתה שיכולה בעצמה להכיל תת-שאילתה, כלומר יצירת שאילתה הבנויה ממספר רב של שאילתות. בסופו של דבר, תכונה זו היא רק תכונה אחת של השפה ואפילו לא התכונה העיקרית שלה. המהדרים של שפת SQL נבנו על סמך מיטב הניסיון שנצבר בכל הקשור בתורת ההידור (Compilation) ולכן תומכים בכתיבת פקודות SQL בצורות שונות ומשונות – באותה שורה או בשורות נפרדות, עם מספר משתנה של רווחים, עם אותיות גדולות או קטנות או שילוב כלשהו וכד'. מבחינה זו ניתן לומר ששפת SQL גמישה למדי ופחות מובנית משפות נתונים אחרות.

בהתבסס על כל האמור לעיל, נוטים כיום להתעלם מהמשמעות העומדת מאחורי שמה של SQL ומתייחסים אל הצירוף לאו דווקא כאל קיצור של שלוש אותיות בעלות משמעות כלשהי. כמובן, שבחירה לא מוצלחת זו של שם השפה אינה מפחיתה בצורה כלשהי מהחשיבות העצומה של שפה זו ומהמהפכה שהיא ובסיסי הנתונים הטבלאיים הביאו אל עולם המחשוב המודרני.

בפרק זה

- נציג את שפת SQL כשפה לניהול בסיסי נתונים טבלאיים.
- נסביר את התכונות המייחדות את SQL לתפקידה.
- נסביר את פקודות SQL, נראה כיצד הן פועלות וכיצד ניתן לתאר אותן.
- נציג את טיפוסי הנתונים הנהוגים בשפה.
- נציג בסיס נתונים לדוגמה, אשר נפנה אליו במהלך הקריאה בספר.

שפת SQL כשפה תקנית

בשנת 1982 החל מכון התקנים האמריקאי, ANSI, לעבוד על פיתוח תקן רשמי. תקן זה פורסם באופן רשמי בשנת 1986 – תקן ANSI X3/135 SQL Database Language [AN86]. בשנת 1987 אומץ תקן זה גם על ידי ארגון התקינה הבינלאומי, ISO. מאז קיים תהליך מתמיד של שיפור והרחבת התקן. כיום שפת SQL הוכרה כשפה תקנית על ידי גופי התקינה והארגונים הבאים:

שם מקוצר	השם המלא
ANSI	American National Standards Institute
ISO	International Standards Organization
FIPS	U.S Federal Information Processing Standards
OSF	Open Software Foundation
X/Open	European Organization for portable Unix-based applications
SQL Access Group	Consortium of Database & Computer vendors for SQL call level specification
Microsoft ODBC	Microsoft standard for Open Database Connectivity

הטיפול וההכרה של כל הארגונים האלה בשפה הביאו לתפוצתה הרבה ולשילובה במספר עצום של מערכות RDBMS ובמוצרי תוכנה המסוגלים לפעול על מיגוון רחב של פלטפורמות מיחשוב ומערכות הפעלה רבות. לאחר פרסום התקן הראשון בשנת 1986, פורסמה מהדורה מעודכנת שלו בשנת 1989 שמקובל כיום לקרוא לה בשם תקן SQL1. בשנת 1992 פורסם תקן חדש שמקובל לקרוא לו בשם SQL-92 או SQL2. התקן מגדיר שתי רמות של תאימות: רמה ראשונה – ANSI SQL Level 1, ורמה שנייה – ANSI SQL Level 2 המכילה מספר תוספות לעומת הרמה הראשונה. התקן מכיל גם נספח מיוחד המתייחס לנושאי אמינות ושלמות בסיס הנתונים (Data Integrity). בשנים האחרונות שוחררה טיוטא של התקן החדש שנקרא לעת עתה SQL3, המרחיב את התקן ומטפל בנושאים חדשים כגון אובייקטים, Triggers, רקורסיה ועוד. נכון לזמן כתיבת הספר עדיין לא נקבע המועד הרשמי לשחרור התקן החדש. בגלל ההרחבה הניכרת של הנושאים הכלולים ב-SQL2 וב-SQL3, תהליך גיבוש התקן לוקח הרבה יותר זמן מהמתוכנן ומעורר מספר רב של ויכוחים מקצועיים.

בגלל הפרסום המאוחר יחסית של התקן ובגלל השיפור המתמיד שלו, נוצרה בעיה של תאימות שפת SQL הנתמכת על ידי מערכות RDBMS מסחריות שונות לתקן. בשנים הראשונות של הופעת מערכות מסחריות מבוססות SQL, הרשו לעצמם היצרנים השונים לפתח ניבים שונים של השפה וכן הרחבות שונות שהם האמינו בחשיבותן. דבר זה גרם ליצרנים בעיה של התאמת השפה לתקנים והפסקת התמיכה בהרחבות הקנייניות החורגות מהתקן. ניתן לומר שהרוב הגדול של המערכות המסחריות תומך בתקן של שנת

1989 וחלקן תומך בתקן SQL2. חשוב להדגיש שתקן SQL1 לא התייחס לאוסף גדול מאוד של נושאים כגון קודי שגיאה, טיפוסי נתונים נתמכים, טבלאות קטלוג המערכת, Dynamic SQL, ממשק תכנות מתוך שפה מארחת, פרוצדורות בסיס נתונים וכד'. כך, התקן הזה לא הותיר ברירה בידי היצרנים אלא לקבוע עובדות בשטח.

נכון להיום ובגלל העיכוב הגדול בהופעת תקן SQL3, חלק מהיצרנים אף לקחו נושאים המופיעים בטיוטא של תקן SQL3 והחלו לתמוך בהם, למרות שיתכן שעם פרסום הגירסה הסופית של התקן יחולו שינויים. בהכללה, ניתן לומר שעם השנים סביר להניח שרוב היצרנים של מערכות RDBMS המסחריות יתמכו בגרסאות המתקדמות של התקן. ברור שקצב התאמת השפה לתקן יהיה שונה אצל כל יצרן וייווצר פיגור גדול יחסית בגלל המורכבות של שינוי השפה וההשפעות של דבר זה על יישומים שפותחו בגרסאות קודמות. יחד עם זאת, אין כל ספק שכל שהשנים חולפות, מערכות מסחריות רבות יותר תתמוכנה בגרסאות העדכניות ביותר של התקנים.

לעובדה ששפת SQL הינה שפה תקנית יש חשיבות מרובה במספר היבטים שונים:

- **ניידות (Portability) היישום מול בסיסי נתונים שונים:** מכיון שהשפה הינה שפה תקנית, ניתן לפתח יישומים שיפעלו באופן זהה ושקוף עם מספר בסיסי נתונים מסחריים שונים. לדוגמה מערכת R/3 של חברת SAP, אחת המערכות הנפוצות לניהול כולל של משאבי הארגון (ERP), מסוגלת לפעול באופן זהה עם מערכות RDBMS כגון Oracle, SQL Server, Informix ואחרות.

דוגמה אחרת לניידות זו היא מחולל השאילתות Business Objects המאפשר גישה מול מספר רב של בסיסי נתונים שונים. האחידות של שפת SQL מאפשרת לארגונים לבחור את בסיסי הנתונים שלהם ולבחור את היישומים שיפעלו עם בסיס הנתונים, כמעט באופן בלתי תלוי ומאפשרת ליצרני התוכנה לבנות מוצרים בלתי תלויים בבסיס נתונים מסוים. כמובן שיצרני תוכנה אלה חייבים להיצמד לתקנים של השפה, על כל המשתמע מכך ולהימנע מהפעלת תכונות מסוימות של בסיס הנתונים, למרות שהרחבות מיוחדות אלו יכולות לפשט את הפיתוח ואולי אף לשפר את הביצועים. למרות היות השפה שפה תקנית, לא ניתן להתייחס אל נושא הניידות כאל נושא המובן מאליו ושקוף. מפתחי התוכנה חייבים להשקיע תשומת לב רבה בבואם לבנות מערכות בלתי תלויות בבסיס הנתונים.

- **שימור ההשקעה בהדרכת מהנדסי תוכנה ומשתמשים:** אחידות השפה מביאה לכך שמהנדס תוכנה שלמד והתנסה בעבודה בשפת SQL עם בסיס נתונים מסחרי מסוים, מסוגל במהירות רבה לעבור לעבוד מול בסיס נתונים אחר. לדוגמה, מהנדס תוכנה שעובד עם בסיס הנתונים DB2 במחשב יבמ מרכזי יוכל במהירות רבה לפתח תוכנה בבסיס הנתונים Oracle על שרתי Unix או ב- SQL Server על שרת NT. לדבר זה חשיבות מרובה כיום עם הרחבת השימוש במחסני נתונים (Data Warehouse).

מחסן הנתונים היא סביבת מיחשוב מודרנית לתמיכה בקבלת החלטות שבהן מאפשרים למשתמשי קצה לגשת אל בסיס הנתונים באמצעות מחוללי שאילתות, מחוללי דוחות וכלי ניתוח מידע שונים. מחולל השאילתות מבוסס על שפת SQL ועל ממשק גרפי ידידותי ונוח המסתיר מהמשתמשים חלק מהמורכבות של השפה. כלים

אלה מאפשרים למשתמשים להפעיל מיגוון רחב של ניתוחי מידע ללא חשיפה ישירה לשפת SQL. יחד עם זאת, המשתמשים המתוחכמים המבקשים לבנות שאילתות מורכבות צריכים להשתמש בסופו של דבר בשפת SQL ולכן הם נדרשים להשקיע בלימוד. הנקודה החשובה כאן היא שההשקעה בלימוד השפה אינה אובדת בעת מעבר מהנדס התוכנה או המשתמש מבסיס נתונים אחד למשנהו בגלל האחידות של מאפייני השפה.

- **קישוריות רחבה (Connectivity):** הדמיון הרב בין בסיסי הנתונים הטבלאיים וקיום תקנים מוסכמים מאפשר לפתח ממשקי גישה (Gateways) לשם גישה באופן שקוף ממוצר תוכנה כלשהו למספר גדול מאוד של בסיסי נתונים שונים. לדוגמה, שימוש בתקן ODBC של חברת Microsoft מאפשר לפתח יישום הניגש באופן שקוף לבסיס נתונים Oracle, Sybase, DB2 ואחרים. מבחינת מפתח התוכנה הוא משתמש בשפת SQL תקנית ואינו מודע כלל לעובדה שקיים שוני כלשהו בין בסיסי הנתונים השונים.

הקישוריות מאפשרת לארגונים לבחור במוצרי תוכנה הפועלים על בסיסי נתונים שונים ולפתח נגישות קלה יחסית אל הנתונים של יישומים אלה. זאת למרות שהם פועלים על בסיסי נתונים שונים ולעיתים גם על פלטפורמות חומרה ומערכות הפעלה שונות. דוגמה נוספת לקישוריות זו הוא התקן DRDA של חברת יבמ המאפשר נגישות של יישום לנתונים המנוהלים בבסיס נתונים DB2 במחשב מרכזי המבוסס על מערכת הפעלה OS/390, או בבסיס נתונים DB2/6000 המנוהל על שרת RS/6000 הפועל עם מערכת הפעלה AIX או בבסיס נתונים DB2/400 במחשב AS/400 הפועל עם מערכת הפעלה OS/400.

- **ביזור נתונים (Data Distribution):** הארגונים המודרניים עוברים תהליכי ביזור בלתי פוסקים והיכולת לנהל את הנתונים באופן מבוזר הופכת לתנאי חשוב ליכולת הארגון לתפקד. העובדה שמוצרי תוכנה שונים הם מבוססי SQL מאפשרת לארגון לבזר בצורה קלה יותר את הנתונים, אפילו על מערכות בסיסי נתונים של יצרנים שונים. התקן הוא הדבק המבטיח שמוצרים שונים אלה ידעו לתפקד ולפעול יחדיו בצורה כלשהי.

- **תפוצה נרחבת שהולכת וגדלה כל הזמן:** האחידות וקיום התקן הביאו לתפוצה רחבה מאוד של מוצרים המבוססים על שפת SQL. אחת החברות שנתנה דחיפה חזקה לנושא זה היתה חברת המחשבים יבמ, שאימצה את שפת ה-SQL כשפת הגישה הסטנדרטית לכל בסיסי הנתונים שלה הפועלים על הפלטפורמות השונות שהיא מייצרת. כיום נתמכת שפה זו על ידי מערכת DB2 הפועלת במחשבים מרכזיים עם מערכת הפעלה MVS או OS/390, על ידי מחשבי הביניים AS/400, על ידי שרתי Unix מסוג RS/6000 ועל ידי מערכות הפועלות על מחשבים אישיים. החברה פיתחה תקן פנימי משלה בשם DRDA, המאפשר ליישומים לפעול תוך שיתוף נתונים המנוהלים על פלטפורמות שונות. אימוץ זה של שפת SQL על ידי חברת יבמ היווה תמריץ למספר גדול מאוד של יצרנים נוספים לאמץ את השפה ולפתח מערכות המבוססות עליה.

- **שיפור מתמיד:** אימוץ שפת SQL על ידי מספר רב כל כך של גופים מסחריים רבי- עוצמה ומשאבים, תרם לכך שהשפה הולכת ומשתפרת כל הזמן. חוקרים רבים במעבדות מחקר באוניברסיטאות ובחברות מסחריות משקיעים מאמצים רבים לשכלל את השפה, להבטיח ביצועים טובים יותר, להבטיח את יכולת בסיסי הנתונים לטפל בבסיסי נתונים גדולים מאוד (VLDB - Very Large Databases), להרחיב את השפה לטיפול בטיפוסי נתונים חדשניים כגון תמונות, קול, גרפים, להכניס לשפה רעיונות הבאים ממערכות Object Oriented, להתאים את השפה לסביבת האינטרנט ולאחסון דפי HTML, לשילוב שפת Java לכתיבת פרוצדורות בסיס נתונים ותאימות גבוהה יותר ליישומי Web ועוד. שיפור מתמיד זה מאפשר לארגון הרוכש מוצר מבוסס SQL להיות בטוח שהוא נמצא בזרם העיקרי של עולם המחשוב והוא לא יישאר בודד עם מוצר ייחודי.

אי-תלות בנתונים בשפת SQL

אחת המטרות העיקריות בפיתוח שפת SQL היתה פשטות הטיפול בנתונים. ואמנם, אחת התכונות המעניינות ביותר של השפה היא הניווט האוטומטי בבסיס הנתונים. העברת המטלה לקביעת נתיב הניווט בבסיס הנתונים ממהנדס התוכנה או המשתמש אל מערכת RDBMS, מאפשרת להם להגדיר ***מה*** הם הנתונים המבוקשים מתוך בסיס הנתונים, ולא לעסוק ב***איך*** לקבל את הנתונים.

כל שפות הנתונים של בסיסי הנתונים הלא טבלאיים (היררכיים, רשתיים ואחרים) מחייבות את מהנדס התוכנה לקבוע איך להגיע אל הנתונים, באמצעות אוסף של אופרטורים המופעלים בעיתוי מתאים מתוך תוכנית היישום: "קרא רשומת אב", "קרא את רשומת הבן הראשונה", "קרא את רשומת הבן הבאה", "בדוק אם הגעת לרשומת הבן האחרונה" וכד'. סוג זה של אופרטורים מחייב את מהנדס התוכנה להכיר באופן מעמיק את מבנה בסיס הנתונים ואת כל מסלולי הגישה הקיימים אל הנתונים: אינדקסים, מצביעים, הסדר שבו נשמרות הרשומות בבסיס הנתונים וכד'. דבר זה הביא כמובן לכך שהשימוש בבסיסי הנתונים הלא יחסיים דרש התמחות מיוחדת והיה נחלתם הפרטית של מהנדסי התוכנה ולא של משתמשי הקצה שלא יכלו ולא רצו להשקיע את הזמן הרב הנדרש בלימוד טכניקות הניווט בבסיסי הנתונים.

לעומת מצב זה, שפת SQL אינה מחייבת הכרת מסלולי הניווט והגישה השונים והמורכבים אל הנתונים. כל שעל מהנדס התוכנה או המשתמש לדעת הוא באילו טבלאות נמצאים הנתונים הדרושים לו ואילו עמודות יוצרות את הקשרים הלוגיים בין הנתונים כדי לבצע צירוף (Join) של הטבלאות השונות. המהדר של שפת SQL יפעיל אוסף של החלטות אוטומטיות לבניית מסלול הגישה והשליפה האופטימלי של הנתונים. למרכיב זה של המהדר קוראים בשם רכיב האופטימיזציה (Optimizer) והוא זה שאחראי על תרגום אוטומטי של ***מה*** ל***איך***. המערכות המסחריות השונות לניהול בסיסי הנתונים הטבלאיים נבחנות, בין היתר, על פי טיב ומידת התחכום של רכיב האופטימיזציה שלהן וביכולתן לתרגם את פקודות SQL לגישה מהירה ויעילה אל הנתונים. זהו הבדל עצום לעומת שפות הנתונים הישנות שהתבססו על מהנדסי התוכנה בביצוע תרגום זה.

להפרדה זו בין ***מה*** לבין ***איך*** היתה גם השלכה רבה על פיתוח תפישת שרת/לקוח (Client/Server) בגלל היכולת לשגר פקודת SQL מתחנת העבודה, המחשב האישי בדרך כלל, אל השרת המרוחק ולאפשר למערכת RDBMS להחליט כיצד לשלוף את הנתונים ולאחר מכן לשגר אל מחשב הלקוח רק את תוצאת השליפה. שיטה זו הביאה להקטנה משמעותית ברמת האינטראקציה בין תחנת העבודה של הלקוח לבין השרת. רמת האינטראקציה הזו היא נקודה כואבת ובעייתית בסביבה שרת/לקוח המבוססת על רשת תקשורת המקשרת בין מחשב השרת לבין מחשב הלקוח.

להעברת תהליך הניווט בבסיס הנתונים ממהנדס התוכנה אל המערכת לניהול בסיס הנתונים יש גם השלכה מיידית על מידת הפריון של מהנדסי התוכנה המשתמשים בשפת SQL – הפריון שלהם גדול יותר, מאחר וחלק גדול מהלוגיקה שהם היו צריכים לבנות בתוך התוכנה ועסקה בניווט בבסיס הנתונים, נעלמת ומועברת אל מערכת RDBMS ומשחררת אותם להתמקד ביישום ופחות בצורת הגישה אל הנתונים. בנוסף, אפשרה תכונה זו של שפת SQL גם למשתמשים, שאינם אנשי מקצוע בפיתוח תוכנה, ללמוד את השפה בקלות יחסית ולבצע מטלות מורכבות למדי של שליפת נתונים באמצעות פקודות SQL. דבר זה קיבל משנה חשיבות עם הופעתם של מחסני הנתונים וניתן לומר שבמובן מסוים יכולת זו היא גם זו שתרמה רבות להופעתם.

לסיכום, בהחלט ניתן לומר ששפת SQL שהפכה לשפה מקובלת ותקנית, נחשבת לאבן דרך משמעותית בהתפתחות הנדסת התוכנה המודרנית והיא תורמת רבות לפיתוח מערכות תוכנה באופן מהיר ואמין יותר.

חלק זה של הספר מציג את שפת SQL על כל עושרה ואפשרויותיה. רוב הדוגמאות תואמות את תקן SQL2 ואינן מבוססות על מערכת מסחרית כלשהי. הקורא מתבקש לקחת נקודה זו לתשומת ליבו בבואו לתרגל ולעבוד עם מערכת מסחרית מסוימת ולהתייחס להבדלים האפשריים בין המובא בספר לבין שפת SQL כפי שהיא מיושמת במערכת כלשהי.

כיצד מפעילים פקודות SQL

שפת SQL הינה שפה עשירה מאוד ומכילה מיגוון רחב של אפשרויות. ניתן להפעיל את השפה בשני אופני פעולה שונים: בצורה אינטראקטיבית מתחנת עבודה או על ידי שיבוצה בתוך שפת תכנות מארחת. ללא קשר לאופן ההפעלה, התחביר והמבנה של שפת SQL דומים בשני אופני הפעולה, למעט מספר שינויים שנדרשו כדי לאפשר את הפעלת פקודות SQL מתוך השפה המארחת.

- **הפעלה אינטראקטיבית (Interactive SQL):** בשיטת הפעלה זו המשתמש משגר פקודות SQL מתחנת העבודה שלו אל מערכת RDBMS. המערכת מקבלת את הפקודה, מבצעת הידור ואופטימיזציה, מבצעת את הפקודה ומחזירה אל תחנת העבודה של המשתמש את התוצאה. צורת עבודה זו מאפשרת למשתמשים גישה מהירה ומזדמנת אל בסיס הנתונים. בגלל העוצמה של שפת SQL ניתן לפעמים לענות תוך שניות או דקות על דרישות מידע מורכבות למדי. עקרונית ניתן לגשת לבסיס הנתונים באופן

אינטראקטיבי על ידי כתיבה ישירה של פקודות SQL על המסך ושיגורם למערכת, או על ידי שימוש במחולל שאילתות (Query Generator).

בשנים האחרונות הופיעו בשוק מספר רב של מחוללי שאילתות שמאפשרים גישה נוחה וקלה מאוד אל בסיסי נתונים טבלאיים. מחוללי שאילתות אלה מאפשרים למשתמש לגשת לבסיס הנתונים אפילו ללא ידיעת SQL. תוך שימוש בממשק גרפי ותכונה של Drag & Drop, המשתמש מצביע על הטבלאות, על העמודות המבוקשות, מגדיר את תנאי השליפה, את סדר המיון וכד' ומחולל השאילתות בונה באופן אוטומטי את פקודת SQL המתאימה. מחולל השאילתות משגר את הפקודה אל בסיס הנתונים ועם קבלת התוצאה הוא עורך אותה בצורה נוחה על המסך.

- **תכנות עם פקודות SQL (SQL Programming):** שיטת הפעלה זו מיועדת עבור מהנדסי תוכנה מקצועיים המפתחים תוכניות יישום בשפת תכנות כלשהי וצריכים לגשת אל הנתונים המאוחסנים בבסיס הנתונים. מהנדס התוכנה משבץ את פקודות SQL בתוך שפת תכנות כלשהי המשמשת כשפה מארחת (Host Language). פקודות השפה המארחת משמשות לכתיבת הלוגיקה של היישום בעוד פקודות SQL המשובצות בתוכה משמשות לגישה אל בסיס הנתונים לצורך שליפת ועדכון הנתונים. השפה המארחת יכולה להיות שפת תכנות כגון Cobol, C++, Pascal, Visual Basic, Java או מחולל יישומים מדור רביעי כגון Power Builder, Delphi, Magic, או מחולל דוחות כגון SQR, Oracle SQL*ReportWriter וכד'.

  קיימות מספר דרכים לשיבוץ פקודות SQL בתוך שפה מארחת: שימוש בקדם-מהדר לשיבוץ פקודות SQL סטטיות (Embedded Static SQL) וביצוע הידור מראש של הפקודות; שימוש בקדם-מהדר לשיבוץ פקודות SQL הנבנות באופן דינמי במהלך ריצת התוכנית (Embedded Dynamic SQL); או שימוש בממשק תכנות (API) מיוחד המסופק על ידי מערכת RDBMS אשר מאפשר את הפעלת הפונקציות השונות של המערכת ללא צורך בהידור מראש.

בהמשך נסביר בצורה מפורטת כל אחת מהשיטות הללו. רוב הפקודות והתוצאות המתקבלות מוצגות מתוך הנחה שהן פועלות בסביבה אינטראקטיבית. לנושא התכנות עם פקודות SQL בתוך שפה מארחת מוקדש פרק נפרד.

סוגי פקודות SQL

שפת SQL מכילה מספר עשרות פקודות שונות שבאמצעותן ניתן להגדיר ולנהל את בסיס הנתונים. ניתן לחלק את כל פקודות השפה לחמש הקטגוריות, כפי שנפרט להלן.

פקודות להגדרת בסיס הנתונים (Data Definition)

פקודות אלו מאפשרות להגדיר טבלאות, עמודות בטבלה, טיפוס הנתונים בכל עמודה, אינדקסים לטבלה, טבלאות מדומות ועוד.

שם הפקודה	מהות הפקודה
CREATE SCHEMA	בניית סכימה חדשה
DROP SCHEMA	ביטול סכימה קיימת
CREATE DOMAIN	בניית מרחב ערכים
DROP DOMAIN	ביטול מרחב ערכים
CREATE TABLE	הגדרת טבלה חדשה בבסיס הנתונים
DROP TABLE	ביטול טבלה מבסיס הנתונים
ALTER TABLE	שינוי הגדרת הטבלה (הוספת עמודה, שינוי גודל עמודה וכד')
CREATE VIEW	הגדרת טבלה מדומה
DROP VIEW	ביטול טבלה מדומה
CREATE INDEX	הגדרת אינדקס חדש לטבלה
DROP INDEX	ביטול אינדקס קיים

הפקודות ליצירת/ביטול סכימה, יצירת/ביטול תחום ערכים עדיין אינן מיושמות בחלק מהמערכות המסחריות.

פקודות לטיפול בנתונים (Data Manipulation)

אוסף פקודות המאפשרות גישה אל הנתונים, שינוי תוכן הנתונים, ביטול והוספת נתונים.

שם הפקודה	מהות הפקודה
SELECT	שליפת נתונים מבסיס הנתונים
INSERT	הוספת שורה חדשה בטבלה
DELETE	ביטול שורה אחת או יותר מבסיס הנתונים
UPDATE	עדכון שורה אחת או יותר בבסיס הנתונים

פקודות בקרת גישה (Data Access Control)

אוסף פקודות המאפשרות למנהל בסיס הנתונים או למי שהוסמך מטעמו, להעניק הרשאות גישה לנתונים המאוחסנים בבסיס הנתונים, לקבוע לאיזה משתמשים מותר לגשת לאיזה טבלאות, איזה פעולות מותר למשתמשים לבצע על הטבלאות וכד'.

שם הפקודה	מהות הפקודה
GRANT	הענקת זכות גישה לבסיס הנתונים
REVOKE	סילוק זכות הגישה לבסיס הנתונים

פקודות בקרת תנועות (Transaction Control)

אוסף פקודות שתפקידן לתמוך בעיבוד תנועות תוך שמירה על שלמות ואמינות בסיס הנתונים.

שם הפקודה	מהות הפקודה
COMMIT	הודעה למערכת שהתנועה הסתיימה בהצלחה וכל העדכונים שבוצעו הם בלתי הפיכים. אם תנועה מסתיימת ללא ביצוע COMMIT – לדוגמה כתוצאה מתקלה באמצע התנועה – המערכת תבטל באופן אוטומטי את כל העדכונים שבוצעו מתחילת התנועה.
ROLLBACK	גלגול חזרה של כל העדכונים שבוצעו על ידי התנועה, כלומר ביטול כל ההשפעה שהיתה לתנועה על בסיס הנתונים. מאפשר לתוכנית היישום לבטל באופן יזום את השפעת התנועה.

פקודות מיוחדות לשילוב שפת SQL בשפה מארחת (Programmatic SQL)

אוסף פקודות המיועדות לשימוש בתוך שפה מארחת כלשהי. פקודות אלו מצטרפות לפקודות הבסיסיות ומאפשרות את עיבוד הנתונים בתוך שפה מארחת. העבודה עם שפה מארחת דורשת להכיר טבלה מיוחדת בשם **סמן** - **Cursor** - אשר תוסבר בהמשך.

שם הפקודה	מהות הפקודה
DECLARE	הצהרה על Cursor (סמן)
EXPLAIN	קבלת הסבר על תוכנית הגישה לנתונים כפי שהמהדר בנה
OPEN	פתיחת Cursor
CLOSE	סגירת Cursor
WHENEVER	לטיפול במצבי שגיאה בעקבות ביצוע פקודות SQL
DELETE CURRENT OF	לביטול שורות מתוך Cursor
UPDATE CURRENT OF	לעדכון שורה מתוך Cursor
FETCH	שליפת שורה מתוך Cursor
PREPARE	הכנת פקודת SQL לביצוע דינמי שלה
EXECUTE	ביצוע דינמי של פקודת SQL
DESCRIBE	תיאור דינמי של התוצאה המתקבלת

אבני הבניין של פקודת SQL

פקודת SQL מורכבת ממספר אבני בניין משותפות. נעמוד כאן על מספר תכונות משותפות לרוב הפקודות. נשתמש בדוגמה כדי להמחיש את השימוש באבני הבניין (את משמעות הפקודה נסביר בשלב מאוחר יותר):

```
SELECT   COURSE_ID, STUDENT_ID, GRADE
FROM     GRADES
WHERE    COURSE_ID = 'M-100'  AND
         GRADE BETWEEN 50 AND 75
```

- **אי רגישות לגודל האות (Case Insensitivity):** פקודות SQL נכתבות באותיות לועזיות. השפה עצמה אינה רגישה לסוג האותיות שבו נכתבות הפקודות, כלומר ניתן לכתוב את הפקודות באותיות רישיות (גדולות, Upper Case), אותיות רגילות (קטנות, Lower Case) או שילוב כלשהו שלהם. מסיבה זו מקובל לומר שהשפה היא Case Insensitive, כלומר לא רגישה לסוג האות. מבחינת השפה אין הבדל אם נרשום SELECT או select, או אפילו SELecT. לשם הבהירות, נשתמש באותיות גדולות לכתיבת הפקודות, אולם אין חובה לנהוג כך. הרגישות לגודל האות קיימת בין גרשיים. לדוגמה קיים שוני בין כתיבת 'M-100' = COURSE_ID לבין כתיבת 'm-100' = COURSE_ID (כיתה M-100, אינה זהה לכיתה m-100).
- **שם הפקודה (Statement Name):** כל פקודה מתחילה עם שם הפקודה שהוא פועל (Verb) המתאר את הפעולה שהפקודה מבצעת. לדוגמה, SELECT היא פקודה המבצעת בחירה של שורות מתוך בסיס הנתונים, CREATE TABLE היא פקודה להגדרת טבלה חדשה, DELETE היא פקודה לביטול שורות, INSERT היא פקודה להוספת שורה חדשה בטבלה וכד'.

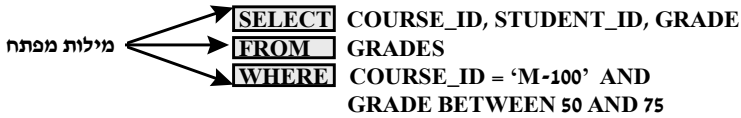
שם הפקודה →

```
SELECT   COURSE_ID, STUDENT_ID, GRADE
FROM     GRADES
WHERE    COURSE_ID = 'M-100'  AND
         GRADE BETWEEN 50 AND 75
```

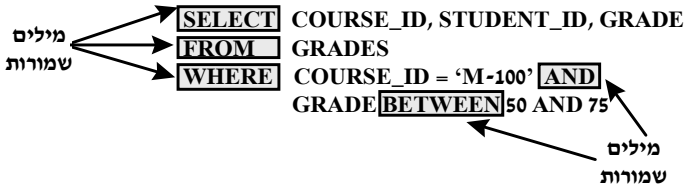
- **משפט (Clause):** כל פקודה יכולה להיות מורכבת ממשפט אחד או יותר. בדוגמה שהצגנו, הפקודה מורכבת משלושה משפטים: משפט SELECT הקובע אילו עמודות יש להציג, משפט FROM הקובע אילו טבלאות משתתפות בשאילתה ומשפט WHERE המגדיר תנאי לוגי שעל השורות הנשלפות לקיים. כל משפט יכול להיכתב בשורה אחת או יותר, ומכאן שפקודה ו/או משפט אחד יכולים להיכתב במספר שורות. בדוגמה המופיעה בהמשך, משפטים 1 ו-2 כתובים בשורה אחת ומשפט 3 כתוב בשתי שורות, למשל. חלק מהמשפטים הם חובה וחלק מהם רשות (אופציונליים).

```
SELECT   COURSE_ID, STUDENT_ID, GRADE   ← משפט 1
FROM     GRADES                         ← משפט 2
WHERE    COURSE_ID = 'M-100'  AND       ← משפט 3
         GRADE BETWEEN 50 AND 75
```

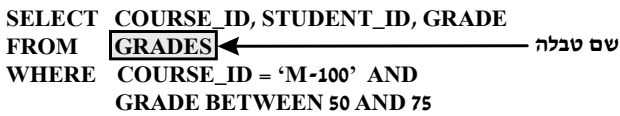
❖ **מילת מפתח (Keyword):** כל משפט מתחיל במילת מפתח כגון SELECT, FROM או WHERE. לכל משפט המתחיל במילת מפתח יש מבנה פנימי מוגדר.



❖ **מילים שמורות (Reserved Words):** לשפת SQL אוסף של מילים שמורות, המשמשות את מהדר השפה בפיענוח משמעות הפקודה, ולכן אסור להשתמש בהן למטרות אחרות. בדוגמה מופיעות מילים שמורות אלו: SELECT, FROM, WHERE, AND, BETWEEN. אסור להשתמש במילים שמורות עבור שמות של עמודות וטבלאות. לדוגמה, אסור לקרוא לעמודה מסוימת בשם BETWEEN. דוגמאות נוספות למילים שמורות: UNIQUE, INDICATOR, PROCEDURE, FETCH וכך הלאה. תקן SQL2 מגדיר כ-300 מילים שמורות.



❖ **שמות טבלאות (Tables Names):** במשפטים מסוימים חובה לציין את שמות הטבלאות עליהן פועלת הפקודה. לדוגמה, במשפט המתחיל במילת המפתח FROM בפקודה SELECT, חובה לציין את שמות הטבלאות שמהן יש לשלוף את השורות. על פי תקן SQL2, שם טבלה יכול להיות באורך תו אחד ועד 128 תווים, חייב להתחיל באות ולא יכול להכיל רווחים או סמנים מיוחדים. במציאות, המערכות המסחריות השונות קובעות כללים שונים מהתקן לגבי שמות טבלאות. למטרות ניידות מומלץ לשמור על שמות קצרים ככל האפשר.



❖ **שמות עמודות (Columns Names):** במשפטים מסוימים חובה לציין את שמות העמודות שפקודת SQL פועלת עליהן. כללי הכתיבה החלים על שמות טבלאות חלים גם על שמות עמודות. במידת הצורך ניתן להקדים את שם העמודה בשם הטבלה כדי לציין במפורש את הטבלה שהעמודה שייכת לה, ובמיוחד במקרים שבהם שמות עמודות זהים נמצאים בטבלאות שונות. לדוגמה, אם בשתי טבלאות שונות מופיעה עמודה בשם DEP_ID יש להשתמש במציין (Qualifier) שיקבע לאיזו עמודה (מאיזו טבלה) המשפט מתייחס, למשל GRADES.DEP_ID.

```
SELECT  COURSE_ID, STUDENT_ID, GRADE
FROM    GRADES
WHERE   COURSE_ID = 'M-100' AND
        GRADE BETWEEN 50 AND 75
```

שם עמודה

❖ **קבועים (Constants):** במשפטים מסוימים מותר שיופיעו קבועים נומריים או אלפאנומריים. לדוגמה, בפקודה SELECT במשפט WHERE המגדיר את התנאים הלוגיים לשליפת שורות, ניתן להשתמש בקבועים כגון **100**, **C-200** או **חיפה**. שפת SQL דורשת שקבוע אלפאנומרי יירשם כאשר הוא עטוף בגרש משני הצדדים. קיימות מערכות מסחריות התומכות גם בגרשיים כסימן העוטף קבוע אלפאנומרי. שפת SQL תומכת גם בקבועים נומריים שלמים (Integer) ומספרים עשרוניים (Floating Point).

```
SELECT  COURSE_ID, STUDENT_ID, GRADE
FROM    GRADES
WHERE   COURSE_ID = 'M-100' AND
        GRADE BETWEEN 50 AND 75
```

❖ **קבועים סימליים (Symbolic Constants):** נוסף לקבועים המקובלים מאפשרת שפת SQL שימוש במספר קבועים בעלי משמעות מיוחדת. משתנים אלה מנוהלים על ידי מערכת RDBMS וניתן לפנות אליהם באמצעות פקודות SQL. לדוגמה, תקן SQL2 מגדיר משתנים סימליים כגון CURRENT DATE המחזיר את התאריך הנוכחי, CURRENT TIME המחזיר את הזמן של המחשב, USER המחזיר את שם המשתמש וכד'.

טיפוסי נתונים (Data Types)

תקן SQL2 מגדיר אוסף של טיפוסי נתונים בהם מערכת RDBMS חייבת לתמוך ולאפשר את ניהולם בתוך בסיס הנתונים. לכל עמודה בטבלה בבסיס הנתונים יש להגדיר את טיפוס הנתונים. הטבלה הבאה מציגה את טיפוסי הנתונים הנתמכים:

טיפוס הנתונים	תיאור
CHAR (N)	מחרוזת תווים בעלת אורך קבוע של n תווים. המחרוזת יכולה להכיל כל ערך: אותיות, ספרות וסימנים מיוחדים.
VARCHAR (N)	מחרוזת תווים בעלת אורך משתנה. N הוא האורך המירבי.
DEC (N,M)	מספר עשרוני בעל n ספרות שמתוכן m ספרות אחרי הנקודה העשרונית. לדוגמה: DEC(7,2) הינו מספר בן 7 ספרות כאשר 5 מציינות ערכים שלמים ו-2 ספרות הן עבור השבר העשרוני.
INTEGER	מספר שלם התופס מילה שלמה במחשב, ולכן יכול להגיע לערכים גבוהים מאוד.

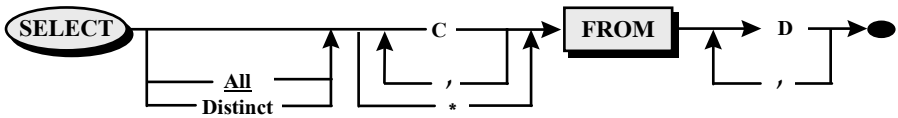
SMALLINT	מספר שלם התופס חצי מילה במחשב ולכן יכול להכיל ערכים קטנים יותר מאשר INTEGER.
BIT (N)	מחרוזת סיביות באורך קבוע של n.
BIT VARYING (N)	מחרוזת סיביות באורך משתנה, כאשר n הוא האורך המרבי.
DATE	תאריך קלנדרי. נתמכים מספר פורמטים שונים של תאריכים. לדוגמה: YYYY/MM/DD.
TIME	זמן שעון במבנה HH:MM:SS.
TIMESTAMP	חותמת זמן המכילה שילוב של התאריך והשעה.
BOOLEAN	משתנה היכול להכיל ערך בינארי (כן/לא).
BLOB	קיצור של Binary Large Object. משתנה המסוגל להכיל אובייקט גדול מאוד כמו תמונה, הקלטה של קול, מפה וכד'.
MONEY	משתנה המכיל ערכי כסף. ניתן להגדיר את המבנה ואת צורת ההצגה של הסכום, כמו למשל עם סימן $ בצד הנכון.

למרות קיום התקן של שפת SQL, הנושא "טיפוסי נתונים" הינו אחד המורכבים ביותר. תורמים לכך הרמה גבוהה של חוסר תאימות בייצוג הנתונים על ידי המערכות המסחריות השונות, השוני בצורת ייצוג הנתונים בין מחשבים שונים, והמגבלות החלות על טיפוסי הנתונים הללו. מומלץ לבחון סוגיה זו היטב לפני שמחליטים על ניוד יישומים מבסיס נתונים מסוים לבסיס נתונים אחר.

שיטה לתיאור מבנה פקודות SQL

קיימות מספר שיטות לתיאור המבנה העקרוני של פקודות מחשב בכלל, ושל פקודות SQL בפרט. שיטה נפוצה לתיאור פקודות נקראת **BNF** (Backus Naur Form) והיא משתמשת במספר סימנים כדי לתאר את הרכיבים שחייבים להופיע בפקודה, הרכיבים שהם רשות, הרכיבים שיכולים לחזור בפקודה ועוד.

שיטה אחרת לתיאור מבנה פקודות מבוססת על תרשים תחביר (Syntax Diagram) המתאר באופן גרפי את האפשרויות השונות לבניית הפקודה. בהמשך נשתמש בשיטה הגרפית לתיאור תחביר הפקודה. נציג כאן את העקרונות של תרשימים אלה באמצעות דוגמה פשוטה ולא מלאה של הפקודה SELECT.

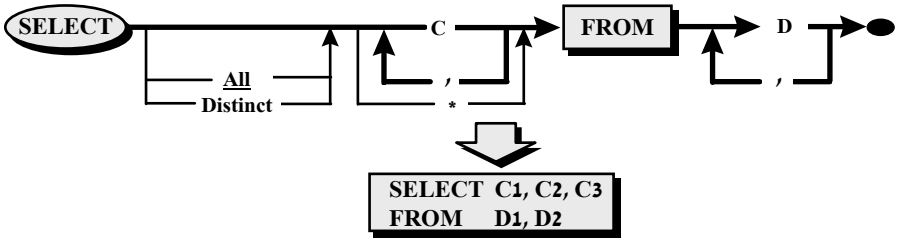


תרשים 7.1: תרשים מבנה עקרוני של הפקודה SELECT.

העיקרון של הרכבת פקודה חוקית הוא מעקב אחר המסלולים האפשריים השונים בתרשים. כל תרשים תחביר מתחיל בשם הפקודה בתוך אליפסה. כל משפט בתוך הפקודה מתחיל במילת המפתח בתוך מלבן. התנועה היא בהתאם לכיוון החץ קדימה ובחלק מהמקרים גם אחורה כדי לבטא חלקים שמותר לחזור עליהם. לסדר התנועה בתרשים יש משמעות ולא ניתן להשתמש בסדר שונה. קו תחתי מבטא ברירת מחדל. כל מסלול מסתיים על ידי העגול השחור.

מתוך התרשים שלמעלה נובעים מספר מבנים חוקיים אפשריים של הפקודה. להלן מספר דוגמאות כאשר הקו העבה מציין את המסלול שנבחר.

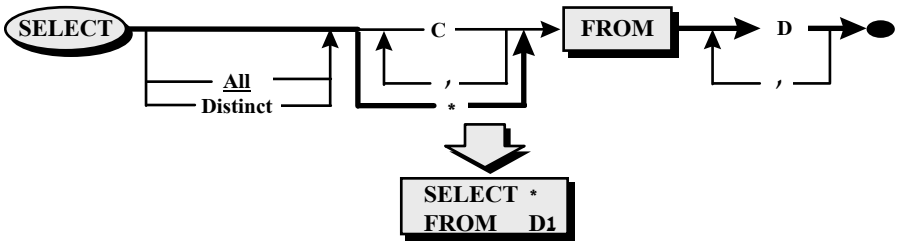
דוגמה א':



תרשים 7.2: דוגמה למבנה אפשרי של פקודה.

נשים לב שהביטויים C ו-D חוזרים על עצמם, כאשר פסיק מפריד ביניהם.

דוגמה ב':

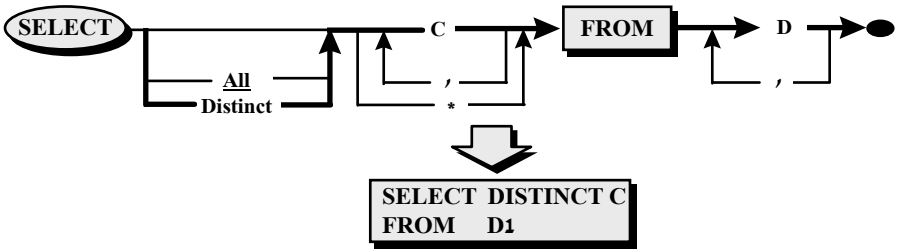


תרשים 7.3: דוגמה למבנה אפשרי של פקודה.

בדוגמה זו בחרנו במסלול הבוחר את הכוכבית (*) במקום הביטוי C.

דוגמה ג':

בדוגמה זו (תרשים 7.4) בחרנו באחת משתי האפשרויות All או Distinct לפני הביטוי C. נשים לב ש-All מסומן בקו תחתי והמשמעות היא שזו ברירת המחדל.



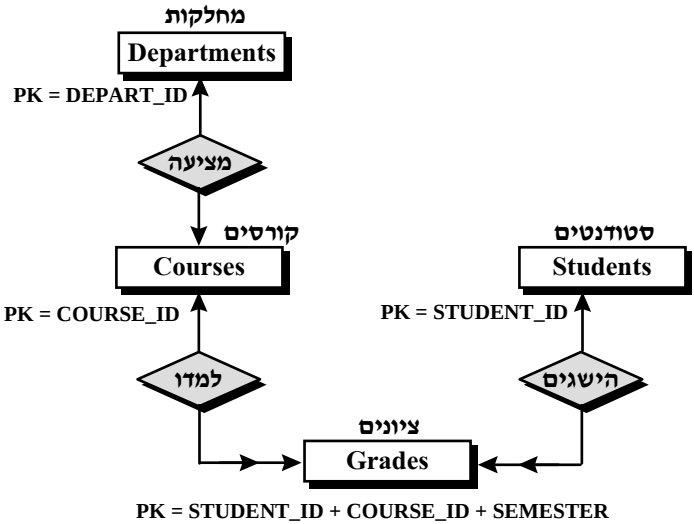
תרשים 7.4: דוגמה למבנה אפשרי של פקודה.

בסיס נתונים לדוגמה

תרשים 7.6 מציג את בסיס הנתונים לדוגמה שישמש אותנו להדגמת שפת SQL. לצורך הדוגמאות נניח שהטבלאות של בסיס הנתונים כבר מוגדרות. בהמשך נלמד כיצד להגדיר טבלאות ולהכניס בהן תוכן. בסיס הנתונים מורכב מארבע טבלאות:

- **טבלת מחלקות:** מכילה את נתוני המחלקות האקדמיות במכללה. לכל מחלקה מנוהלים הנתונים הבאים: קוד המחלקה, שם המחלקה ושם ראש המחלקה. המפתח העיקרי של הטבלה הוא קוד המחלקה.
- **טבלת קורסים:** מכילה את נתוני הקורסים השונים המועברים על ידי המחלקות האקדמיות. לכל קורס מנוהלים הנתונים הבאים: מספר הקורס, שם הקורס, סוג הקורס המציין אם הקורס הוא מסוג שיעור, מעבדה או סמינר ומספר נקודות הזכות שהסטודנט מקבל אם הוא מסיים בהצלחה את הקורס. המפתח העיקרי של הטבלה הוא מספר הקורס.
- **טבלת סטודנטים:** מכילה את נתוני הסטודנטים שנרשמו ללימודים במכללה. לכל סטודנט מנוהלים הנתונים הבאים: מספר סטודנט, שם סטודנט ועיר המגורים שלו. המפתח העיקרי של הטבלה הוא מספר סטודנט.
- **טבלת ציונים:** מכילה את הציונים שסטודנטים קיבלו בקורסים בהם למדו. הנתונים המנוהלים בטבלה הם: מספר הסטודנט, מספר הקורס, הסמסטר שבו הסטודנט למד בקורס, מועד הבחינה שבה קיבל את הציון, הציון הסופי בקורס והציון הסמסטריאלי. אם הסטודנט נבחן בשני מועדים שונים, רק המועד האחרון שלו נשמר. המפתח העיקרי של הטבלה הוא צירוף של מספר סטודנט, מספר קורס וסמסטר.

תרשים 7.5 מציג תרשים ישויות-קשרים של בסיס הנתונים לדוגמה, ותרשים 7.6 מציג את טבלאות בסיס הנתונים הזה.



תרשים 7.5: מודל ישויות-קשרים של בסיס נתונים לדוגמה.

ארבע הטבלאות של בסיס הנתונים לדוגמה, המפורטות בתרשים 7.6 (חלקים א' ו-ב') מכילות נתונים באנגלית מכיון שקל ופשוט יותר להציג את הדוגמאות בצורה זו. עם זאת, המערכות המסחריות הנמכרות בארץ **מאפשרות כמובן להכניס טקסט בעברית** לכל עמודה ותומכות ביישור השדה לימין ומיון לפי עמודות המכילות ערכים בעברית.

Departments ***מחלקות***

DEPART	NAME	HEAD
קוד	מחלקה	ראש מחלקה
CS	Computer Science	Dr. Israel
MT	Mathematics	Prof. Levy
BS	Business	Dr. Eyal
CH	Chemistry	Prof. Doron

Courses ***קורסים***

COURSE_ID	COURSE_NAME	TYPE	POINTS	DEPARTMENT_ID
מס. קורס	שם קורס	סוג קורס	נק. זכות	קוד מחלקה
C-200	Programming	LAB	4	CS
C-300	Pascal	LAB	4	CS
C-55	Data Base	CLASS	3	CS
M-100	Linear Algebra	CLASS	3	MT
M-200	Numeric Analysis	CLASS	3	MT
B-10	Marketing	CLASS	2	BS
B-40	Operations Res.	SEMIN	3	BS

תרשים 7.6: בסיס נתונים לדוגמה (חלק א').

Students *סטודנטים*

STUDENT_ID	NAME	CITY
מס. סטודנט	שם סטודנט	עיר
105	Moshe	Haifa
210	Dan	Tel Aviv
107	Eyal	Tel Aviv
110	Ran	Haifa
245	Yoel	Haifa
240	Ayelet	Tel Aviv
200	David	Tel Aviv
310	Tova	Jerusalem

Grades *ציונים*

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE	GRADE_SEM
מס. סטודנט	מס. קורס	סמסטר	מועד בחינה	ציון סופי	ציון סמסטר
105	C-55	SUM1998	A	58	70
210	M-100	AUT1999	A	90	90
105	M-100	SUM1998	B	75	50
105	C-200	AUT1999	A	90	85
210	C-200	AUT1999	A	85	80
210	B-10	WIN1999	A	78	50
105	B-40	WIN1999	B	70	70
245	M-100	AUT1999	A	90	80
245	B-10	AUT1999	A	80	70
200	C-200	AUT1999	B	78	50
200	B-10	AUT1999	A	70	65
245	B-40	WIN1998	A	85	95
200	M-100	SUM1998	B	90	90
310	M-100	SUM1998	A	65	100

תרשים 7.6: בסיס נתונים לדוגמה (חלק ב')

שאלות חזרה ותרגילים

שאלות חזרה

1. מהן הדרכים השונות בהן ניתן להפעיל את שפת SQL ?
2. הסבר מהן הקטגוריות השונות של פקודות SQL.
3. סקור את הסיבות העיקריות שהפכו את שפת SQL לשפת הנתונים הנפוצה ביותר.
4. מהם ההבדלים העיקריים בין שפת SQL לבין שפות הגישה לנתונים של בסיסי נתונים, המבוססים על מודל רישתי או היררכי?
5. ענה בכן/לא על המשפטים הבאים:
   א. משפט SQL אחד מורכב ממספר פקודות SQL.
   ב. פקודת SQL אחת מורכבת ממספר משפטי SQL.
   ג. כל משפט SQL חייב להתחיל במילת מפתח.
   ד. כל משפט SQL חייב להירשם בשורה נפרדת.
   ה. שם פקודת SQL חייב להופיע במקום כלשהו במשפט SQL הראשון.
   ו. ניתן לרשום את פקודות SQL באותיות קטנות או גדולות.
   ז. שפת SQL תומכת במספר בלתי מוגבל של טיפוסי נתונים שונים.

תרגילים

1. תאר בצורה גרפית את פקודת SQL הבאה:

```
1. SELECT  A, B, C, D
2. FROM    TAB1, TAB2
3. WHERE A = 100 AND B BETWEEB 200 AND 300
4. ORDER BY A
```

2. הסבר מהי מילת המפתח בפקודה שבתרגיל הקודם, מהם המשפטים מהם היא מורכבת, מהם המשתנים המופיעים בה, אילו טבלאות מופיעות בטבלה ומהם הקבועים המופיעים בפקודה.

פרק 8

פקודות לטיפול בנתונים (Data Manipulation)

1. מבוא
2. הפקודה SELECT
3. שאילתות לטבלה אחת (Single Table Queries)
4. שאילתות מקובצות (Grouped Queries)
5. שאילתות למספר טבלאות (Multi Table Queries)
6. צירוף טבלאות חיצוני (Outer Join)
7. צירוף טבלה אל עצמה (Self Join)
8. מכפלה קרטזית בין טבלאות (Cartesian Product)
9. תת-שאילתות (Sub Queries)
10. איחוד תוצאות של שאילתות (Union)
11. חיתוך תוצאות של שאילתות (Intersect)
12. פקודות לעדכון בסיס הנתונים
13. טיפול בערכים חסרים (Null Values)
14. סיכום
15. שאלות חזרה ותרגילים

מבוא

פרק זה מציג פקודות SQL המאפשרות את ניהול הנתונים בבסיס הנתונים. השם הכללי של פקודות אלו הוא **שפה לטיפול בנתונים** – **DML** (Data Manipulation Language). בפרק זה מוצגות ארבע הפקודות הבסיסיות לטיפול בנתונים:

- SELECT – שליפת נתונים מתוך טבלאות קיימות.
- INSERT – הוספת שורות חדשות בטבלה.
- UPDATE – עדכון הערכים בשורות בטבלה.
- DELETE – ביטול שורות קיימות בטבלה.

למרות שמספר הפקודות לטיפול בנתונים קטן, הרי שמספר האפשרויות השונות בכל פקודה הוא גדול. דבר זה בולט בעיקר בפקודה SELECT. בנוסף לארבע הפקודות הבסיסיות האלו, מוצגות שתי פקודות מיוחדות המטפלות בטבלאות שלמות בבת אחת, פקודות ייחודיות למודל הטבלאי:

- UNION – לאיחוד התוצאות של מספר שאילתות.
- INTERSECT – לחיתוך התוצאות של מספר שאילתות.

מתכונת הצגת הפקודות הינה אחידה, ומורכבת משלושה קטעים אלה:

- **עיקרון הפעולה:** קטע זה מסביר כיצד הפקודה פועלת. ההסבר מציג את עיקרון הפעולה בלבד, בידיעה שצורת המימוש במערכות RDBMS מסחריות יכולה להיות שונה.
- **תחביר הפקודה:** קטע זה מתייחס למבנה התחבירי של הפקודה. תחביר הפקודה מוצג באמצעות תרשים תחביר המציג באופן גרפי את האפשרויות השונות לבניית הפקודה. קטע זה מסביר את החלופות השונות בבניית הפקודה, מתייחס לברירות המחדל של הפקודה ומתאר את האילוצים השונים הקיימים בבניית הפקודה.
- **דוגמאות:** קטע זה מציג מספר דוגמאות של הפקודה. הפקודות מבוססות על בסיס הנתונים לדוגמה שהוצג בפרק 7. כל פקודה מוצגת יחד עם התוצאה שלה, לו הופעלה על בסיס הנתונים לדוגמה.

בנוסף להצגת פקודות שפת SQL, מוצג דיון נפרד לסוגית הטיפול בערכים חסרים (Null Values). נושא זה הוא בעל חשיבות רבה במודל הטבלאי המבוסס על קשרים בין טבלאות הנוצרים כתוצאה מערכים זהים. עמודות המכילות ערכים חסרים מציבות אתגר מיוחד בפני שפת SQL ומחייב אותה להרחיב את הלוגיקה הבוליאנית הרגילה המבוססת על שני ערכים בלבד, אמת או שקר, ללוגיקה המבוססת על שלושה ערכים: אמת, שקר ו"לא ידוע". שפת SQL יכולה להתייחס לערכים חסרים הן בזמן הוספת שורות חדשות והן בעת ביצוע שאילתות ושימוש באופרטורים מתקדמים.

בפרק זה

- ❖ נציג את קבוצת הפקודות לטיפול בנתונים ואת הפקודה SELECT.
- ❖ נסביר מהי שאילתה ואיזה סוגי שאילתות מתאימים לטיפול בנתונים.
- ❖ נראה אפשרויות הצירוף השונות של טבלאות ומכפלה קרטזית שלהן.
- ❖ נציג כיצד להגדיר תת-שאילתות ונראה כיצד לבצע איחוד וחיתוך של תוצאות.
- ❖ נציג את הפקודות הבסיסיות לעדכון בסיס הנתונים ואפשרויותיהן.
- ❖ נסביר מהם ערכים חסרים, וחשיבות הטיפול בהם בעת בניית בסיס נתונים ועדכונו.

הפקודה SELECT

הפקודה SELECT היא הפקודה העיקרית של שפת SQL. היא מיועדת לשלוף ערכים מתוך בסיס הנתונים ומגלמת את העוצמה של השפה, מצד אחד ואת הפשטות, מצד שני. הדרך הטובה ביותר להבין את הפקודה היא על ידי דוגמאות. הדוגמאות בהמשך מוצגות "מהקל אל הכבד": בתחילה, פקודות השולפות נתונים מטבלה בודדת ובהמשך, פקודות השולפות נתונים ממספר טבלאות בו-זמנית.

מבנה כללי של הפקודה SELECT

המבנה הכללי של הפקודה SELECT:

SELECT	***List of Columns Names***
FROM	***List of Tables Names***
WHERE	***Predicate***
GROUP BY	***List of Column Names***
HAVING	***Predicate***
ORDER BY	***List of Column Names***

כפי שניתן לראות, הפקודה SELECT מורכבת משישה משפטים שונים, שרק שני הראשונים הם חובה ואילו הארבעה הנוספים הם רשות. אם כותבים את משפטי הרשות, הם חייבים להופיע בסדר הרשום למעלה.

להלן תיאור כללי של כל משפט המרכיב את הפקודה:

- ❖ **משפט SELECT**: מגדיר את שמות העמודות שיש לשלוף ולהציג. הרשימה יכולה להכיל שמות של עמודות מטבלה אחת או יותר. בנוסף לשמות של עמודות, ניתן להגדיר במשפט זה גם חישובים על עמודות (כמו למשל להכפיל עמודה בערך כלשהו), ניתן לבצע חישובים בין עמודות ולמעשה להציג עמודה מדומה שלא קיימת בטבלה. ניתן להגדיר פונקציות הפועלות על עמודה (כמו למשל, חישוב הממוצע של עמודה, הסיכום של עמודה וכד'), ניתן להגדיר שם נוסף לעמודה ועוד. הסדר שבו הפונקציות רשומות הוא הסדר שבו הן תוצגנה.

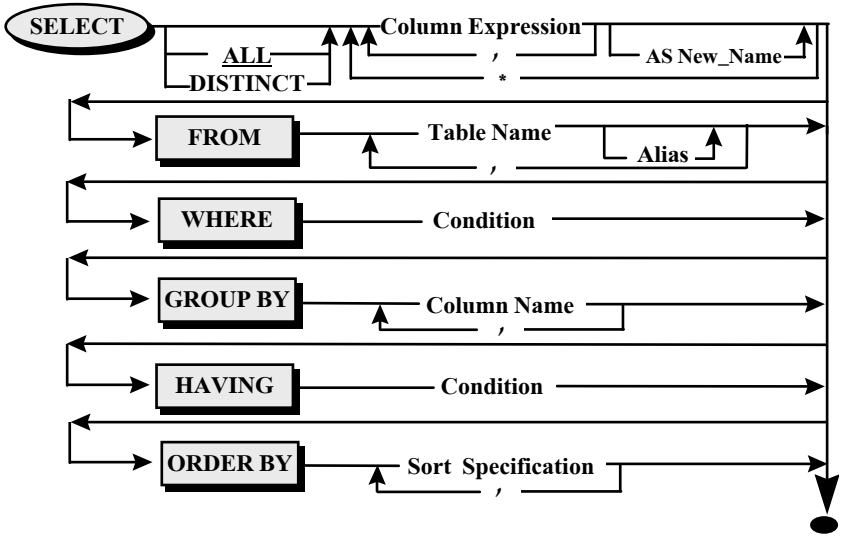
- **משפט FROM:** משפט זה מגדיר את שמות הטבלאות שמהן יש לבצע את השליפה. ניתן להגדיר במשפט זה גם שמות נרדפים (Alias, ראה הסבר בעמ' 273). אין מגבלה למספר הטבלאות המופיעות במשפט זה.
- **משפט WHERE**: משפט זה מאפשר הגדרת תנאי לוגי (Predicate) שיקבע איזה שורות תישלפנה מתוך הטבלאות המופיעות במשפט FROM. התנאי הלוגי יכול להכיל:
  - אופרטורים השוואתיים, כגון =, >, <, =<, =>, <>.
  - אופרטורים לוגיים, כגון AND, OR, NOT.
  - בדיקת קיום ערך בתוך קבוצה סגורה של ערכים על ידי שימוש ב-IN.
  - בדיקת קיום ערך בתוך טווח ערכים על ידי שימוש ב-BETWEEN וציון הערך הנמוך והגבוה בטווח.
  - בדיקת קיום מחרוזת תווים בתוך ערכים של עמודה על ידי שימוש ב-LIKE.
  - בדיקה אם ערך עמודה הוא ריק על ידי שימוש ב- IS NULL.
  - בדיקת קיום ערך של עמודה מול קבוצת ערכים המוחזרת כתוצאה מהפעלת תת-שאילתה על ידי שימוש ב- EXISTS, ANY או SOME.
- **משפט GROUP BY:** מאפשר הקבצת שורות שבהן מופיעים בעמודות מסוימות ערכים זהים.
- **משפט HAVING**: מאפשר הגדרת תנאי לוגי אותו יקיימו השורות המקובצות בלבד.
- **משפט ORDER BY**: מאפשר הגדרת סדר המיון שבו תוצגנה השורות הנשלפות.

כמו כל שפת תכנות מודרנית, גם שפת SQL היא בעלת מבנה כתיבה חופשי למדי ומספר מגבלות מעטות. שני המשפטים הראשונים חייבים להופיע בכל פקודה. שאר המשפטים הם רשות, אבל אם הם מופיעים – הם חייבים להופיע בסדר הרשום בדוגמה. ניתן לכתוב את המשפטים ברצף, לשבור אותם לשורות שונות בכל מקום בתוך השורה, כל עוד לא שוברים שם כלשהו באמצע. צורת הכתיבה המופיעה למעלה היא צורת הכתיבה הנפוצה אולם היא אינה חובה ונבחרה אך ורק לצורך נוחות ובהירות.

חשוב להדגיש שהפקודה SELECT של שפת SQL שונה באופן מהותי מהפקודה SELECT של האלגברה הטבלאית. כפי שנראה בהמשך, SELECT בשפת SQL מכילה בתוכה את כל שלושת האופרטורים של האלגברה הטבלאית: SELECT, PROJECT ו- JOIN.

בכל הדוגמאות מופיע מספור לכל שורה של הפקודה, למען הבהירות וכדי להקל על ההסבר המופיע בחלק מהדוגמאות. **מספור זה מיועד אך ורק למען הבהירות ואינו מהווה חלק מהשפה.**

תרשים תחביר של הפקודה SELECT



תרשים 8.1: תרשים תחביר של הפקודה SELECT.

כפי שניתן לראות, המבנה של הפקודה SELECT מורכב יחסית, כי קיימות בו אפשרויות רבות. בהמשך נציג מיגוון דוגמאות המציגות את האפשרויות השונות. תחילה דוגמאות המתייחסות לטבלה אחת ובהמשך, דוגמאות המתייחסות לכמה טבלאות בפקודה אחת.

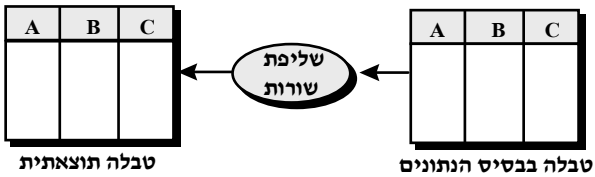
שאילתות לטבלה אחת (Single Table Queries)

שליפת כל העמודות וכל השורות

המקרה הפשוט ביותר הוא שליפת כל העמודות והשורות מטבלה מסוימת.

עיקרון הפעולה:

השאילתה שולפת את כל שורות הטבלה ובונה את טבלת התוצאה. סדר הופעת העמודות בטבלת התוצאה הוא בהתאם לסדר הופעתן בטבלה בבסיס הנתונים.



תרשים 8.2: עיקרון הפעולה לשליפת כל השורות וכל העמודות.

תחביר הפקודה:

תרשים 8.3: תרשים תחביר לשליפת כל השורות והעמודות מטבלה.

דוגמה : הצג את כל העמודות ואת כל השורות מהטבלה מחלקות.

```
1. SELECT *
2. FROM    DEPARTMENTS
```

DEPART	NAME	HEAD
CS	Computer Science	Dr. Israel
MT	Mathematics	Prof. Levy
BS	Business	Dr. Eyal
CH	Chemistry	Prof. Doron

נשים לב שהפקודה מורכבת משני משפטים: משפט SELECT המופיע בשורה הראשונה ומציין את העמודות המבוקשות, במקרה זה את כולן, ומשפט FROM המופיע בשורה השנייה ומציין את הטבלה שממנה יש לשלוף את הנתונים. הכוכבית המופיעה במשפט SELECT מציינת שיש לשלוף את **כל** העמודות של הטבלה. הסדר שבו תוצגנה העמודות הוא הסדר שבו הן מופיעות בתוך הטבלה, כלומר הסדר בעת הגדרת הטבלה על ידי מנהל בסיס הנתונים.

זוהי צורת רישום מקוצרת החוסכת את הצורך ברישום השמות של העמודות. רישום זה זהה לרישום הבא של אותה דוגמה:

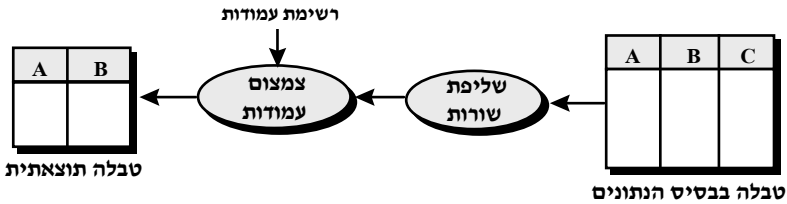
```
1. SELECT  DEPART, NAME, HEAD
2. FROM    DEPARTMENTS
```

שליפת שורות ועמודות מסוימות

כאשר מבקשים לשלוף רק עמודות מסוימות, יש לרשום את העמודות המבוקשות במשפט SELECT. הסדר שבו נרשום את שמות העמודות, הוא הסדר שבו הן תוצגנה על המסך.

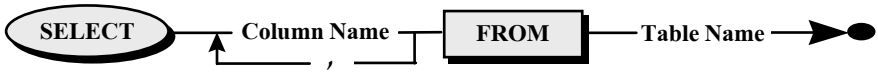
עיקרון הפעולה:

השאילתה שולפת את כל שורות הטבלה, בוחרת את העמודות המופיעות במשפט SELECT, מסדרת את העמודות לפי הסדר בו הן מופיעות במשפט SELECT ובונה את טבלת התוצאה.



תרשים 8.4: עיקרון הפעולה לשליפת כל השורות וחלק מהעמודות.

תחביר הפקודה:



תרשים 8.5: תרשים תחביר לשליפת עמודות מסוימות מטבלה.

לפי תקן ANSI לא ניתן לשלב בין בחירה מפורשת של עמודות, לבין בחירת כל העמודות על ידי שימוש בסימן הכוכבית.

דוגמה : הצג את שמות הסטודנטים ועיר המגורים שלהם.

```
1. SELECT  NAME, CITY
2. FROM    STUDENTS
```

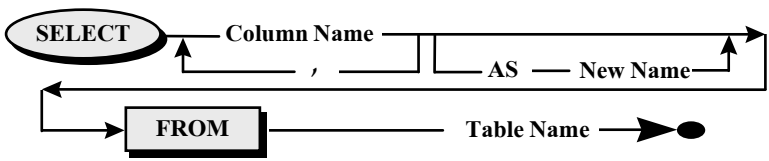
NAME	CITY
Moshe	Haifa
Dan	Tel Aviv
Eyal	Tel Aviv
Ran	Haifa
Yoel	Haifa
Ayelet	Tel Aviv
David	Tel Aviv
Tova	Jerusalem

זוהי הצורה בה רושמים בשפת SQL את פעולת ההיטל האלגברית, פעולה הבוחרת במספר עמודות מתוך כלל העמודות של הטבלה.

שינוי שם עמודה

בדרך כלל הכותרת של העמודה נלקחת מתוך ההגדרה של הטבלה בבסיס הנתונים. לפעמים קיים צורך לשנות את שם העמודה משיקולי בהירות בהצגת הטבלה התוצאתית. שפת SQL מאפשרת שינוי הכותרת המוצגת של העמודה על ידי שימוש בפרמטר AS ליד שם של עמודה והגדרת הכותרת החדשה של העמודה.

תחביר הפקודה:



תרשים 8.6: תרשים תחביר למתן שם חדש לעמודה.

דוגמה : הצג את שמות הסטודנטים. שם העמודה יהיה Student_Name.

```
1. SELECT  NAME AS STUDENT_NAME
2. FROM    STUDENTS
```

STUDENT_NAME
Moshe
Dan
Eyal
Ran
Yoel
Ayelet
David
Tova

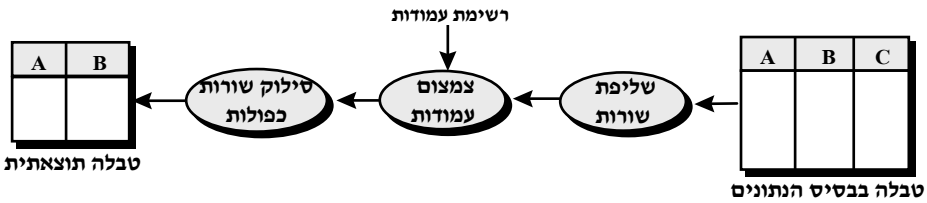
תרשים 8.7: שליפת עמודה אחת מטבלה.

שליפת שורות ללא הצגת שורות כפולות

על פי כללי המודל הטבלאי, פעולת ההיטל צריכה לבטל שורות כפולות במידה ונוצרות כאלו בטבלה התוצאתית. פעולת הביטול של שורות כפולות היא פעולה יקרה מבחינת צריכת משאבי המחשוב ולכן ברוב מערכות RDBMS המסחריות, ברירת המחדל היא להציג את השורות הכפולות. אם יש צורך במניעת הצגת שורות כפולות, יש לבקש זאת בצורה מפורשת. בדרך כלל סילוק שורות כפולות מחייב מיון של הטבלה התוצאתית וסריקתה למציאת שורות כפולות. פעולת מיון זו היא שהופכת את פעולת סילוק השורות הכפולות לפעולה יקרה מבחינת משאבי המחשב. לעיתים, הזמן הנדרש לביצוע המיון של טבלת התוצאה יכול להיות ארוך יותר מהזמן לביצוע השאילתה.

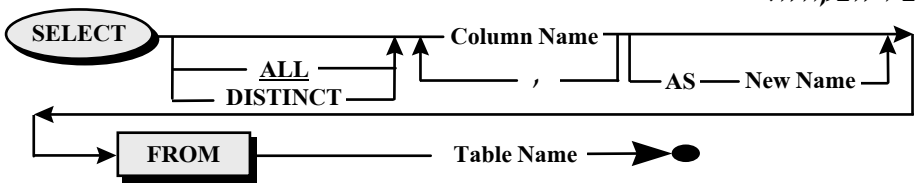
עיקרון הפעולה:

השאילתה שולפת את השורות, בוחרת את העמודות המופיעות במשפט SELECT, מסלקת שורות כפולות, מסדרת את העמודות על פי סדר הופעתן במשפט SELECT ובונה את טבלת התוצאה.



תרשים 8.8: עיקרון הפעולה לסילוק שורות כפולות.

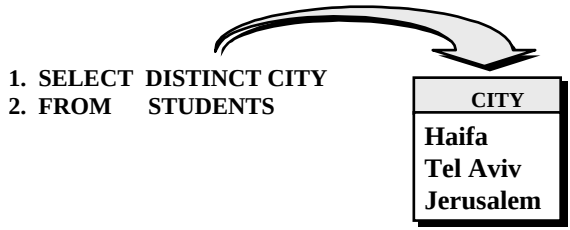
תחביר הפקודה:



תרשים 8.9: תרשים תחביר לשליפת שורות ללא שורות כפולות.

ברירת המחדל היא ALL, כלומר הצגת כל השורות שנשלפו. מניעת הצגת השורות הכפולות מתבצעת על ידי שימוש בפרמטר DISTINCT.

דוגמה : הצג את רשימת הערים בהן גרים סטודנטים, תוך הצגת כל עיר פעם אחת בלבד.



סדר הופעת השורות בטבלה התוצאתית הוא בהתאם למה שמופיע בבסיס הנתונים. אם נרצה להציג את השורות בסדר שונה נצטרך לבקש זאת במפורש ועל כך בהמשך.

בחירת שורות (Row Selection)

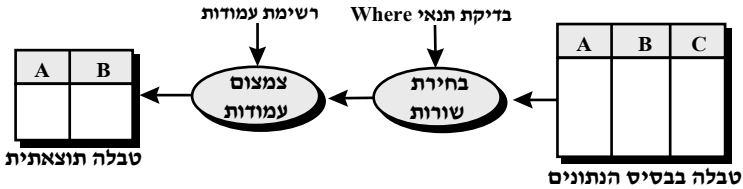
עד כאן התייחסנו לדוגמאות בהן היה צורך לשלוף את כל השורות של הטבלה. כמובן ששפת SQL מאפשרת לבחור קבוצה חלקית של השורות, קבוצה המקיימת תנאי לוגי כלשהו. פקודת ה SELECT מאפשרת הגדרת מיגוון של תנאים לוגיים שעל השורות לקיים כדי שהן תישלפנה. הפקודה תומכת במספר סוגים שונים של תנאים לוגיים :

- **בדיקה השוואה:** תנאי שבוחן אם מתקיים יחס כלשהו בין ערך של ביטוי אחד עם ערך של ביטוי אחר. התנאי יכול להיות פשוט ולבחון את היחס בין שני ביטויים בלבד, או מורכב – שבו נבחן היחס בין מספר ביטויים שונים ותוך שימוש באופרטורים בוליאניים כגון AND, OR ו-NOT לקישור בין הביטויים.
- **בדיקת טווח ערכים רציף:** תנאי שבוחן אם ערך של ביטוי כלשהו נמצא בתוך טווח ערכים רציף. הטווח מוגדר על ידי הערך הגבוה והערך הנמוך בטווח.
- **בדיקת קיום בקבוצת ערכים:** תנאי שבוחן אם ערך של ביטוי כלשהו מופיע בתוך קבוצת ערכים נתונה כלשהי.
- **בדיקת מחרוזת:** תנאי הבוחן אם עמודה כלשהי מכילה מחרוזת כלשהי של תווים.

❖ **בדיקת ערכים חסרים:** תנאי הבוחן אם ערך כלשהו של עמודה מכיל את הערך Null, המצביע על העובדה שערך העמודה אינו ידוע או חסר. לקראת סוף הפרק נציג דיון נרחב יותר בסוגיית הערכים החסרים.

עיקרון הפעולה:

השאילתה בוחרת את השורות המקיימות את התנאי המופיע במשפט WHERE, בוחרת את העמודות המופיעות במשפט SELECT, מסדרת את העמודות על פי הסדר המבוקש ובונה את הטבלה התוצאתית.

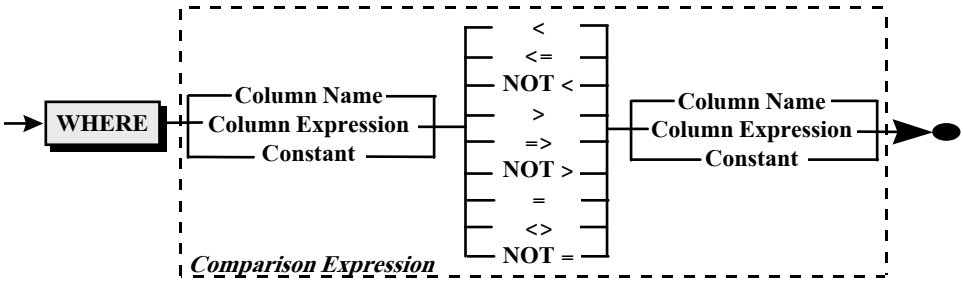


תרשים 8.10: עיקרון הפעולה לבחירת שורות

בדיקת השוואה (Comparison Test)

תנאי זה מאפשר שליפה של השורות המקיימות תנאי לוגי כלשהו, המוצג על ידי ביטוי השוואה (Comparison Expression). ביטוי ההשוואה מוגדר במשפט WHERE.

תחביר הפקודה:



תרשים 8.11: תרשים תחביר לשליפת שורות עם ביטוי השוואה.

ביטוי ההשוואה מורכב משלושה חלקים: אגף שמאל, אופרטור השוואה ואגף ימין. אגף ימין או שמאל של הביטוי יכולים להכיל שם עמודה (Column Name), ערך קבוע (Constant) או פעולה אריתמטית בין עמודות שונות או בין עמודות לערך קבוע (Column Expression). אופרטור ההשוואה יכול להיות אחד מאלה: =, <>, <, >, <=, =<. ניתן לרשום את אופרטור האי-שוויון בשתי צורות רישום שונות: או <> או NOT =.

התוצאה של כל ביטוי השוואה יכולה להיות אמת (True) או שקר (False). בתהליך זה תישלפנה רק שורות עבורן מתקבלת התוצאה **אמת**.

דוגמה א': הצג את כל העמודות של סטודנטים הגרים בחיפה.

```
1. SELECT  *
2. FROM    STUDENTS
3. WHERE  CITY = 'Haifa'
```

STUDENT_ID	NAME	CITY
105	Moshe	Haifa
110	Ran	Haifa
245	Yoel	Haifa

בדוגמה זו התנאי המופיע בשורה 3 מגדיר מבחן השוואתי בין ערך של שני ביטויים, ביטוי אחד המכיל שם של עמודה וביטוי שני המכיל ערך קבוע.

נדגיש כי זוהי צורת הרישום בשפת SQL: שילוב בין האופרטור SELECT האלגברי הבוחר שורות מסוימות, לבין האופרטור PROJECT האלגברי הבוחר עמודות מסוימות.

דוגמה ב' : הצג שמות סטודנטים הגרים בתל אביב.

```
1. SELECT  NAME
2. FROM    STUDENTS
3. WHERE  CITY = 'Tel Aviv'
```

NAME
Dan
Eyal
Ayelet
David

דוגמה ג' : הצג את מספר הסטודנט ומספר הקורס מטבלת הציונים בהם מתקיים התנאי שהמכפלה של הציון הסופי ב- 1.1 גדולה מציון הסמסטר.

```
1. SELECT  STUDENT_ID, COURSE_ID,
2.         GRADE, GRADE_SEM
3. FROM    GRADES
4. WHERE  GRADE * 1.1 > GRADE_SEM
```

STUDENT_ID	COURSE_ID	GRADE	GRADE_SEM
105	M-100	75	50
105	C-200	90	85
210	C-200	85	80
210	B-10	78	50
245	M-100	90	80
245	B-10	80	70
200	C-200	78	50
204	B-10	70	65

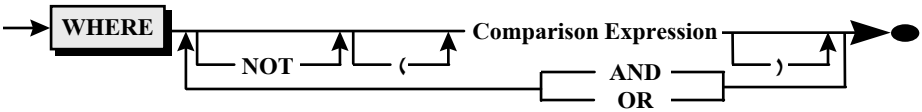
תרשים 8.12: שליפת נתונים על פי תנאי.

בדוגמה זו, ביטוי ההשוואה שבשורה 4 מכיל באגף השמאלי מכפלה בין עמודה וערך קבוע, באגף הימני – שם עמודה ובדיקה אם הערך באגף השמאלי גדול מהערך באגף הימני.

בדיקה השוואה עם תנאי בוליאני (Boolean Condition)

תנאי השליפה המופיע במשפט WHERE יכול להיות תנאי מורכב המכיל גם אופרטורים בוליאניים – AND, OR, NOT המופעלים על ביטויים השוואתיים. התוצאה של כל ביטוי השוואתי יכולה להיות אמת או שקר. יישלפו רק שורות המקיימות את התנאי הבולאני ונותנות תוצאה סופית של אמת. שימוש באופרטורים אלה מאפשר לבנות תנאי שליפה מורכבים. ניתן להשתמש בסוגריים כדי לבטא עדיפות בסדר ביצוע הפעולות.

תחביר הפקודה:



תרשים 8.13: תרשים תחביר לבדיקת תנאי בוליאני.

ביטוי המכיל אופרטורים בוליאניים מפוענח בצורה הבאה:

- הביטוי מפוענח משמאל לימין.
- ביטויים בסוגריים מפוענחים ראשונים.
- NOT קודם ל-AND ו-OR.
- AND קודם ל-OR.

מומלץ מאוד להשתמש בסוגריים כדי להסיר אי-בהירויות בסדר הפיענוח של ביטויים מורכבים.

דוגמה א': הצג את שמות ומספרי הקורסים לכל הקורסים שהם מסוג שיעור והמעניקים מעל 2 נקודות זכות ומועברים על ידי המחלקה למתמטיקה.

```
1. SELECT  COURSE_NAME, COURSE_ID
2. FROM    COURSES
3. WHERE   TYPE = 'Class' AND
4.         POINTS > 2 AND
5.         DEPARTMENT_ID = 'MT'
```

COURSE_NAME	COURSE_ID
Linear Algebra	M-100
Numeric Analysis	M-200

דוגמה ב': הצג את השם ואת מספר נקודות הזכות לכל הקורסים המועברים על ידי המחלקה למתמטיקה או למדעי המחשב המעניקים יותר משלוש נקודות זכות.

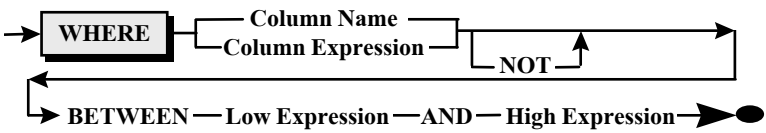
```
1. SELECT  COURSE_NAME, POINTS
2. FROM    COURSES
3. WHERE   (DEPARTMENT_ID ='MT'  OR
4.          DEPRTMENT_ID ='CS'  ) AND
5.          POINTS >3
```

COURSE_NAME	POINTS
Programming	4
Pascal	4

בדיקת טווח ערכים רציף (Range Test)

ניתן לבנות תנאי המתייחס לבדיקת קיום ערך כלשהו בתוך טווח ערכים רציף. טווח הערכים מבוטא על ידי הגדרת הערך הנמוך ביותר והערך הגבוה ביותר בטווח.

תחביר הפקודה:



תרשים 8.14: תרשים תחביר לבדיקת תחום ערכים.

טיפוסי הנתונים המשתתפים בהגדרת הטווח צריכים להיות תואמים לטיפוס הנתונים של העמודה. לדוגמה, אם העמודה הנבדקת היא בעלת טיפוס נתונים נומרי, גם הערך הגבוה והנמוך בטווח חייבים להיות נומריים.

דוגמה : הצג את מספר הקורס ואת מספר הסטודנט של כל הסטודנטים שקיבלו במועד א' ציון סופי בין 50 ל70-.

```
1.  SELECT  COURSE_ID, STUDENT_ID
2.  FROM    GRADES
3.  WHERE   TERM = 'A' AND
4.          GRADE BETWEEN 50 AND 70
```

COURSE_ID	STUDENT_ID
C-55	105
B-10	200
M-100	310

כמובן שניתן להשתמש גם בתנאי רגיל במקום הגדרת הטווח, כמודגם בהמשך.

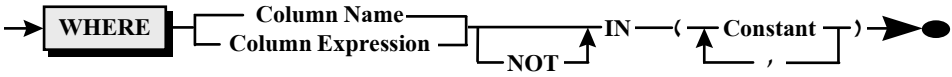
```
1.  SELECT  COURSE_ID, STUDENT_ID
2.  FROM    GRADES
3.  WHERE   TERM = 'A' AND
4.          GRADE >=  50 AND GRADE <= 70
```

הגדרת טווח היא למעשה צורת רישום נוחה וקומפקטית יותר. ניתן להשתמש גם בשילוב עם אופרטור השלילה – NOT BETWEEN – כדי למצוא ערכים שאינם בטווח.

בדיקת קיום ערך בתוך קבוצת ערכים (Set Membership Test)

כאשר רוצים לשלוף שורות שבהן עמודה כלשהי מכילה ערך אחד מבין קבוצת ערכים ידועה, ניתן להשתמש במבחן הקיום בקבוצה במקום בדיקת תנאי מול כל ערך בנפרד.

תחביר הפקודה:



תרשים 8.15: תרשים תחביר לבדיקת קיום ערך בקבוצה.

טיפוס הנתונים של העמודה חייב להיות תואם את טיפוס הנתונים המופיע בקבוצת הערכים. התקן אינו מגדיר את מספר הערכים המרבי שניתן לרשום בתוך הסוגריים.

דוגמה : הצג את מספר הקורס, מספר הסטודנט והציון שקיבלו סטודנטים בקורסים B-10, B-40 ו- C55.

```
1. SELECT COURSE_ID,
2.        STUDENT_ID, GRADE
3. FROM   GRADES
4. WHERE  COURSE_ID
5.        IN ('B-10', 'B-40', 'C-55')
```

COURSE_ID	STUDENT_ID	GRADE
C-55	105	58
B-10	210	78
B-40	105	70
B-10	245	80
B-10	200	70
B-40	245	85

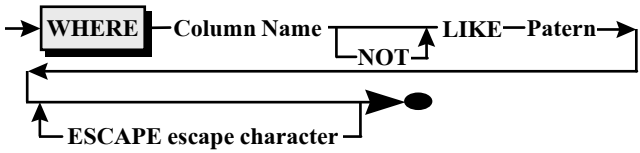
בדומה ל- BETWEEN, גם את בדיקת קיום הערך בקבוצה ניתן לבטא על ידי שימוש בתנאי מורכב ואופרטורים בולאניים. זוהי רק צורת רישום נוחה יותר, במיוחד אם יש לבדוק קיום בתוך קבוצה גדולה יחסית.

```
1. SELECT COURSE_ID, STUDENT_ID, GRADE
2. FROM   GRADES
3. WHERE COURSE_ID  = 'B-10' OR
4.        COURSE_ID = 'B-40' OR
5.        COURSE_ID = 'C-55')
```

בדיקת מחרוזת (Pattern Matching)

בדיקה זו מאפשרת הגדרת תנאי המאפשר שליפה של שורות שבהן ערך עמודה כלשהי מכיל מחרוזת תווים מסוימת. האופרטור LIKE מאפשר להגדיר את בדיקת המחרוזת הנדרשת. כמובן שהשימוש באופרטור זה נדרש רק כאשר מבקשים לבדוק קיום מחרוזת חלקית כלשהי ולא את כל המחרוזת. עבור בדיקת כל המחרוזת ניתן להשתמש בתנאי השוואתי פשוט.

תחביר הפקודה:



תרשים 8.16: תרשים תחביר לבדיקת מחרוזת תווים.

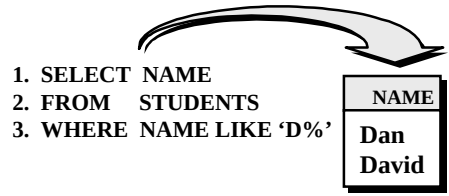
האופרטור LIKE מאפשר להגדיר את המחרוזת המבוקשת ולהשתמש בשני **תווי הכללה** (Wild Cards) מיוחדים:

- **אחוז** (%) המייצג אוסף כלשהו של תווים, ויכול להכיל תו אחד או יותר.
- **קו תחתי** (_) המייצג תו בודד כלשהו ומתאים למחרוזת בעלת תו אחד בלבד.

כל שאר התווים מייצגים את עצמם. נראה מספר דוגמאות המבהירות את השימוש בתווי הכללה:

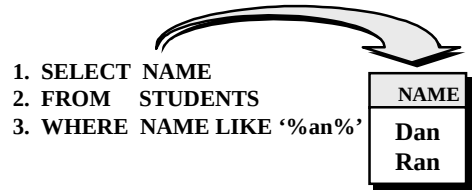
- 'A%' LIKE – מחרוזת המתחילה בתו A ואחריו רצף כלשהו של תוים. לדוגמה: AB או ABC או ABBCC וכד'.
- 'A%' LIKE – מחרוזת המכילה רצף כלשהו של תווים ומסתיימת בתו A. לדוגמה: BA או CBA או CCBBAA וכד'.
- '_A' LIKE – מחרזות בעלת שני תווים כאשר התו הראשון חייב להיות A והבא אחריו תו כלשהו. לדוגמה: AB או AC או AX.
- '_ A_' LIKE – מחרוזת בעלת שלושה תווים כאשר התו הראשון הוא תו כלשהו, התו השני A והתו השלישי תו כלשהו. לדוגמה: XAY או FAA.
- 'A_B_%' LIKE – מחרוזת בה התו הראשון הוא A, התו השני תו כלשהו, התו השלישי הוא B, התו הרביעי תו כלשהו ואחר כך רצף כלשהו של תווים. לדוגמה: AXBY או AZBFGH.
- '%AB%' LIKE – מחרוזת באורך כלשהו המכילה את הרצף AB במקום כלשהו. לדוגמה: AB או AAB או XXYABZZ.

דוגמה א': הצג את שמות כל הסטודנטים ששמם מתחיל באות D.



נשים לב ששימוש בתווי הכללה לאחר תו כלשהו מאפשר לבדוק אם המחרוזת מתחילה בתו המבוקש ולאחריו מספר כלשהו של תווים כלשהם.

דוגמה ב': הצג את שמות כל הסטודנטים שבשמם מופיע המחרוזת an.



נשים לב שבשורה 3 עטפנו את המחרוזת משני צדדיה תו הכללה, ולכן תישלפנה כל השורות בהן השם מכיל מחרוזת זו במקום כלשהו בתוך השם.

דוגמה ג' : הצג את שמות הסטודנטים שהאות v מופיעה במקום השלישי בתוך שמם.

```
1. SELECT  NAME
2. FROM    STUDENTS
3. WHERE  NAME LIKE '_ _v%'
```

NAME
David
Tova

נשים לב שבשורה 3 מופיע תו ההכללה המיוחד _ פעמיים ברציפות, נקבל את כל המחרוזות שמכילות תו כלשהו בשני המקומות הראשונים, את התו v לאחר מכן ומספר כלשהו של תווים לאחר מכן.

דוגמה ד' : הצג את קוד המחלקה ושם המחלקה עבור כל המחלקות שבהן ראש המחלקה אינו בעל תואר דוקטור.

```
1. SELECT  DEPART, NAME
2. FROM    DEPARTMENTS
3. WHERE  HEAD  NOT LIKE 'Dr. %'
```

DEPART	NAME
MT	Mathematics
CH	Chemistry

דוגמה ה' : בעיה מיוחדת מתעוררת אם המחרוזת הנבדקת עלולה להכיל בעצמה את אחד משני תווי ההכללה, כלומר את סימן האחוז או קו תחתי. במקרה זה, אם נשתמש בתו % בתוך המחרוזת המבוקשת הוא יפוענח כתו הכללה, ולא כתו חוקי בתוך המחרוזת. כדי להתמודד עם בעיה זו, ניתן להגדיר תו מיוחד המסמן שאחריו יבוא תו הכללה. תו מיוחד זה נקרא בשם **תו חילוף** (Escape Character). דוגמה זו באה להדגים את השימוש בתו מיוחד זה.

נניח שבתוך שם המחלקה בטבלת המחלקות יכול להופיע גם קו תחתי. הטבלה הבאה מדגימה מצב זה:

DEPART	NAME	HEAD
קוד	שם מחלקה	ראש מחלקה
CS	Computer_Science	Dr. Israel
MT	Mathematics	Prof. Levy
BS	Business_Administration	Dr. Eyal
CH	Chemistry	Prof. Doron

נניח שאנו מבקשים להציג את כל המחלקות בהן מופיעה בשם המחלקב המחרוזת _er או _ess. אם נכתוב את פקודת SQL בצורה רגילה, נקבל:

```
1. SELECT  *
2. FROM    DEPARTMENTS
3. WHERE   NAME LIKE  '%er_%' OR
4.         NAME  LIKE '%ess_%'
```

מעבד SQL יפענח את התו _ (מקף תחתי) המופיע במחרוזת המבוקשת כתו הכללה, ולא כתו חוקי שיכול להופיע בשם המחלקה. הפתרון הוא להגדיר את תו החילוף המיוחד המסמן למעבד שאין לפרש את התו המופיע אחריו במחרוזת כתו הכללה, אלא כתו רגיל. הצורה הנכונה לכתוב את הפקודה היא:

```
1. SELECT  *
2. FROM    DEPARTMENTS
3. WHERE   NAME  LIKE '%er&_%'
4.         OR  NAME  LIKE '%ess&_%'
5.         ESCAPE &
```

DEPART	NAME	HEAD
CS	Computer_Science	Dr. Israel
BS	Business_Administration	Dr. Eyal

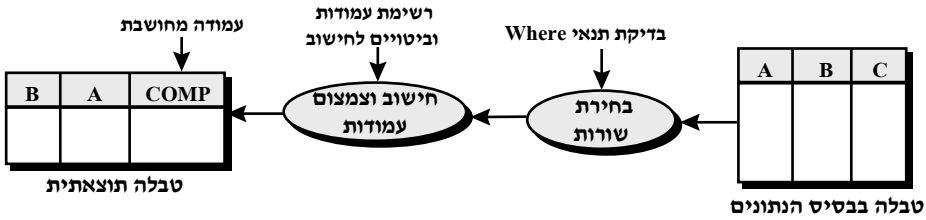
בדוגמה זו, תו החילוף הוא & כפי שמוגדר בשורה 5, ולכן המהדר "מבין" שאחרי סימן מיוחד זה לא בא תו הכללה אלא תו רגיל. כמובן שיש לוודא שתו החילוף לא יהיה תו חוקי בתוך המחרוזת.

עמודות מחושבות (Calculated Columns)

שפת SQL מאפשרת ביצוע חישובים על עמודות המכילות ערכים נומריים. האופרטורים המותרים לשימוש הם חיבור, חיסור, כפל וחילוק. ניתן להשתמש בסוגריים לקביעת סדר הביצוע. העמודה המתקבלת היא עמודה מחושבת – תוצאה של חישוב כלשהו בין מספר עמודות. מקובל לקרוא לביטוי שבו משתתפת עמודה בתהליך החישוב כ**ביטוי עמודות** (Column Expression). פיענוח ביטוי העמודה מתבצע משמאל לימין עם קדימות לסוגריים.

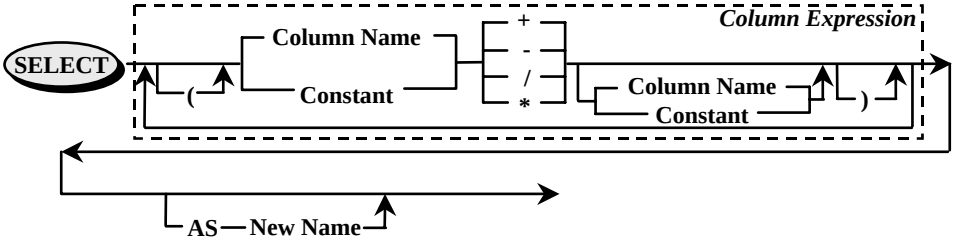
עיקרון הפעולה:

השאילתה שולפת את השורות של הטבלה, בוחרת את השורות המקיימות את התנאי המופיע במשפט WHERE, מחשבת את העמודות על פי הביטויים המופיעים במשפט SELECT, בוחרת את העמודות המבוקשות, מסדרת אותן על פי הסדר המבוקש ובונה את טבלת התוצאה.



תרשים 8.17: עיקרון הפעולה של שאילתה עם עמודות מחושבות.

תחביר הפקודה:



תרשים 8.18: תרשים תחביר לעמודות מחושבות.

דוגמה א' : הצג את מספר הסטודנט והציון הסופי של כל מי שלמד בקורס C-200 בסתיו '99, אילו היינו מוסיפים חמש נקודות לכל נבחן.

```
1. SELECT STUDENT_ID, GRADE + 5
2. FROM   GRADES
3. WHERE COURSE_ID = 'C-200' AND
4.            SEMESTER = 'AUT1999'
```

STUDENT_ID	COL2
105	95
210	90
200	83

לעמודה השנייה אין שם, מאחר והיא עמודה מחושבת. לכן נקבע לה שם שרירותי – COL2, כלומר העמודה השנייה.

דוגמה ב' : הצג את רשימת הקורסים בהם סטודנט קיבל ציון סופי הגבוה ב-20 נקודות מהציון במבחן הסמסטריאלי. יש להציג את ההפרש בין הציון הסופי לבין הציון הסמסטר עבור סטודנטים אלה. לעמודה החדשה נקרא DELTA.

```
1. SELECT   STUDENT_ID, COURSE_ID,
2.                GRADE - GRADE_SEM AS DELTA
3. FROM     GRADES
4. WHERE   GRADE - GRADE_SEM > 20
```

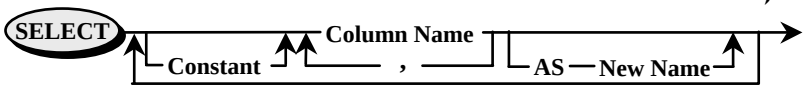
STUDENT_ID	COURSE_ID	DELTA
105	M-100	25
210	B-10	28
200	C-200	28

נשים לב שבשורה 2 מופיעה גם האפשרות לתת שם לעמודה המחושבת.

הוספת כיתוב קבוע בשורות המוצגות

קיימת אפשרות להוסיף כיתוב קבוע בכל אחת מהשורות המוצגות. ניתן להשתמש באפשרות זו לשיפור בהירות המידע המוצג.

תחביר הפקודה:



תרשים 8.19: תרשים תחביר להוספת כיתוב קבוע בכל שורה

דוגמה : הצג את מספר הסטודנט והציון הסופי של כל מי שלמד בקורס C-200 בסתיו *99'*, אילו היינו מוסיפים חמש נקודות לכל נבחן. יש להציג את הכיתוב "Updated Grade =" בעמודה שלפני הציון המחושב. לעמודה המחושבת יש לקבוע את הכותרת GRD.

```
1.  SELECT STUDENT_ID, 'UPDATED GRADE = ',
2.             GRADE + 5 AS GRD
3.  FROM       GRADES
4.  WHERE      COURSE_ID = 'C-200' AND
5.             SEMESTER = 'AUT1999'
```

STUDENT_ID	UPDATED GRADE =	GRD
105	UPDATED GRADE =	95
210	UPDATED GRADE =	90
200	UPDATED GRADE =	83

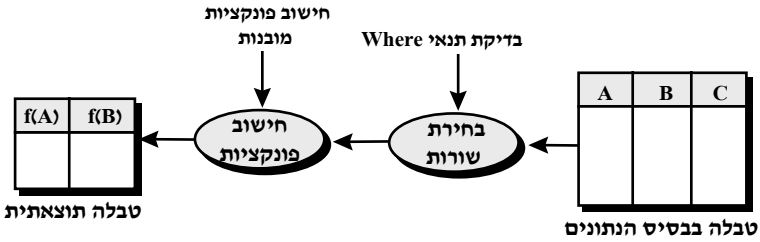
פונקציות מובנות (Built-in Functions)

פונקציות הסיכום פועלות על עמודה כלשהי ומחזירות ערך יחיד. שפת SQL מכילה מספר פונקציות פנימיות:

- COUNT – ספירת המופעים של ערכים בתוך עמודה.
- AVG – חישוב ממוצע הערכים של עמודה.
- SUM – חישוב סכום הערכים של עמודה.
- MAX – מציאת הערך הגבוה ביותר מבין כל הערכים בעמודה.
- MIN – מציאת הערך הנמוך ביותר מבין כל הערכים בעמודה.

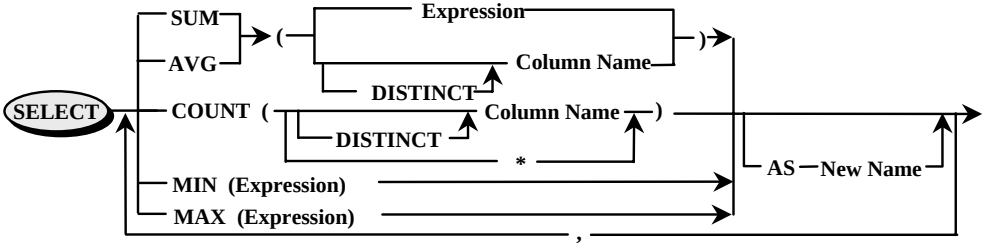
עיקרון הפעולה:

השאילתה בוחרת את השורות המקיימות את התנאי המופיע במשפט WHERE, בונה טבלת ביניים המכילה את השורות שנשלפו, מבצעת את החישובים הנדרשים על ידי הפונקציות הפנימיות המופיעות במשפט SELECT, מסדרת את העמודות לפי הסדר המבוקש ובונה את טבלת התוצאה.



תרשים 8.20: עיקרון הפעולה של שאילתה עם פונקציות מובנות.

תחביר הפקודה:



תרשים 8.21: תרשים תחביר לפונקציות מובנות.

מותר להשתמש בפונקציות מובנות רק במשפטי SELECT ובמשפטי HAVING, כפי שיוסבר בהמשך. הפונקציות SUM ו-AVG פועלות על עמודות נומריות בלבד בעוד שהפונקציות MIN, MAX ו-COUNT פועלות על כל סוגי העמודות. אם רוצים שהפונקציה תתעלם מערכים כפולים בעמודה, ניתן להשתמש ב-DISTINCT לפני שם העמודה. הביטוי DISTINCT יכול להופיע בפונקציה מובנית אחת בלבד בפקודה SELECT. משפט SELECT המכיל פונקציות מובנות אינו יכול להכיל גם עמודות.

דוגמה א' : הצג את הציון הסופי הנמוך ביותר והגבוה ביותר שהושגו בקורס M-100.

```
1. SELECT MIN (GRADE) , MAX (GRADE)
2. FROM    GRADES
3. WHERE COURSE_ID = 'M-100'
```

COL1	COL2
65	90

ניתן לתת שמות לעמודות המתקבלות על ידי שימוש ב-AS.

```
1. SELECT MIN (GRADE) AS MIN_GRADE,
2.            MAX (GRADE) AS MAX_GRADE
3. FROM    GRADES
4. WHERE  COURSE_ID = 'M-100'
```

MAX_GRADE	MIN_GRADE
65	90

דוגמה ב' : הצג את הציון הממוצע שקיבלו כל הסטודנטים בקורס M-100 ואת מספר הסטודנטים שלמדו בקורס.

```
1. SELECT COUNT (STUDENT_ID) ,
2.            AVG (GRADE)
3. FROM    GRADES
4. WHERE  COURSE_ID = 'M-100'
```

COL1	COL2
5	82

דוגמה ג' : נניח שאנו מגדירים מקדם שהוא היחס בין ציון הסמסטר לבין הציון הסופי. יש להציג את היחס הטוב ביותר שהושג על ידי סטודנט כלשהו. את היחס יש להציג באחוזים. יש לתת לעמודה זו את השם Best_Ratio.

```
1. SELECT MAX (100 * GRADE_SEM / GRADE)
2.            AS BEST_RATIO
3. FROM    GRADES
4. WHERE  COURSE_ID = 'M-100'
```

BEST_RATIO
150

נשים לב שבשורה ראשונה מופיעה פעולה אריתמטית על עמודות והפונקציה MAX למציאת הערך המחושב הגבוה ביותר.

דוגמה ד' : מנה את מספר הסטודנטים שנבחנו בקורס M-100.

```
1. SELECT COUNT (*) AS STUDENT_COUNT
2. FROM    GRADES
3. WHERE COURSE_ID = 'M-100'
```

STUDENT_COUNT
5

אין משמעות לשם העמודה בפונקציה COUNT, מאחר והיא מונה את מספר המופעים. לכן, נקבל אותה תוצאה אם נמנה את הערכים בעמודה STUDENT_ID, בעמודה COURSE_ID, או בכל עמודה אחרת. לכן ניתן לרשום כוכבית (*) במקום לרשום שם מפורש של עמודה.

דוגמה ה' : מנה את מספר הקורסים השונים שבהם נבחנו סטודנטים בסמסטר קיץ 1998.

```
1. SELECT COUNT (DISTINCT COURSE_ID)
2. FROM    GRADES
3. WHERE SEMESTER = 'SUM1998'
```

COL1
2

נשים לב שבשורה ראשונה מופיע DISTINCT כדי שלא למנות פעמיים קורס שנבחנו בו סטודנטים שונים. אם לא היינו מונעים ספירה כפולה, התוצאה היתה 4: שלושה סטודנטים שנבחנו בקורס M-100 ועוד סטודנט שנבחן בקורס C-55.

אריתמטיקה של תאריכים (Date Arithmetics)

שפת SQL תומכת בטיפוסי נתונים שונים וביניהם גם בטיפוס נתונים מסוג תאריך. היתרון של תמיכה ישירה בטיפוס נתונים נפוץ זה, מתבטא הן בבדיקות חוקי וּת תאריך המבוצעות באופן אוטומטי על עמודה מסוג זה, בעת הכנסת שורות חדשות, והן בתמיכה באריתמטיקה של תאריכים. אריתמטיקה זו מאפשרת לבצע פעולות כגון חיסור תאריכים, הוספת מספר ימים לתאריך וקבלת תאריך חדש.

להדגמת התמיכה בתאריכים נשתמש בטבלת המחלקות ונוסיף לה עמודה חדשה: תאריך מינוי של ראש המחלקה. הפעולה מוצגת בתרשים 8.22.

DEPART	NAME	HEAD	APPOINT_DATE
קוד	מחלקה	ראש מחלקה	תאריך מינוי
CS	Computer Science	Dr. Israel	15-JUN-1998
MT	Mathematics	Prof. Levy	01-JAN-1996
BS	Business	Dr. Eyal	20-AUG-1997
CH	Chemistry	Prof. Doron	25-JUL-1996

תרשים 8.22: דוגמה לטבלת מחלקות עם עמודה בעלת טיפוס נתונים תאריך.

שפת SQL מספקת מספר משתנים גלובליים המקבלים את הערכים שלהם באופן ישיר ממערכת RDBMS ואחד מהם הוא המשתנה CURRENT DATE המכיל את התאריך של היום. ניתן להשתמש במשתנה זה בתוך השפה כדי להתייחס לתאריך הנוכחי.

דוגמה א': הצג את כל שמות המחלקה בהן מכהן ראש מחלקה שמונה לתפקידו לפני שנה. נניח שהתאריך הנוכחי הוא 01/12/1998.

```
1. SELECT NAME, HEAD,
2.        APPOINT_DATE
3. FROM   DEPARTMENTS
4. WHERE  APPOINT_DATE >
5.        CURRENT DATE - 365
```

NAME	HEAD	APPOINT_DATE
Computer Science	Dr. Israel	15-JUN-1998

בשורה 5 השתמשנו במשתנה הגלובלי CURRENT DATE במשפט WHERE.

דוגמה ב': הוסף לכל תאריך מינוי 10 ימים והצג את תאריכי המינוי שמתקבלים. הכותרת של העמודה החדשה תהיה New Date (ראה שרטוט בעמוד הבא).

נשים לב שעבור המחלקה לכימיה הוספת 10 ימים לתאריך המינוי נותנת תוצאה נכונה, מכיון שבחודש יולי יש 31 יום, ולכן הוספת 10 ימים לתאריך 25/07/1996 נותנת את התוצאה 04/08/1996.

```
1. SELECT NAME, HEAD,
2.        APPOINT_DATE + 10
3.        AS NEW_DATE
4. FROM   DEPARTMENTS
```

NAME	HEAD	NEW_DATE
Computer Science	Dr. Israel	25-JUN-1998
Mathematics	Prof. Levy	11-JAN-1996
Business	Dr. Eyal	30-AUG-1997
Chemistry	Prof. Doron	04-AUG-1996

מיון התוצאה (ORDER BY)

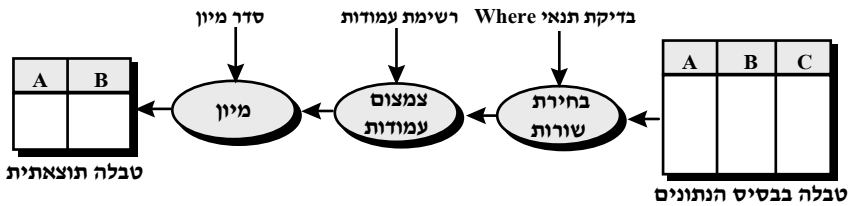
עד כה לא התייחסנו לסדר הופעת השורות בטבלה התוצאתית. הפקודה SELECT מאפשרת להגדיר את הסדר על ידי שימוש במשפט ORDER BY. משפט זה חייב להיות האחרון בפקודה SELECT.

יש לשים לב להבדל המהותי בין שיטה זו לבין המקובל במודלים האחרים של נתונים בהם סדר הנתונים נקבע על ידי מנהל בסיס הנתונים בעת הגדרת בסיס הנתונים. דרישה לקבלת הרשומות בסדר שונה מחייבת את תוכנית היישום לבצע מיון של הרשומות הנשלפות. לעומת זאת, המודל הטבלאי מגדיר את סדר המיון כחלק מהבקשה לשליפת הנתונים ובצורה בלתי תלויה בסדר שבו הנתונים מאוחסנים בבסיס הנתונים.

כאשר השורות בבסיס הנתונים אינן ממוינות בסדר המבוקש, משך השליפה יהיה ארוך יותר. אם קיים אינדקס על העמודות המופיעות במשפט ORDER BY, התשובה תתקבל במהירות רבה, מאחר ורכיב האופטימיזציה יבחר לגשת לנתונים דרך האינדקסים. זוהי דוגמה נוספת ליישום רעיון אי התלות בנתונים. קיום או אי קיום אינדקס על טבלה משפיע רק על זמן התגובה ולא על נכונות התוצאה, כפי שהיה מקובל במודלים האחרים. במודלים האחרים היישום מכיר את סדר קבלת הרשומות ומבסס על סדר זה את הלוגיקה הפנימית ולכן שינוי בסדר קבלת הרשומות – למשל, כתוצאה משינוי שביצע מנהל בסיס הנתונים – יכול לשבש את פעולת התוכנית.

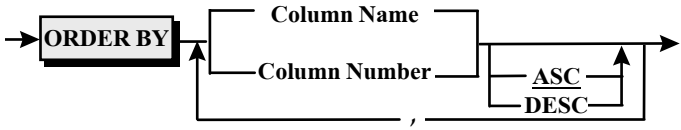
עיקרון הפעולה :

השאילתה בוחרת את השורות המקיימות את התנאים המופיעים במשפט WHERE, מחשבת ומצמצמת את העמודות כפי שמופיע במשפט SELECT, ממיינת את השורות על פי סדר המיון המוגדר במשפט ORDER BY ובונה את הטבלה התוצאתית.



תרשים 8.23: עיקרון הפעולה של שאילתה עם מיון.

תחביר הפקודה:



תרשים 8.24: תרשים תחביר להגדרת סדר המיון.

ניתן להגדיר עמודה למיון על ידי ציון שמה או מספרה הסידורי, אם העמודה היא עמודה מחושבת. מומלץ להמנע מלהשתמש במספרי עמודות ולהשתמש בשמות עמודות. אם העמודה היא עמודה מחושבת מומלץ לתת לה שם על ידי AS ולהשתמש בשם זה בסדר המיון. לא ניתן למיין את התוצאה על סמך עמודה שלא מופיעה במשפט SELECT. היכולת לציין את מספר העמודה חשוב במיוחד אם העמודה היא מחושבת ואין לה שם. ליד שם העמודה או מספרה ניתן לציין אם המיון צריך להיות בסדר עולה, ASC, שהוא ברירת המחדל או בסדר יורד, DESC. העמודה הראשונה המופיעה במשפט GROUP BY נקראת **מפתח מיון ראשי**, העמודה השנייה תקרא **מפתח מיון משני** וכך הלאה. כאשר שתי שורות במיון הראשי מכילות את אותו ערך, המיון שלהן יהיה על פי המפתח המשני ואם במפתח המשני יש שני ערכים זהים המיון של השורות יהיה על פי המפתח השלישי וכך הלאה.

דוגמה א' : הצג את מספר הסטודנט והציון הסופי ממוין מהגבוה לנמוך, לכל מי שלמדו בקורס M-100.

```
1.  SELECT      STUDENT_ID, GRADE
2.  FROM        GRADES
3.  WHERE       COURSE_ID = 'M-100'
4.  ORDER BY    GRADE DESC, STUDENT_ID
```

STUDENT_ID	GRADE
200	90
210	90
245	90
105	75
310	65

בדוגמה זו לא ציינו את סדר המיון עבור מפתח המיון המשני ולכן הוא ימוין בסדר עולה, מכיון שזו ברירת המחדל.

דוגמה ב' : הצג רשימה ממוינת של שמות הקורסים המזכים בשלוש נקודות.

```
1.  SELECT      COURSE_NAME, TYPE
2.  FROM        COURSES
3.  WHERE       POINTS = 3
4.  ORDER BY    COURSE_NAME
```

COURSE_NAME	TYPE
Database	CLASS
Linear Algebra	CLASS
Numeric Analysis	CLASS
Operations Research	SEMIN

דוגמה ג' : הצג את מספר הסטודנט ואת ההפרש בין הציון הסופי לציון הסמסטריאלי. את התוצאה יש להציג במיון לפי ההפרש בסדר יורד.

```
1. SELECT     STUDENT_ID,
2.            GRADE - GRADE_SEM
3. FROM       GRADES
4. WHERE      COURSE_ID = 'M-100'
5. ORDER BY 2 DESC
```

STUDENT_ID	COL2
245	10
210	0
200	0
105	-15
310	-35

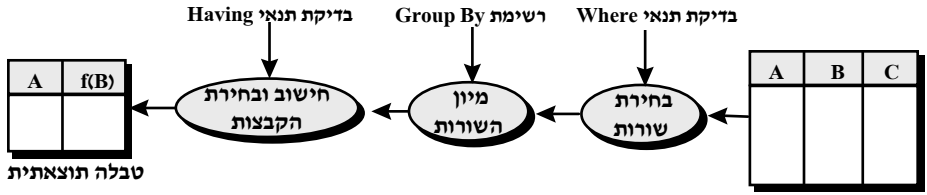
כפי שניתן לראות בשורה 5, השתמשנו כאן במספר העמודה ולא בשמה, מאחר והעמודה השנייה היא עמודה מחושבת ולכן אין לה שם. כזכור, שיטה זו אינה מומלצת ועדיף לתת שם לעמודה מחושבת ולהשתמש בשם זה במשפט ORDER BY.

שאילתות מקובצות (Grouped Queries)

משפט GROUP BY מאפשר הקבצה של שורות בעלות ערך זהה, כלומר להציג שורה אחת עבור אוסף של שורות המכילות את אותו הערך בעמודות מסוימות. יחד עם משפט GROUP BY ניתן להשתמש במשפט HAVING כדי להגדיר תנאי על השורות המקובצות. נשים לב, שלהבדיל מהתנאי המוגדר במשפט WHERE הפועל על שורות רגילות, התנאי המוגדר במשפט HAVING פועל רק על השורות המקובצות.

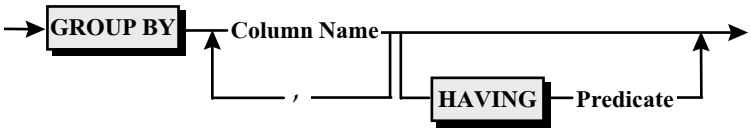
עיקרון הפעולה:

השאילתה בוחרת את השורות המקיימות את התנאים שבמשפט WHERE, ממיינת את השורות שהתקבלו לפי סדר ההקבצה שבמשפט GROUP BY, מחשבת את הפונקציות המובנות שבמשפט SELECT, בוחרת רק את השורות המקיימות את התנאים שבמשפט HAVING, מסדרת את העמודות על פי הסדר המבוקש ובונה את טבלת התוצאה.



תרשים 8.25: עיקרון הפעולה של שאילתה עם הקבצות.

תחביר הפקודה :



תרשים 8.26: תרשים תחביר GROUP BY.

השימוש ב- GROUP BY מטיל מספר מגבלות על משפט SELECT :

- ❖ חייבת להיות התאמה בין העמודות המופיעות במשפט SELECT לבין העמודות המופיעות במשפט GROUP BY. כלומר, כל שם עמודה המופיע במשפט GROUP BY חייב להופיע קודם במשפט SELECT ובאותו הסדר, אבל לא להיפך.
- ❖ משפט SELECT יכול להכיל רק שמות של עמודות, פונקציות מובנות, קבועים או ביטוי המתייחס לאחד מהמרכיבים האלה.

דוגמה א' : הצג את מספר הסטודנטים שנבחנו בכל קורס. מיין את התוצאה לפי מספר הסטודנטים בסדר יורד, כלומר מהקורס שלמדו בו הכי הרבה סטודנטים ועד לקורס שלמדו בו הכי מעט סטודנטים.

```
1. SELECT      COURSE_ID, COUNT(STUDENT_ID)
2.             AS STUDENT_COUNT
3. FROM        GRADES
4. GROUP BY    COURSE_ID
5. ORDER BY    STUDENT_COUNT
```

COURSE_ID	STUDENT_COUNT
M-100	5
B-10	3
C-200	3
B-40	2
C-55	1

דוגמה ב' : כמו בדוגמה קודמת, אולם יש להציג רק קורסים בהם נבחנו 3 סטודנטים ומעלה.

```
1. SELECT      COURSE_ID, COUNT(STUDENT_ID)
2.             AS STUDENT_COUNT
3. FROM        GRADES
4. GROUP BY    COURSE_ID
5. HAVING      COUNT (STUDENT_ID) >=3
6. ORDER BY    STUDENT_COUNT
```

COURSE_ID	STUDENT_COUNT
M-100	5
B-10	3
C-200	3

דוגמה ג' : כמו בדוגמה הקודמת אולם רק עבור סטודנטים שקיבלו ציון 80 ומעלה (ראה שרטוט בעמוד הבא).

זוהי דוגמה לשימוש משולב במשפט WHERE הפועל על השורות הבודדות ומשפט HAVING הפועל על השורות המקובצות.

```
1. SELECT      COURSE_ID, COUNT(STUDENT_ID)
2.             AS STUDENT_COUNT
3. FROM        GRADES
4. WHERE       GRADE >= 80
5. GROUP BY COURSE_ID
6. HAVING      COUNT (STUDENT_ID) >= 3
7. ORDER BY  STUDENT_COUNT
```

COURSE_ID	STUDENT_COUNT
M-100	3

תקן SQL3, הנמצא עדיין במצב טיוטה, מכיל הרחבה בנושא הקבצת שורות והצגת סיכומי ביניים (Sub Totals). הרחבה זו נועדה להקל על שאילתות העוסקות בניתוח רב-מימדי, שהפך לפופולרי מאוד עם הופעת טכנולוגיית מחסני הנתונים ומערכות OLAP (On Line Analytical Processing) המבוססות על היכולת להציג נתונים מסוכמים בחתכים (מימדים) שונים. קורא המעוניין להרחיב ידיעותיו בנושא OLAP מופנה לספרו של המחבר **מחסני נתונים – עקרונות ארכיטקטורה, עיצוב ויישום** שבהוצאת **הוד-עמי** [HR99]. כדי לאפשר הצגת סיכומי ביניים לשורות המופיעות בטבלת התוצאה, הוגדרה הרחבה למשפט GROUP BY – היכולת להגדיר איזה סיכומי ביניים יש להציג. נדגים כדי להבהיר זאת.

דוגמה ה': נניח שברצוננו להציג את מספר הסטודנטים שנרשמו לקורסים השונים בסמסטרים השונים. השאילתה העונה לדרישה זו:

```
1. SELECT      COURSE_ID, SEMESTER, COUNT(*)
2.             AS STUDENT_COUNT
3. FROM        GRADES
4. GROUP BY  COURSE_ID, SEMESTER
5. ORDER BY  COURSE_ID, SEMESTER
```

COURSE_ID	SEMESTER	STUDENT_COUNT
B-10	AUT1999	2
B-10	WIN1999	1
B-40	WIN1998	1
B-40	WIN1999	1
C-55	SUM1998	1
C-200	AUT1999	3
M-100	AUT1999	2
M-100	SUM1998	3

אם נרצה לקבל סיכום ביניים לכל קורס וסה"כ כללי, נצטרך לבצע זאת בשאילתות נפרדות. כדי לאפשר קבלת סיכומי ביניים אלה בתוך אותה שאילתה, הורחב משפט GROUP BY על ידי תוספת הפרמטר GROUPING SETS המגדיר את סיכומי הביניים.

```
1. SELECT      COURSE_ID, SEMESTER, COUNT(*)
2.              AS STUDENT_COUNT
3. FROM        GRADES
4. GROUP BY    GROUPING SETS
5.             ((COURSE_ID, SEMESTER), (COURSE_ID), ( ))
6. ORDER BY    COURSE_ID, SEMESTER
```

COURSE_ID	SEMESTER	STUDENT_COUNT
B-10	AUT1999	2
B-10	WIN1999	1
B-10		3
B-40	WIN1998	1
B-40	WIN1999	1
B-40		2
C-55	SUM1998	1
C-55		1
C-200	AUT1999	3
C-200		3
M-100	AUT1999	2
M-100	SUM1998	3
M-100		5
Total		14

סיכומי ביניים

סה"כ כללי

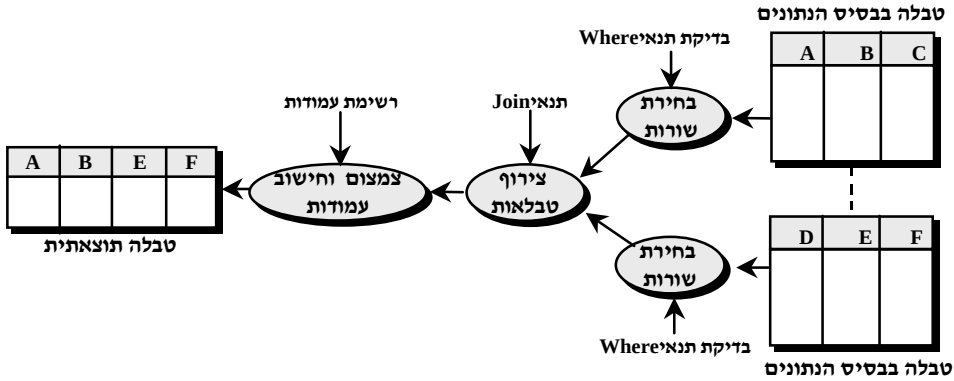
כפי שניתן לראות, שורה 4 של פקודת SQL קובעת שיש לבנות סיכומי ביניים ושורה 5 קובעת מהם ההקבצות וסיכומי הביניים שיש להציג. את ההקבצה יש להציג עבור מספר הקורס והסמסטר ובנוסף יש להציג את סיכום הביניים עבור מספר הקורס ואת הסה"כ הכללי, המופיע בתור שני סוגריים.

שאילתות עם מספר טבלאות (Multi-table Queries)

שפת SQL מסוגלת כמובן לטפל בצורה נוחה גם בשאילתות המתייחסות למספר טבלאות בו-זמנית, תוך ניצול הקשרים הלוגיים הקיימים בין הטבלאות בבסיס הנתונים. ניתן לצרף (Join) בקלות טבלאות שונות על סמך שוויון או יחס כלשהו בין הערכים של עמודות השייכות לטבלאות שונות. הצירוף בין הטבלאות מתבצע על ידי הגדרת תנאי כלשהו בין העמודות.

עיקרון הפעולה:

השאילתה בוחרת את השורות מכל הטבלאות המופיעות במשפט FROM על פי התנאי המופיע במשפט WHERE והמתייחס לכל אחת מהטבלאות, מבצעת את צירוף כל טבלאות הביניים על פי תנאי הצירוף המופיע במשפט WHERE ובונה טבלת ביניים חדשה, בוחרת ומחשבת את העמודות מתוך טבלת הביניים החדשה ובונה את הטבלה התוצאתית. אם מופיע גם משפט ORDER BY, טבלת הביניים תמוין על פי הנדרש.



תרשים 8.27: עיקרון הפעולה לשאילתה עם צירוף בין שתי טבלאות.

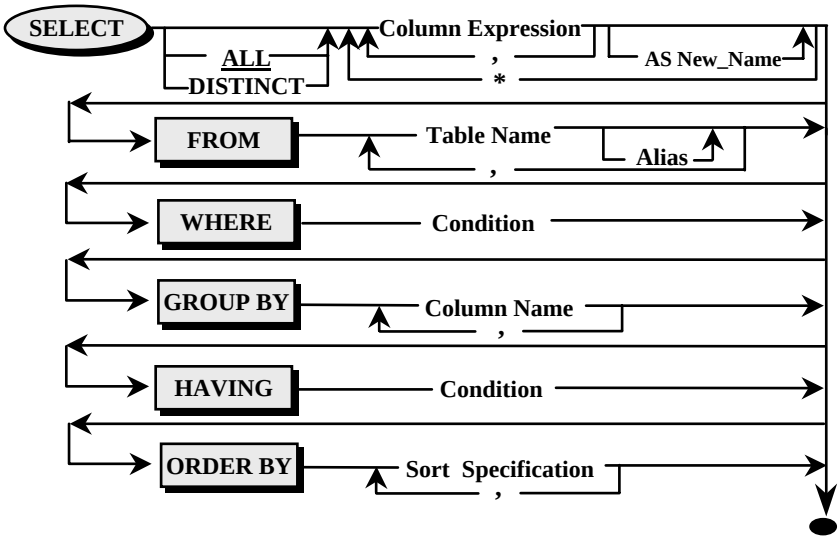
ככל שמספר הטבלאות המשתתפות בשאילתה גדל, ברור שהזמן הנדרש לבצע את השאילתה מתארך גם כן. למרות שתקן ANSI אינו קובע מגבלה על מספר הטבלאות המשתתפות בשאילתה אחת, מערכות מסחריות קובעות מגבלות. בעיקרון מקובל להניח שהמגבלה המעשית למספר הטבלאות המשתתפות בשאילתה אחת לא יעלה על שבע או שמונה טבלאות. במערכות OLTP מגבלה זו סבירה, כי רוב התנועות אינן פונות למספר רב של טבלאות. מגבלה זו יכולה להיות מורגשת במערכות מחסני נתונים, בהן סביר למצוא גם מספר גדול יותר של טבלאות המשתתפות בשאילתה אחת.

תחביר הפקודה:

שאילתה המתייחסת למספר טבלאות דומה מאוד מבחינה תחבירית לשאילתה רגילה. ההבדלים העיקריים הם:

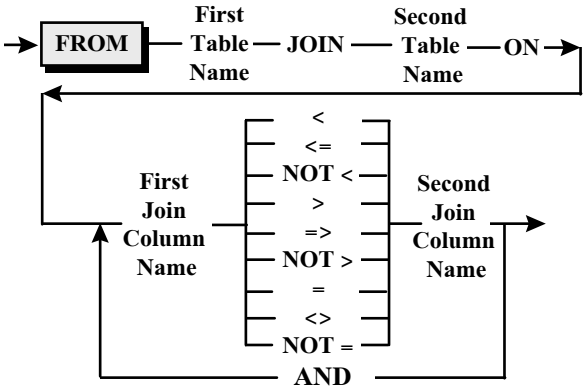
- **ריבוי טבלאות:** במשפט FROM מופיעות מספר טבלאות ולא רק טבלה אחת.
- **זיהוי חד-משמעי של עמודות (Column Qualification):** שם של עמודה המופיעה בפקודה חייב להיות חד-משמעי ולכן צריך להשתמש בשיטה מיוחדת לזיהוי מקור העמודה – לאיזו טבלה היא שייכת. הדרך ליצירת זיהוי חד-משמעי היא על ידי שימוש **במציין עמודה** (Column Qualifier) המקדים את שם העמודה. למשל, GRADES.GRADE_ID מציין שהעמודה GRADE_ID שייכת לטבלת GRADES. מאחר ושמות הטבלאות יכולים להיות ארוכים, שיטת זיהוי זו יכולה להכביד על כתיבת השאילתה ולכן מאפשרת שפת SQL קביעת שם נרדף לטבלה (Table Alias) ושימוש בשם הנרדף של הטבלה במקום השם המלא. לדוגמה, ניתן לקבוע לטבלת GRADES שם נרדף G ואז הזיהוי החד-משמעי יהיה G.GRADE_ID.
- **תנאי צירוף (Join Condition):** במשפט WHERE יכולים להופיע בנוסף לתנאים הרגילים שמטרתם לבחור אוסף מסוים של שורות, גם מספר תנאים מיוחדים שמקובל לקרוא להם **תנאי הצירוף**. כתיבת תנאים אלה דומה לכתיבת תנאי השוואה כלשהם, אולם מייחדת אותם העובדה שמשני צידי התנאי יש התייחסות לעמודות השייכות לטבלאות שונות. התנאי יכול לדרוש שוויון בין ערכי עמודות (Equi Join) או

תנאי לוגי כלשהו בין ערכי עמודות (Theta Join), כמו למשל שהערכים בעמודה A יהיו גדולים מהערכים בעמודה B.



תרשים 8.28: תרשים תחביר לפקודה SELECT הפועלת על מספר טבלאות.

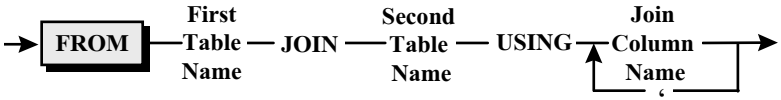
תקן SQL2 מאפשר חלופה נוספת לתיאור הגדרת הצירוף. במקום הגדרת תנאי הצירוף במשפט WHERE, ניתן להגדיר את הצירוף במשפט FROM בזמן הגדרת הטבלאות.



תרשים 8.29: תרשים תחביר לצירוף טבלאות במשפט FROM.

תקן SQL2 מספק עוד צורת רישום מקוצרות עבור **צירוף שוויון** (Equi Join) כאשר שמות עמודות הצירוף הן בעלות שמות זהים בשתי הטבלאות.

חלופה א': בצורת רישום זו משתמשים ב-USING ובציון שמות העמודות המשתתפות בצירוף.



תרשים 8.30: שימוש ב-USING לצירוף טבלאות.

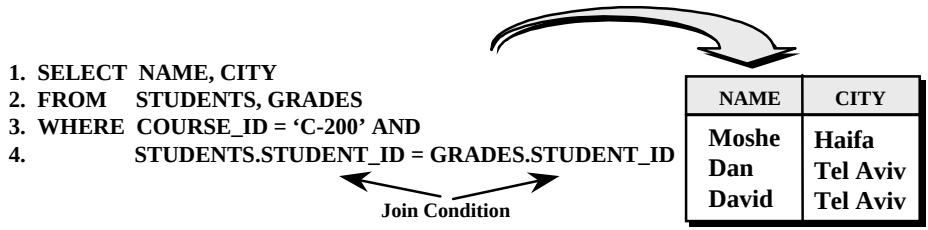
חלופה ב': בחלופה זו משתמשים ב**צירוף הטבעי** (Natural Join), שבו אין צורך לציין במפורש את שמות העמודות המשתתפות בצירוף. במקרה זה, עמודות הצירוף נקבעות באופן לא מפורש, כך שכל העמודות בעלות שמות זהים בשתי הטבלאות, הופכות לעמודות צירוף.

תרשים 8.31: רישום של צירוף טבעי.

בהמשך נציג את הדוגמאות תוך שימוש בתקן הישן יותר וכן בחלופות שונות לכתיבת השאילתה על פי תקן SQL2.

דוגמאות לשאילתות

דוגמה א': הצג את שמות הסטודנטים ואת כתובתם עבור כל הסטודנטים שלמדו בקורס C-200.



נשים לב שלצורך קבלת התוצאה, בוצע צירוף בין טבלת הציונים לבין טבלת הסטודנטים בהתבסס על שוויון בערכים של מספר הסטודנט המופיע בשתי הטבלאות. תנאי צירוף זה מופיע בשורה 4 כחלק ממשפט WHERE. למעשה זהו קשר אב-בן בין טבלת הסטודנטים לבין טבלת הציונים והוא בא לידי ביטוי בעובדה שבטבלת הסטודנטים העמודה של מספר הסטודנט היא **מפתח עיקרי** (Primary Key) ואילו בטבלת הציונים, מספר הסטודנט הוא **מפתח זר** (Foreign Key). מאחר והתנאי לצירוף הוא שוויון, זהו Equi Join. נשים לב לצורה בה יצרנו זיהוי חד משמעי של שמות עמודות על ידי הוספת קידומת של שם הטבלה לפני שם העמודה.

דוגמה ב' : אותה שאילתה אבל תוך שימוש בשמות נרדפים לטבלאות :

```
1. SELECT   NAME, CITY
2. FROM     STUDENTS  S, GRADES G
3. WHERE    COURSE_ID = 'C-200' AND
4.           S .STUDENT_ID = G.STUDENT_ID
```

בדוגמה זו רואים כיצד השימוש בשמות נרדפים לטבלאות, המופיע בשורה 2, הפך את כתיבת השאילתה לפשוטה יותר. תקן SQL2 מאפשר בנוסף לרישום תנאי הצירוף עוד מספר צורות כתיבה חלופיות לצירוף.

דוגמה ג' : אותה שאילתה עם צורת רישום תנאי הצירוף במשפט FROM, לפי תקן SQL2. נציג את שלושת החלופות לצורת הרישום.

חלופה א':

```
1.  SELECT  NAME, CITY
2.  FROM    STUDENTS  S JOIN GRADES G
3.          ON S .STUDENT_ID = G.STUDENT_ID
4.  WHERE   COURSE_ID =   'C-200' AND
```

בשורה 2 מופיע JOIN בין שתי הטבלאות ובשורה 3 מופיע תנאי הצירוף. צורת כתיבה זו מבחינה באופן ברור בין תנאים לשליפת שורות לבין תנאים לצירוף. לדוגמה, בשורה 4 מופיע תנאי רגיל לשליפת שורות, ובשורה 3 מופיע התנאי לצירוף.

חלופה ב':

```
1.  SELECT  NAME, CITY
2.  FROM    STUDENTS  S JOIN GRADES G USING STUDENT_ID
3.  WHERE   COURSE_ID =   'C-200' AND
```

החלופה בשורה 2 מתאימה למקרה ששמות העמודות המשתתפות בצירוף זהים.

חלופה ג':

```
1.  SELECT NAME, CITY
2.  FROM    STUDENTS  NATURAL JOIN GRADES
3.  WHERE   COURSE_ID =   'C-200' AND
```

כפי שניתן לראות בשורה 2, השתמשנו כאן בצירוף טבעי (Natural Join) שבו אין צורך להגדיר את עמודות הצירוף. המערכת מחפשת את העמודות עם שמות זהים בשתי הטבלאות ומשתמשת בהם לביצוע הצירוף. מאחר ובמקרה שלנו, רק העמודות STUDENT_ID הן בעלות שם זהה בשתי הטבלאות, הצירוף יבוצע בצורה נכונה.

דוגמה ד': כמו שאילתה ג', אבל יש להציג את כל העמודות של טבלת הסטודנטים ושל טבלת הציונים.

נשים לב לצורת הרישום של בחירת כל העמודות משתי הטבלאות, כפי שמופיע בשורה ראשונה.

```
1.  SELECT     S.*, G.*
2.  FROM       STUDENTS  S, GRADES G
3.  WHERE      COURSE_ID = 'C-200' AND
4.             S .STUDENT_ID = G.STUDENT_ID
```

STUDENT_ID	NAME	CITY	STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE	GRADE_SEM
105	Moshe	Haifa	105	C-200	AUT1999	A	90	85
210	Dan	Tel Aviv	210	C-200	AUT1999	A	85	80
200	David	Tel Aviv	200	C-200	AUT1999	B	78	50

דוגמה ה' : הצג את ההפרש בין הציון הסופי לבין ציון הסמסטר ואת מספר הסטודנט, לכל הסטודנטים ששמם מתחיל באות R.

```
1.  SELECT   G.GRADE - G.GRADE_SEM AS DIFF,
2.           G.STUDENT_ID
3.  FROM     STUDENTS  S, GRADES G
4.  WHERE    S.NAME LIKE 'R%'  AND
5.           S .STUDENT_ID = G.STUDENT_ID
```

DIFF	STUDENT_ID

ההפרש בין הציונים מחושב בשורה ראשונה. הסטודנט Ran הוא היחידי העונה על הדרישה של שם סטודנט המתחיל באות R. מאחר והוא לא למד בקורס כלשהו, נקבל טבלה תוצאתית ריקה.

דוגמה ו' : הצג רשימה ממוינת של שם הסטודנט וסכום נקודות הזכות שהוא קיבל בכל הקורסים בהם השתתף. לצורך הדוגמה נניח שהנוסחה לחישוב נקודות הזכות היא :

ציון סופי בקורס X מספר נקודות הזכות
100

נקודות הזכות ניתנות רק אם הוא קיבל ציון סופי של 60 ומעלה בקורס.

```
1 .  SELECT      NAME, SUM (GRADE * POINTS /100 )
2 .              AS W_GRADE
3.   FROM        STUDENTS  S, GRADES  G, COURSES  C
4 .  WHERE       GRADE >=   60  AND
5 .              S.STUDENT_ID = G.STUDENT_ID AND
6 .              G.COURSE_ID = C.COURSE_ID
7.   GROUP BY    NAME
8.   ORDER BY    W_GRADE DESC
```

NAME	W_GRADE
Moshe	7.95
David	7.22
Dan	6.88
Yoel	2.55
Tova	1.85

נציג את אותה שאילתה בצורת הכתיבה החלופית של תקן SQL2.

```
1. SELECT      NAME, SUM (GRADE * POINTS / 100) AS W_GRADE
2. FROM        (STUDENTS  S NATURAL JOIN GRADES  G)
3.              AS SG NATURAL JOIN COURSES  C
4. WHERE       GRADE >= 60
5. GROUP BY    NAME
6. ORDER BY    W_GRADE DESC
```

נשים לב שבצירוף הזה משתתפות שלוש טבלאות. בשורה 2 מופיע הצירוף הטבעי בין טבלת הסטודנטים לטבלת הציונים. הצירוף יבוצע עם העמודות STUDENT_ID, מאחר ורק הן בעלות שם זהה בשתי הטבלאות. לטבלת הביניים המתקבלת קראנו בשם SG ואותה אנו מצרפים בצירוף טבעי לטבלת הקורסים, כפי שמופיע בשורה 3. הצירוף הטבעי יבוצע עם העמודות COURSE_ID, מאחר ורק שם זה זהה הן בטבלת הביניים והן בטבלת הקורסים. נדגיש שצורת רישום זו אינה קובעת את סדר ביצוע הצירופים, בדיוק כמו שצורת הרישום של צירוף במשפט WHERE לא קבעה את סדר הביצוע. בכל מקרה, סדר ביצוע הצירופים יקבע על ידי מעבד פקודות SQL בשלב האופטימיזציה.

דוגמה ז': יש להציג את שם הקורס, שם המחלקה, שם הסטודנט לכל הסטודנטים שקיבלו ציון סופי גבוה מ-90 במקצוע כלשהו במועד א'.

```
1.  SELECT  S.NAME, COURSE_NAME, D.NAME
2.  FROM    STUDENTS  S, GRADES  G,
3.          COURSES  C, DEPARTMENTS  D
4.  WHERE   GRADE >= 90 AND TERM =  'A' AND
5.           S.STUDENT_ID = G.STUDENT_ID AND
6.          G.COURSE_ID = C.COURSE_ID
7.          C.DEPARTMENT_ID = D.DEPART
```

NAME	COURSE_NAME	NAME
Yoel	Operations Res.	Business
Tova	Linear Algebra	Mathematics

כפי שניתן לראות בשורות 3 ו-4, השאילתה מתייחסת בו-זמנית לכל ארבע הטבלאות המופיעות בבסיס הנתונים לדוגמה. נציג את אותה שאילתה בצורת הכתיבה החלופית של תקן SQL2:

```
1. SELECT      S.NAME, COURSE_NAME, D.NAME
2. FROM        ((STUDENTS  S NATURAL JOIN GRADES  G) AS SG
3.               NATURAL JOIN COURSES  C) AS SGC
4.               JOIN  DEPARTMENTS  D ON  D.DEPART = SGC.DEPARTMENT_ID
5. WHERE       GRADE >= 90 AND TERM = 'A'
```

נשים לב שבשני הצירופים הראשונים המופיעים בשורה 2 ו-3, השתמשנו בצירוף טבעי, מאחר וניתן להתבסס על שמות זהים של עמודות בשלוש הטבלאות. בצירוף השלישי, המופיע בשורה 4, לא ניתן להשתמש בצירוף טבעי משתי סיבות: (א) לעמודות מספר מחלקה יש שם שונה בטבלת המחלקות ובטבלת הקורסים; (ב) בטבלת המחלקות מופיעה עמודה עם השם NAME וגם בטבלת הסטודנטים מופיעה עמודה עם אותו השם ולכן צירוף

טבעי היה משתמש גם בעמודה זו לביצוע הצירוף. לכן, בצירוף השלישי השתמשנו בפרמטר ON תוך ציון מפורש של שמות עמודות הצירוף. נשים לב שהשתמשנו בשם טבלת הביניים SGC, כדי לזהות חד-ערכית את שם העמודה DEPARTMENT_ID.

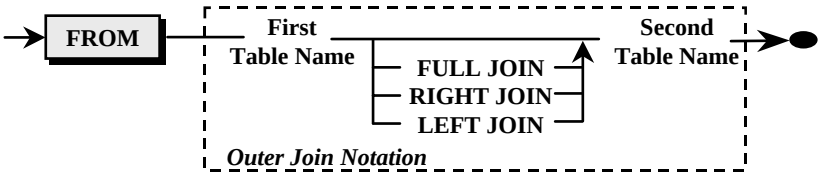
צירוף טבלאות על ידי Outer Join

הצירוף הרגיל, הנקרא לפעמים גם Inner Join, מעביר לטבלה התוצאתית רק זוגות של שורות משתי הטבלאות עבורן מתקיים תנאי הצירוף. הצירוף החיצוני, Outer Join, מאפשר צירוף שתי טבלאות, כך שאל הטבלה התוצאתית תועברנה גם שורות שאינן מקיימות את תנאי הצירוף. מאחר וצירוף זה מעביר אל הטבלה התוצאתית את כל השורות של אחת או שתי הטבלאות המשתתפות בצירוף, מקובל גם לקרוא לצירוף זה כ**צירוף מְשַמֵר מידע** (Information Preserving Join). קיימות שלוש חלופות לביצוע Outer Join:

- **Left Outer Join** – הצירוף מעביר אל הטבלה התוצאתית רק שורות מהטבלה השמאלית שלא ענו על תנאי הצירוף.
- **Right Outer Join** – הצירוף מעביר אל הטבלה התוצאתית רק שורות מהטבלה הימנית שלא ענו על תנאי הצירוף.
- **Full Outer Join** – הצירוף מעביר אל הטבלה התוצאתית שורות משתי הטבלאות שלא ענו על תנאי הצירוף.

תקן SQL1 לא הגדיר את הצירוף הזה ולכן נוצרו מספר שיטות לסימון הצירוף. חלק מהמערכות מסמנות את הצירוף במשפט WHERE על ידי שימוש בכוכבית המופיעה בצד שמאל, ימין או משני צידי סימן השוויון בתנאי הצירוף. תקן SQL2 מאפשר להגדיר את הצירוף הזה במשפט FROM בין שתי הטבלאות המשתתפות בצירוף.

תחביר הפקודה:



תרשים 8.32: תרשים תחביר לתנאי צירוף עם אפשרות לצירוף חיצוני.

דוגמה: הצג את רשימת ראשי המחלקות, שם המחלקה שבראשה הם עומדים ושמות הקורסים המועברים על ידי אותה המחלקה.

נתבונן בתרשים 8.33 ונשים לב שאם לא היינו מבצעים צירוף רגיל, השורה של המחלקה לכימיה לא היתה מופיעה, כי בטבלת הקורסים לא מופיע קורס כלשהו של מחלקה זו. שורה זו תופיע בטבלת התוצאה רק אם נבצע Full Join או Right Join, מאחר וטבלת הקורסים היא הטבלה הימנית בצירוף.

```
1. SELECT  HEAD, NAME,COURSE_NAME
2. FROM    DEPARTMENTS D FULL JOIN  COURSES  C
3.         ON  C.DEPARTMENT_ID  =  D.DEPART
```

HEAD	NAME	COURSE_NAME
Dr. Israel	Computer Science	Programming
Dr. Israel	Computer Science	Pascal
Dr. Israel	Computer Science	Data Base
Prof. Levy	Mathematics	Linear Algebra
Prof. Levy	Mathematics	Numeric Analysis
Dr. Eyal	Business	Marketing
Dr. Eyal	Business	Operations Res.
Prof. Doron	Chemistry	

תרשים 8.33: פלט של צירוף חיצוני.

צירוף טבלה אל עצמה (Self Join)

שפת SQL מאפשרת להתייחס מספר פעמים אל אותה טבלה במשפט FROM כאילו הן טבלאות שונות. כדי לאפשר אבחנה בין טבלאות זהות, יש להשתמש בשמות נרדפים. מקובל לקרוא לצירוף זה גם בשם Reflexive Join.

דוגמה : הצג רשימת כל זוגות מספרי סטודנטים שקיבלו את אותו ציון בקורס M-100.

```
1.  SELECT  A.STUDENT_ID, A.GRADE,
2.          B.STUDENT_ID, B.GRADE
3.  FROM    GRADES  A, GRADES  B
4.  WHERE   A.STUDENT_ID < B.STUDENT_ID AND
5.          A.GRADE = B.GRADE AND
6.          A.COURSE_ID = 'M-100' AND
7.          B.COURSE_ID = 'M-100'
```

A.STUDENT_ID	A.GRADE	B.STUDENT_ID	B.GRADE
210	90	245	90
200	90	210	90
200	90	245	90

זו דוגמה לצירוף כללי, הנקרא גם Theta Join, המשתמש ביחס 'קטן' (<) ולא בתנאי שוויון. כפי שניתן לראות משורה 3, בצירוף משתתפת אותה טבלה פעמיים. הדרישה המופיעה בשורה 4 שמספר סטודנט מטבלה B יהיה גדול ממספר הסטודנט מטבלה A, מבטיחה שלא נשלוף זוגות בהן אותו סטודנט מופיע בשתי הטבלאות. הדרך הטובה ביותר להבין צירוף זה היא לחשוב שקיימים שני עותקים זהים של טבלת הציונים, עותק אחד נקרא A והשני נקרא B. עכשיו, צריך לבצע צירוף פשוט בין שני העותקים על פי התנאים המופיעים במשפט WHERE.

מכפלה קרטזית בין טבלאות (Cartesian Product)

כפי שהוסבר בפרק של האלגברה הטבלאית, הצירוף הוא מקרה מיוחד של המכפלה הקרטזית בין טבלאות. שפת SQL תומכת במכפלה קרטזית בין שתי טבלאות. הטבלה התוצאתית תכיל את כל הצירופים האפשריים בין השורות של שתי הטבלאות.

דוגמה: יש לבצע מכפלה קרטזית בין טבלה המחלקות לטבלת הסטודנטים. למען הפשטות נניח שטבלת המחלקות וטבלת הסטודנטים מכילות רק שתי שורות.

Departments

DEPART	NAME	HEAD
CS	Computer Science	Dr. Israel
MT	Mathematics	Prof. Levy

Students

STUDENT_ID	NAME	CITY
105	Moshe	Haifa
210	Dan	Tel Aviv

תרשים 8.34: בסיס נתונים לדוגמה עבור מכפלה קרטזית.

נשים לב שהמכפלה הקרטזית בין הטבלאות נוצרת, כי בפקודה לא מופיע משפט WHERE עם תנאי לצירוף הטבלאות.

```
1. SELECT  D.*, S.*
2. FROM    DEPARMENTS A, STUDENTS  S
```

DEPART	D.NAME	HEAD	STUDENT_ID	S.NAME	CITY
CS	Computer Science	Dr. Israel	105	Moshe	Haifa
CS	Computer Science	Dr. Israel	210	Dan	Tel Aviv
MT	Mathematics	Prof. Levy	105	Moshe	Haifa
MT	Mathematics	Prof. Levy	210	Dan	Tel Aviv

תקן SQL2 מאפשר חלופה נוספת לכתיבת שאילתה של מכפלה קרטזית, תוך שימוש באפשרות CROSS JOIN בתוך משפט FROM:

```
1. SELECT  D.*, S.*
2. FROM    DEPARMENTS A CROSS JOIN   STUDENTS  S
```

תת-שאילתות (Sub Queries)

שפת SQL מאפשרת ביצוע פקודת SELECT מקוננת (Nested SELECT), כלומר הפעלת שאילתה בתוך שאילתה. היכולת הזו של קינון שאילתות היוותה את הסיבה המקורית להחלטה לקרוא לשפת SQL שפה **מובנית** והיא שהעניקה לשפה את השם Structured.

השאילתה ברמה הגבוהה ביותר תקרא שאילתה **ראשית** או שאילתה **חיצונית**, בעוד כל האחרות תקראנה **תת-שאילתות**. התוצאה של תת-שאילתה מוחזרת לתת-השאילתה ברמה הגבוהה יותר וכך הלאה עד לשאילתה הראשית. התוצאה של תת-שאילתה יכולה

לשמש לבדיקות קיום תנאים במשפט WHERE או במשפט HAVING. ניתן לבצע קינון שאילתות במספר רמות.

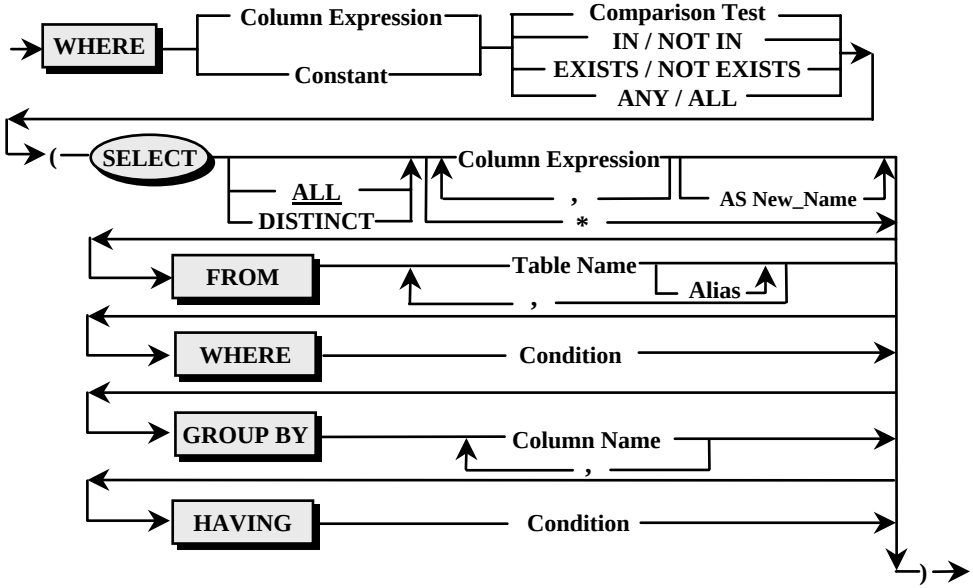
עיקרון הפעולה:

סדר ביצוע השאילתות הוא מתת-השאילתה הפנימית ביותר ועד לשאילתה הראשית. תת-השאילתה הראשונה מבוצעת בצורה רגילה, התוצאה שלה מוחזרת לתת-שאילתה ברמה גבוהה יותר וכך חוזר חלילה עד להחזרת התוצאה לשאילתה הראשית. עם החזרת התוצאה לשאילתה הראשית, השאילתה הראשית מבוצעת כשאילתה רגילה לכל דבר.

תחביר הפקודה :

תת-השאילתה מוקפת בסוגריים ומופיעה בתוך משפט WHERE של השאילתה הראשית. קיימים מספר הבדלים בין שאילתה רגילה לבין תת-שאילתה :

- תת-שאילתה חייבת לייצר טבלה תוצאתית המכילה עמודה אחת בלבד. משמעות הדבר היא שבמשפט SELECT של תת-השאילתה יכולה להופיע עמודה אחת בלבד, למעט חריג אחד – במקרה שמילת הקשר בין השאילתה לתת-השאילתה היא EXISTS.



תרשים 8.35: תרשים תחביר של תת-שאילתה.

- תת-שאילתה אינה יכולה להכיל מיון, כלומר לא ניתן להשתמש במשפט ORDER BY בתוך תת-שאילתה.

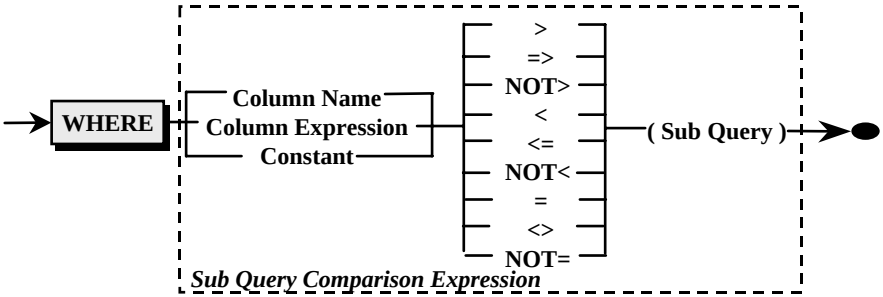
- תת-שאילתה יכולה להכיל פנייה לעמודות השייכות לטבלאות של השאילתה הראשית ולא רק לעמודות השייכות לטבלאות המופיעות במשפט FROM של תת-השאילתה.
- משפט WHERE של תת-שאילתה יכול להכיל בעצמו תת-שאילתה נוספת, כלומר זהו מבנה רקורסיבי.

בדיקת תנאי השוואה (Subquery Comparison Test)

נתחיל במצב הפשוט ביותר, שבו תת-השאילתה מחזירה ערך בודד בלבד. ניתן להשתמש בערך בודד זה לצורך השוואה עם ערכים של עמודה בשאילתה הראשית. ההשוואה יכולה להתבצע תוך שימוש באופרטורים ההשוואתיים הרגילים: =, <, >, =<, =>, <>.

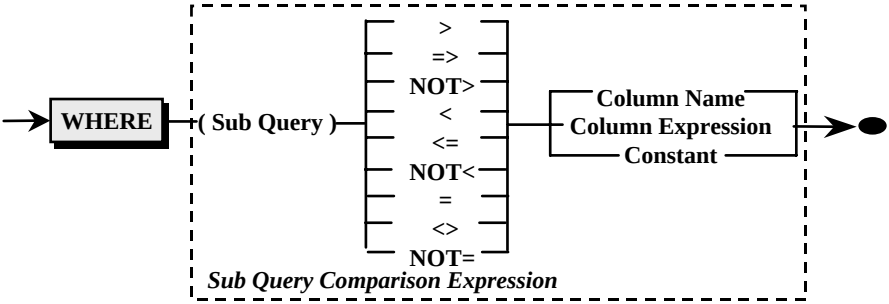
תחביר הפקודה:

תת-השאילתה יכולה להופיע באגף השמאלי או הימני של התנאי. שני התרשימים הבאים מציגים את שתי החלופות, שבהן תת-שאילתה נמצאת באגף ימין או באגף שמאל.



תרשים 8.36: תרשים תחביר לבדיקת תנאי השוואה, שבו תת-השאילתה באגף ימין.

בחלופה זו מופיע תחילה שם העמודה ואחר כך מופיעה תת-השאילתה.



תרשים 8.37: תרשים תחביר לבדיקת תנאי השוואה, שבו תת-השאילתה באגף שמאל.

דוגמה א' : הצג את רשימת כל הסטודנטים שקיבלו ציון סופי גבוה מאשר ממוצע הציונים בקורס C-200.

```
1. SELECT   DISTINCT STUDENT_ID
2. FROM     GRADES
3. WHERE    GRADE >
4.                (SELECT   AVG(GRADE)
5.                 FROM     GRADES
6.                 WHERE    COURSE_ID = 'C-200')
```

STUDENT_ID
105
200
210
245

תת-השאילתה שמוצגת בשורות 4 עד 6 מחשבת את ממוצע הציונים בקורס C-200 ומוסרת את התוצאה לשאילתה הראשית השולפת את כל השורות בהן הציון הסופי גדול מהציון הממוצע. מאחר והציון הממוצע במקרה שלנו הוא 84 יישלפו רק סטודנטים שהציון הסופי שלהם גדול מ-84.

דוגמה ב' : הצג את מספר הסטודנט והציון עבור כל הסטודנטים שקיבלו במבחן הסמסטריאלי ציון גבוה מהציון הגבוה ביותר שסטודנט כלשהו קיבל במבחן הסופי.

```
1. SELECT   STUDENT_ID, GRADE
2. FROM     GRADES
3. WHERE    GRADE_SEM >
4.                (SELECT   MAX  (GRADE)
5..               FROM      GRADES)
```

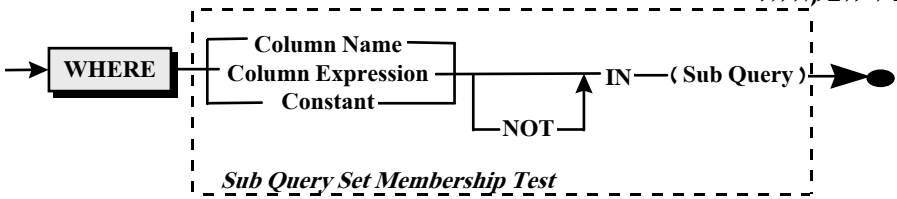
STUDENT_ID	GRADE
245	85
310	95
	100

תת-השאילתה המופיעה בשורות 4 ו-5 מחשבת את הציון הסופי הגבוה ביותר בטבלת הציונים. מכיון שהציון הסופי הגבוה ביותר הוא 90, נקבל רק שני סטודנטים שהציון הסמסטריאלי שלהם גבוה יותר.

בדיקת קבוצת ערכים (Set Membership Test)

תת-שאילתה יכולה להחזיר ערך בודד או קבוצת ערכים. כאשר תת-השאילתה מחזירה קבוצת ערכים, ניתן לבדוק אם הערכים של עמודה מהשאילתה הראשית מופיעים בתוך קבוצת הערכים המוחזרים על ידי תת-השאילתה. בדיקת קיום הערכים בקבוצה מתבצעת באמצעות אופרטור IN. פגשנו כבר אופרטור זה עבור בדיקת קיום בקבוצת ערכים, אולם הפעם קבוצת הערכים נבנית באופן דינמי על ידי תת-השאילתה להבדיל מהמצב הקודם שבו קבוצת הערכים מוגדרת מראש בעת כתיבת השאילתה.

תחביר הפקודה:



תרשים 8.38: תרשים תחביר לבדיקת קבוצת ערכים

דוגמה א' : הצג את שמות הסטודנטים שלמדו קורס אחד או יותר מבין הקורסים שסטודנט שמספרו 105 לא למד.

```
1.  SELECT    NAME, COURSE_ID
2.  FROM      GRADES  G, STUDENTS  S
3.  WHERE     G.STUDENT_ID = S.STUDENTS_ID AND
4.                COURSE_ID  NOT  IN
5.                (SELECT   COURSE_ID
6.                 FROM      GRADES
7.                 WHERE    STUDENT_ID = '105')
```

NAME	COURSE_ID
Dan	B-10
Yoel	B-10
David	B-10

תת-השאילתה המופיעה בשורות 5 ועד 7 מחזירה קבוצה המכילה את כל הקורסים שסטודנט 105 למד והם: C-55, M-100, C-200, B-40. טבלה זמנית זו מועברת לשאילתה הראשית הבודקת כל שורה בטבלת הציונים ושולפת רק שורות שמכילות מספר קורס שאינו מופיע ברשימה, כמוגדר בשורה 4.

דוגמה ב' : הצג את עיר המגורים ואת שם הסטודנט של כל הסטודנטים שלמדו לפחות שני קורסים מבין הקורסים ש-Dan או Eyal למדו. מיין את התוצאה לפי שם העיר.

```
1.  SELECT      CITY, NAME
2.  FROM        GRADES  G, STUDENTS  S
3.  WHERE       G.STUDENT_ID = S.STUDENTS_ID AND
4.                COURSE_ID  IN
5.                (SELECT   COURSE_ID
6.                FROM      GRADES G1, STUDENTS S1
7.                WHERE     NAME IN (' Dan', 'Ran') AND
8.                          G1.STUDENT_ID = S1.STUDENT_ID)
9.   GROUP BY NAME, CITY
10. HAVING      COUNT(*) >= 2
11. ORDER BY CITY
```

CITY	NAME
Haifa	Yoel
Tel Aviv	Dan
Tel Aviv	David

תת-השאילתה שבשורות 5 ועד 8 בודקת איזה קורסים למדו שני הסטודנטים. Ran לא למד באף קורס ואילו Dan למד בקורסים M-100, C-200 ו B-10, לכן זו הופכת להיות הקבוצה שמועברת לשאילתה הראשית. השאילתה הראשית סורקת את השורות של טבלת הציונים ומוצאת את כל מספרי הסטודנטים שלמדו באחד מהקורסים בקבוצה ומצרפת שורות אלו לטבלת הסטודנטים. השאילתה בונה את ההקבצות לפי העיר ושם הסטודנט וסופרת את מספר השורות בכל הקבצה. לפי הרשום בשורה 10 יישלפו רק הקבצות בהן יש שתי שורות ומעלה.

דוגמה ג' : הצג את רשימת הקורסים שבהם סטודנט כלשהו קיבל ציון זהה לאחד הציונים שקיבל הסטודנט ששמו משה, או שסטודנטים שלמדו בקורס מסוג סמינר קיבלו.

```
1.  SELECT    DISTINCT G.COURSE_ID
2.  FROM      GRADES  G
3.  WHERE     GRADE  IN
4.                (SELECT  GRADE
5.                FROM     GRADES G1, STUDENTS S1
6.                WHERE    NAME = 'Moshe' AND
7.                         G1.STUDENT_ID = S1.STUDENT_ID)
8.                 OR GRADE  IN
9.                (SELECT  GRADE
10.               FROM     GRADES G2, COURSES C
11.               WHERE    TYPE = 'SEMIN' AND
12.                        G2.COURSE_ID = C.COURSE_ID)
13. ORDER BY COURSE_ID
```

COURSE_ID
B-10
B-40
C-55
C-200
M-100

זו דוגמה של שתי תת-שאילתות שונות ולא מקוננות המשתתפות בתוך אותה שאילתה. תת-השאילתה הראשונה המופיעה בשורות 4 ועד 7 שולפת את כל הציונים של משה ובונה את הקבוצה שמכילה את הציונים 58, 75, 90, 70. תת-השאילתה השנייה המופיעה בשורות 9 ועד 12 שולפת את כל הציונים שקיבלו סטודנטים בקורס Operations Res. שהוא היחיד מסוג סמינר ובונה את הקבוצה שמכילה את הציונים 70 ו-85. השאילתה הראשית סורקת את השורות של טבלת הציונים ובודקת כל ציון מול שתי הקבוצות הנ"ל ובונה את הטבלה התוצאתית. נשים לב שבשורה 8 מופיע תנאי OR המקשר בין שתי תת-השאילתות.

השוואת ערך בודד מול ערך כלשהו (Any)

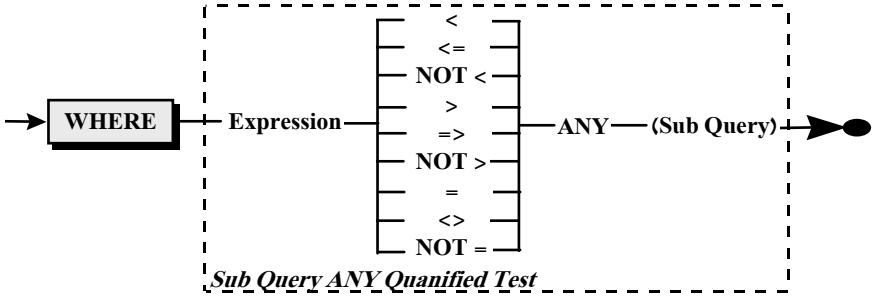
ניתן לקשר בין השאילתה הראשית לתת-השאילתה גם באמצעות אופרטור ANY. אופרטור זה בודק ערך כלשהו של השאילתה הראשית מול קבוצת ערכים המוחזרים מתת-השאילתה ומחזיר את הערך True אם המבחן ההשוואתי מתקיים **לפחות פעם אחת**.

עיקרון הפעולה:

האופרטור ANY בודק את הערך המבוקש מול כל אחד מהערכים המוחזרים על ידי תת-השאילתה בהתאם להשוואה המבוקשת. אם הערך הנבדק מקיים את ההשוואה המבוקשת לפחות פעם אחת, הערך המוחזר לשאילתה הראשית יהיה True. למשל אם התנאי בתת-השאילתה הבודקת הוא a >= ANY אזי הערך a ייבדק מול כל אחד מהערכים המוחזרים על ידי תת-השאילתה; ואם הוא גדול או שווה לאחד או יותר מהערכים המוחזרים, התוצאה תהיה True. אם a קטן מכל הערכים המוחזרים, התוצאה תהיה False.

עקרונית, יכולנו גם לבקש מתת-השאילתה להחזיר את הערך המינימלי תוך שימוש בפונקציה המובנית MIN ולדרוש a >= MIN ולקבל את אותה תוצאה.

תחביר הפקודה:



תרשים 8.39: תרשים תחביר לאופרטור ANY

דוגמה : הצג את מספר הסטודנט עבור כל הסטודנטים שקיבלו ציון גבוה יותר מציון כלשהו שקיבל סטודנט שמספרו 210 בכל הקורסים בהם למד.

```
1.  SELECT      DISTINCT STUDENT_ID
2.  FROM        GRADES
3.  WHERE       GRADE > ANY
4.                (SELECT   GRADE
5.                 FROM     GRADES
6.                 WHERE    STUDENT_ID = '210' )
7.  GROUP BY STUDENT_ID
```

STUDENT_ID
105
200
210
245

תת-השאילתה, הרשומה בשורות 4 עד 6, שולפת מטבלת הציונים את כל הציונים שסטודנט שמספרו 210 קיבל: 90, 85, 78. עכשיו השאילתה הראשית מתחילה לסרוק פעם נוספת את השורות של טבלת הציונים ובודקת את הציון של כל סטודנט מול קבוצת הערכים שהתקבלה. מספיק שהציון של הסטודנט גדול מאחד מהערכים בקבוצה כדי שהאופרטור יחזיר את הערך True והשורה תישלף. בדוגמה שלנו רק הסטודנט שמספרו 310 לא מקיים את התנאי, מאחר והוא קיבל ציון 65 שאינו גדול מאף ערך בקבוצה.

השוואת ערך בודד מול כל הערכים (All)

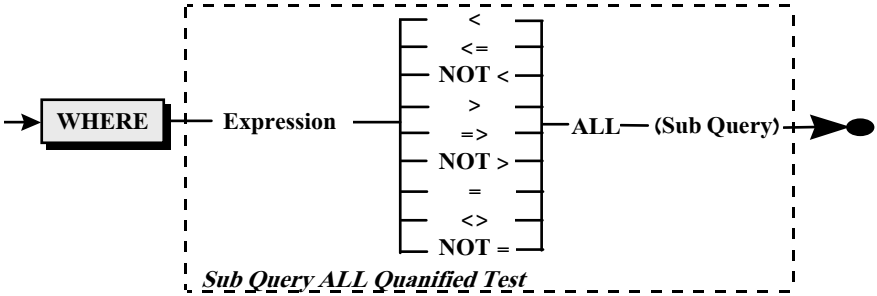
ניתן לקשר בין השאילתה הראשית לתת-השאילתה גם באמצעות אופרטור ALL. אופרטור זה בודק ערך כלשהו של השאילתה הראשית מול קבוצת ערכים המוחזרים מתת-השאילתה ומחזיר True אם מבחן ההשוואה מתקיים עבור **כל הערכים** בקבוצה.

עיקרון הפעולה:

האופרטור ALL בודק את הערך המבוקש מול כל אחד מהערכים המוחזרים על ידי תת-השאילתה על פי ההשוואה המבוקשת. אם הערך הנבדק מקיים את תנאי ההשוואה עבור **כל** הערכים בקבוצה, הערך המוחזר לשאילתה הראשית יהיה True. למשל, אם תת-השאילתה הבודקת היא a >= ALL, הערך a ייבדק מול כל אחד מהערכים המוחזרים על ידי תת-השאילתה; ואם הוא גדול או שווה מכל הערכים המוחזרים, התוצאה תהיה True. אם a קטן לפחות מערך אחד מהערכים המוחזרים, התוצאה תהיה False.

עקרונית, יכולנו גם לבקש מתת-השאילתה להחזיר את הערך המקסימלי תוך שימוש בפונקציה המובנית MAX ולדרוש a >= MAX ולקבל את אותה תוצאה.

תחביר הפקודה:



תרשים 8.40: תרשים תחביר לאופרטור ALL.

דוגמה : הצג את מספר הסטודנט של כל הסטודנטים שקיבלו לפחות ציון אחד שהוא נמוך מכל הציונים שקיבל סטודנט שמספרו 210 בכל הקורסים בהם למד.

```
1. SELECT     DISTINCT STUDENT_ID
2. FROM       GRADES
3. WHERE      GRADE <  ALL
4.             (SELECT GRADE
5.              FROM    GRADES
6.              WHERE  STUDENT_ID = '210' )
7. GROUP BY STUDENT_ID
```

STUDENT_ID
105
310

תת-השאילתה המופיעה בשורות 4 עד 6 שולפת מטבלת הציונים את כל הציונים ששסטודנט שמספרו 210 קיבל והם 90, 85, 78. עכשיו השאילתה הראשית מתחילה לסרוק פעם נוספת את השורות של טבלת הציונים ובודקת את הציון של כל סטודנט מול קבוצת הערכים שהתקבלה. כדי ששורה תישלף מהטבלה הציון שרשום בה חייב להיות נמוך מכל הציונים בקבוצה. בדוגמה שלנו רק לסטודנטים שמספרם 105 ו-310 יש לפחות ציון אחד שהוא נמוך יותר מהציונים בקבוצה.

תת-שאילתות מתואמות (Correlated Sub Queries)

תת-שאילתות מתואמות מתאפיינות בכך שהעברת הנתונים בהן מתבצעת בכיוון הפוך, מהשאילתה הראשית אל תת-השאילתה.

עיקרון הפעולה:

בדרך כלל, תת-שאילתה רגילה מבוצעת לפני השאילתה הראשית ומעבירה אליה תוצאה כלשהי, כלומר סדר הביצוע הוא מבפנים החוצה. בשאילתות מתואמות סדר הביצוע הוא הפוך – מבחוץ פנימה. השאילתה הראשית סורקת את השורות ועל כל שורה היא מפעילה את תת-השאילתה לאחר שהיא מעבירה אליה פרמטרים כלשהם. תת-השאילתה מתבצעת

עם הפרמטרים המבוקשים ואם היא מחזירה ערך True, השורה של השאילתה הראשית נשלפת. אם תת-השאילתה מחזירה את הערך False, השורה אינה נשלפת.

הדרך שבה שאילתות אלו פועלות היא באמצעות הפנייה חיצונית (Outer Reference), כלומר שימוש בשמות של עמודות השייכות לשאילתה הראשית בתוך המשפט WHERE של תת-השאילתה. הפנייה החיצונית יכולה להיות ברמה כלשהי, כלומר תת-שאילתה מקוננת ברמה השלישית יכולה לפנות לעמודה של השאילתה הראשית, ולא לפנות לעמודות של תת-השאילתה ברמה השנייה.

דוגמה : הצג את כל פרטי טבלת הציונים עבור סטודנט שקיבל את הציון הגבוה ביותר בכל מקצוע. אם נכתוב שאילתה זו בצורה הבאה:

```
1. SELECT  *
2. FROM    GRADES
3. WHERE  GRADE =
4.            (SELECT MAX (GRADE)
5.             FROM    GRADES)
```

תת-השאילתה המופיעה בשורות 4 ו-5 תשלוף את הציון הגבוה ביותר בטבלת הציונים, 90, ולאחר מכן השאילתה הראשית תשלוף רק שורות שבהן הציון גבוה מ-90. ברור שזו לא היתה כוונת השאילתה. הדרך הנכונה לבצע בקשה זו היא לסרוק את טבלת הציונים, למסור לתת-שאילתה את מספר הקורס ולתת לה לשלוף את הציון הגבוה ביותר עבור אותו קורס. ציון זה, שהוא שונה לכל קורס ישמש את השאילתה הראשית לבדיקה אם לשלוף את השורה או לא, וכך חוזר חלילה עבור כל השורות בטבלת הציונים. זהו עיבוד מורכב יותר מתת-שאילתה רגילה המתבצעת פעם אחת בלבד. כאן, תת-השאילתה מתבצעת מחדש עבור כל שורה של השאילתה הראשית.

נציג עכשיו את הדרך לבצע בקשה זו ב-SQL תוך שימוש בפנייה חיצונית ותת-שאילתות מתואמות.

```
1. SELECT  *
2. FROM    GRADES  A
3. WHERE  GRADE =
4.            (SELECT MAX (GRADE)
5.             FROM    GRADES  B
6.             WHERE  A.COURSE_ID = B.COURSE_ID)
```

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE	GRADE_SEM
105	C-55	SUM1998	A	58	70
210	M-100	AUT1999	A	90	90
245	M-100	AUT1999	A	90	80
200	M-100	SUM1998	B	90	90
105	C-200	AUT1998	A	90	85
245	B-10	AUT1999	A	80	70
245	B-40	WIN1999	A	85	95

שורה 6 בתת-שאילתה מכילה את הפנייה החיצונית שבה מועבר מספר הקורס מהשאילתה הראשית אל תת-השאילתה. בטבלה זו קיבלנו לכל מקצוע את פרטי הסטודנטים שהשיגו את הציון הגבוה ביותר בכל קורס.

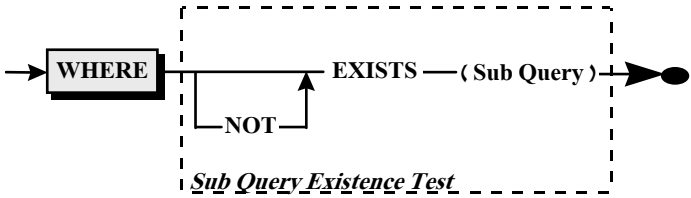
בדיקת קיום (Existence Test)

בדיקה זו מאפשרת לבחון אם תת-שאילתה ב**שאילתה מתואמת** (Correlated Queries) החזירה שורות כלשהן ומחזירה ערך True או False בהתאם. זהו המקרה היחיד בו מותר לתת-שאילתה להחזיר מספר כלשהו של עמודות ולא רק עמודה בודדת, וזאת, מאחר והבדיקה בוחנת רק אם נשלפו שורות ולא בוחנת את תוכן השורות.

עיקרון הפעולה:

השאילתה הראשית מעבירה לתת-השאילתה את הפרמטרים על ידי שימוש בפנייה חיצונית. תת-השאילתה מתבצעת ואם היא מוצאת שורות כלשהן שעונות על התנאי היא מחזירה את הערך True לשאילתה הראשית. אם היא אינה מוצאת שורות היא מחזירה את הערך False. ניתן להפוך את כיוון הבדיקה על ידי שימוש במילת השלילה NOT.

תחביר הפקודה:



תרשים 8.41: תרשים תחביר לבדיקת קיום שורות בתת-שאילתה.

דוגמה א' : הצג את שם הקורס, שם המחלקה שמעבירה אותו ואת שם ראש המחלקה עבור כל הקורסים שיש להם לפחות סטודנט אחד שנבחן במועד ב' בסמסטר סתיו 1999 וקיבל ציון בין 60 ל-80.

```
1.  SELECT COURSE_NAME, D.NAME,HEAD
2.  FROM    GRADES  G, STUDENTS  S,
3.              COURSES C, DEPARTMENTS D
4.  WHERE  G.STUDENT_ID = S.STUDENTS_ID AND
5.              G.COURSE_ID = C.COURSE_ID AND
6.              C.DEPARTMENT_ID = D.DEPART AND
7.              EXISTS
8.                 (SELECT COURSE_ID
9.                  FROM    GRADES G1
10.                 WHERE  TERM = 'B' AND
11.                        GRADE BETWEEN 60 AND 80 AND
12.                        SEMESTER = 'AUT1999' AND
13.                        G.COURSE_ID = G1.COURSE_ID )
```

COURSE_NAME	NAME	HEAD
Programming	Computer Science	Dr. Israel

נשים לב לפנייה החיצונית שרשומה בשורה 13. שורה זו מגדירה תנאי שוויון בין מספר הקורס של טבלת הציונים בשאילתה הראשית לבין מספר הקורס בטבלת הציונים בתת-השאילתה. הדרך להבין פנייה חיצונית זו היא זו: השאילתה הראשית סורקת את טבלת

הציונים ועבור כל שורה היא מעבירה לתת-השאילתה את מספר הקורס ומבצעת את תת-השאילתה. אם תת-השאילתה מוצאת שקיימות שורות שעונות על התנאי, היא מחזירה את הערך True ולכן השאילתה הראשית שולפת את השורה ועוברת לשורה הבאה וכך חוזר חלילה עבור כל השורות בטבלת הציונים.

בדוגמה שלנו, תת-השאילתה תחזיר את הערך True רק עבור השורה של סטודנט 200 וקורס C-200. לכן בטבלה התוצאתית קיבלנו שורה אחת בלבד.

דוגמה ב' : הצג את שם המחלקה ושם ראש המחלקה שאין לה כל קורס בטבלת הקורסים.

```
1.  SELECT NAME, HEAD
2.  FROM    DEPARTMENTS D
3.  WHERE NOT EXISTS
4.              (SELECT   *
5.               FROM      COURSES C
6.               WHERE    D.DEPART = C.DEPARTMENT_ID )
```

NAME	HEAD
Chemistry	Prof. Doron

בשאילתה זו מופיעה הפנייה החיצונית בשורה 6. השאילתה הראשית סורקת את שורות טבלת המחלקות ועל כל שורה מעבירה לתת-השאילתה את מספר המחלקה ומבקשת לבדוק אם יש שורות בטבלת הקורסים למחלקה המסוימת. אם אין שורות, אזי שורת המחלקה תישלף אל טבלה התוצאה. מכיון שרק למחלקה לכימיה אין אף קורס, היא היחידה שתישלף.

תת-שאילתות במשפט HAVING

עד כה התייחסנו לתת-שאילתות במשפט WHERE. שפת SQL מאפשרת להשתמש בתת-שאילתות גם במשפט HAVING המשמש לבדיקת תנאים על שורות של הקבצות. השימוש והתחביר לתת-שאילתות במשפט HAVING הוא רגיל, ודומה לתחביר תת-שאילתות המופיעות במשפט WHERE.

תת-שאילתות אלו יכולות להיות גם שאילתות מתואמות המשתמשות בפנייה חיצונית. מכיון שהן מופיעות במשפט HAVING הן תבוצענה פעם אחת לכל הקבצה ולא לכל שורה.

דוגמה : הצג את מספר הקורס, את שם הקורס ואת הציון הסופי הממוצע לכל הקורסים שמעניקים 3 נקודות זכות ובתנאי שהציון הממוצע לקורס גדול מהציון הסופי הממוצע של כל הקורסים מסוג מעבדה.

```
1.  SELECT      COURSE_ID, COURSE_NAME,
2.                AVG (GRADE)
3.  FROM         GRADES  G, COURSES C
4.  WHERE       POINTS = 3
5.  GROUP BY COURSE_ID, COURSE_NAME
6.  HAVING      AVG (GRADE) >
7.                  (SELECT AVG (GRADE)
8.                   FROM     GRADES  G1, COURSES C1
9.                   WHERE  G1.COURSE_ID = C1.COURSE_ID
10.                          AND TYPE = 'LAB')
```

COURSE_ID	COURSE_NAME	COL3

השאילתה הראשית בונה את ההקבצות לכל הקורסים שמעניקים 3 נקודות זכות, כלומר לקורסים C-55, M-100, M-200 ו- B-40. לקורס C-55 ממוצע הציונים הוא 58, לקורס M-100 ממוצע הציונים הוא 82, לקורס M-200 אין ציונים ולקורס B-40 ממוצע הציונים הוא 77.5. עכשיו מופעלת תת-השאילתה המופיעה בשורות 7 עד 10 והיא בודקת מהו הציון הממוצע עבור כל הקורסים מסוג מעבדה, כלומר לקורסים C-200 ו- C-300. מכיון שלקורס C-300 אין ציונים, הממוצע הכולל יהיה הממוצע של קורס C-200 והוא 84.3. לכן, יישלפו רק הקבצות שהציון הממוצע שלהם גבוה מ-84.3 ומכיון שאין אף הקבצה כזאת, נקבל טבלת תוצאה ריקה.

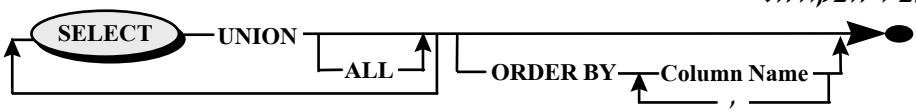
איחוד תוצאות של שאילתות (Union)

שפת SQL מאפשרת לאחד את התוצאות של מספר שאילתות לטבלה תוצאתית אחת באמצעות האופרטור UNION. השאילתות שאת תוצאותיהן מבקשים לאחד חייבות להחזיר תוצאות במבנה זהה. מקובל לקרוא לשאילתות המחזירות תוצאות במבנה זהה שאילתות **תואמות איחוד** (Union Compatible). משמעות הדבר היא שטבלאות התוצאות הן במבנה דומה מבחינת מספר העמודות, מבחינת טיפוס הנתונים של העמודות ומבחינת המשמעות של העמודות.

עיקרון הפעולה:

בשלב ראשון, כל השאילתות המשתתפות באיחוד מבוצעות ומייצרות טבלאות ביניים. בשלב שני, מתבצע האיחוד של כל טבלאות הביניים לקבלת הטבלה התוצאתית. במידת הצורך, תוך כדי ביצוע האיחוד מתבצע גם סילוק של שורות כפולות.

תחביר הפקודה:



תרשים 8.42: תרשים תחביר לפקודה UNION

אופרטור האיחוד מאפשר לאחד שתי טבלאות תוצאתיות המתקבלות כתוצאה מהפעלת פקודות SELECT ולקבל טבלה תוצאתית מאוחדת. ברירת המחדל של אופרטור האיחוד היא סילוק שורות כפולות. אם מבקשים בכל זאת לשמור על השורות הכפולות יש להשתמש בפרמטר UNION ALL. מאחר והפעולה לסילוק שורות כפולות יקרה מבחינת משאבי המחשב, מומלץ לבחור בפרמטר UNION ALL, אלא אם יש סיבה מיוחדת שבגללה מבקשים לסלק את השורות הכפולות. נשים לב שברירת מחדל זו היא **הפוכה** לברירת המחדל של הפקודה SELECT שאינה מסלקת שורות כפולות, אלא אם מבקשים זאת במפורש באמצעות ההוראה DISTINCT. אין מגבלה למספר הטבלאות שמבקשים לאחד.

התקן קובע שמשפט SELECT המשתתף באיחוד יכול להכיל רק שמות של עמודות או כוכבית (*) לבחירת כל העמודות, אולם אינו יכול להכיל ביטויים ופונקציות מובנות. למרות זאת, רוב המערכות המסחריות מתעלמות מאיסור זה ומאפשרות זאת.

לעומת זאת, למרות שהתקן אינו אוסר, רוב המערכות המסחריות אינן מרשות משפטי GROUP BY ו-HAVING בפקודות SELECT המשתתפות באיחוד.

אסור להשתמש במשפט ORDER BY בתוך פקודות SELECT המשתתפות בפעולת איחוד. הסיבה לכך היא שממילא אין כל משמעות למיון טבלאות הביניים, מאחר וממילא הן עוברות לאחר מכן תהליך של איחוד. יחד עם זאת, ניתן לבקש מיון של התוצאה על ידי הוספת משפט ORDER BY לאחר הפקודה SELECT האחרונה המשתתפת באיחוד.

דוגמה : שלוף את כל נתוני הקורסים המעניקים 3 נקודות זכות או קורסים מסוג CLASS.

```
1. SELECT  *
2. FROM    COURSES C
3. WHERE   POINTS = 3
4. UNION
5. SELECT  *
6. FROM    COURSES C
7. WHERE   TYPE = 'CLASS'
```

COURSE_ID	COURSE_NAME	TYPE	POINTS	DEPARTMENT_ID
C-55	Data Base	CLASS	3	CS
M-100	Linear Algebra	CLASS	3	MT
M-200	Numeric Analysis	CLASS	3	MT
B-40	Operations Res.	SEMIN	3	BS

בשורות 1 עד 3 מופיעה השאילתה הראשונה, בשורות 5 עד 7 מופיעה השאילתה השנייה ושורה 4 מכילה את אופרטור האיחוד. עקרונית ניתן לבצע בקשה זו גם על ידי שימוש בפקודת SELECT אחת, אך כאן הצגנו חלופה נוספת לביצוע השאילתה תוך שימוש באיחוד.

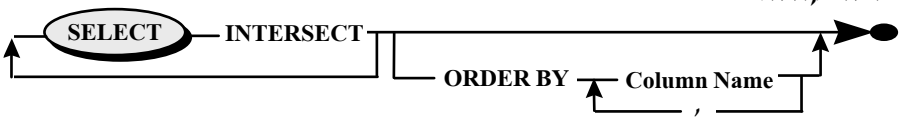
חיתוך תוצאות של שאילתות (Intersect)

שפת SQL מאפשרת לבצע חיתוך בין התוצאות של מספר שאילתות וקבלת טבלת תוצאה אחת באמצעות האופרטור INTERSECT. השאילתות שאת תוצאותיהן מבקשים לאחד חייבות להחזיר תוצאות במבנה אחיד, כלומר כל התוצאות הן **תואמות איחוד**.

עיקרון הפעולה:

בשלב ראשון, כל השאילתות המשתתפות בחיתוך מבוצעות ומייצרות טבלאות ביניים. בשלב שני, מתבצע החיתוך של כל טבלאות הביניים לקבלת הטבלה התוצאתית, כך שרק שורות המופיעות בכל טבלאות הביניים מועברות אל הטבלה התוצאתית.

תחביר הפקודה:



תרשים 8.43: תרשים תחביר לפקודה INTERSECT.

אופרטור החיתוך פועל בין שתי טבלאות תוצאה המתקבלות מהפעלת פקודות SELECT, ומפיק טבלת תוצאה המכילה רק את השורות המופיעות בשתי הטבלאות גם יחד.

אסור להשתמש במשפט ORDER BY בתוך פקודות SELECT המשתתפות בפעולת החיתוך. הסיבה לכך היא שממילא אין כל משמעות למיון טבלאות הביניים, כי ממילא הן עוברות לאחר מכן תהליך חיתוך. יחד עם זאת, ניתן לבקש מיון של התוצאה על ידי הוספת משפט ORDER BY לאחר פקודת SELECT האחרונה המשתתפת בחיתוך.

דוגמה : הצג את כל נתוני הקורסים המעניקים שלוש נקודות זכות והם גם מסוג CLASS.

```
1. SELECT  *
2. FROM     COURSES C
3. WHERE  POINTS = 3
4. INTERSECT
5. SELECT  *
6. FROM     COURSES C
7. WHERE  TYPE = 'CLASS'
```

COURSE_ID	COURSE_NAME	TYPE	POINTS	DEPARTMENT_ID
C-55	Data Base	CLASS	3	CS
M-100	Linear Algebra	CLASS	3	MT
M-200	Numeric Analysis	CLASS	3	MT

עקרונית ניתן לבצע בקשה זו גם על ידי שימוש בפקודת SELECT אחת, אך כאן הצגנו חלופה נוספת לביצוע בקשה זו באמצעות חיתוך.

פקודות לעדכון בסיס הנתונים

עד כאן הוצגו האפשרויות השונות לשליפת נתונים מתוך בסיס נתונים טבלאי באמצעות הפקודה SELECT של שפת SQL. כפי שכבר צוין בתחילת הפרק, שפת SQL איננה שפת שאילתות בלבד כפי שניתן אולי להבין משמה, אלא שפה מלאה לטיפול בנתונים המאפשרת גם את עדכון בסיס הנתונים. לכן יש להתייחס אליה כאל שפה כוללת ושלמה לתחזוקת בסיסי נתונים טבלאיים. בהשוואה למורכבות הפקודה SELECT, שאר הפקודות לעדכון בסיס הנתונים פשוטות יחסית. בסעיף זה נסקור את הפקודות להוספת שורות, לעדכון שורות ולביטול שורות מתוך טבלאות.

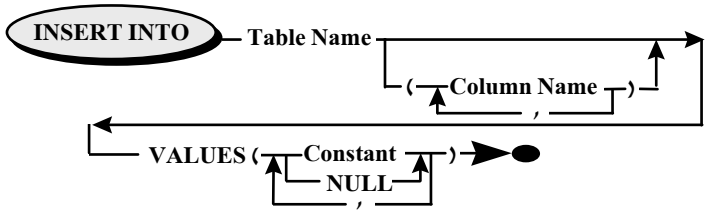
הוספת שורה בודדת (Single Row Insert)

פקודת INSERT מאפשרת הוספת שורות חדשות לטבלה קיימת.

עיקרון פעולה:

פקודה זו מוסיפה שורה חדשה לבסיס הנתונים. הפקודה מכילה את שם הטבלה אליה יש להוסיף את השורה החדשה ואת הערכים שעל השורה החדשה להכיל.

תחביר הפקודה:



תרשים 8.44: תרשים תחביר לפקודה INSERT.

אם עמודה מסוימת היא בעלת טיפוס נתונים של תאריך, ניתן לרשום CURRENT DATE במקום לרשום ערך של תאריך. לשורה יוכנס התאריך הנוכחי.

דוגמה א' : הוסף מחלקה חדשה לטבלת המחלקות.

```
1 . INSERT INTO DEPARTMENTS
2 .                (DEPART, NAME, HEAD)
3 . VALUES   ( 'EG' ,'Engineering' ,'Prof. Ziv')
```

זו הגירסה המלאה של הפקודה. מאחר ובמקרה זה אנו קובעים ערכים לכל העמודות, אין צורך לחזור על שמות כל העמודות של הטבלה, כפי שמופיע בשורה 2. לכן ניתן לרשום:

```
1 . INSERT INTO DEPARTMENTS
2 . VALUES  ( 'EG' ,'Engineering' ,'Prof. Ziv')
```

לאחר ביצוע פקודה זו תיווצר שורה חדשה בטבלת המחלקות.

לפני הוספה

DEPART	NAME	HEAD
CS	Computer Science	Dr. Israel
MT	Mathematics	Prof. Levy
BS	Business	Dr. Eyal
CH	Chemistry	Prof. Doron

INSERT INTO

לאחר הוספה

DEPART	NAME	HEAD
CS	Computer Science	Dr. Israel
MT	Mathematics	Prof. Levy
BS	Business	Dr. Eyal
CH	Chemistry	Prof. Doron
EG	Engineering	Prof. Ziv

תרשים 8.45: מצב טבלת המחלקות לפני ואחרי הוספת השורה.

דוגמה ב' : הוסף מחלקה חדשה לטבלת המחלקות. עדיין לא נקבע ראש המחלקה.

```
1 .  INSERT INTO DEPARTMENTS
2 .                (DEPART, NAME)
3 .  VALUES (  'MD' ,'Medicine')
```

במקרה זה, שורה 2 מציינת את שמות העמודות שהערכים החדשים יוכנסו בהן. העמודה HEAD לא קיבלה עדיין ערך כלשהו ולכן יוצב בה הערך Null, שהוא ציון מיוחד שמשמעותו "ערך חסר", כדי להבחין בין מצב זה לבין מצב שבו מוצב ערך ריק (Space). בהמשך נחזור לעניין ערכים חסרים.

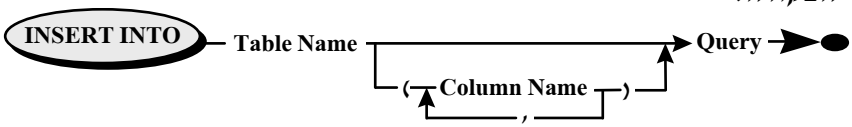
הוספת מספר שורות (Multi-row Insert)

פקודה זו מאפשרת הוספת מספר שורות לטבלה תוך שימוש בשאילתה לשליפת השורות מטבלה אחרת.

עיקרון פעולה:

פקודה זו מאפשרת שליפת שורות מתוך טבלה קיימת בבסיס הנתונים תוך שימוש בפקודה SELECT והוספת כל השורות האלו לטבלה אחרת בבסיס הנתונים.

תחביר הפקודה:



תרשים 8.46: תרשים תחביר לפקודה Multi Row Insert.

השאילתה לשליפת השורות אינה יכולה להכיל ORDER BY וחייבת להיות זהות בין מספר העמודות המופיעות במשפט SELECT לבין מספר העמודות של טבלת היעד.

דוגמה : נבנה טבלה חדה שתכיל רק את ההישגים של הסטודנטים במחלקה למדעי המחשב. הטבלה החדשה תכיל רק את העמודות מספר סטודנט, מספר קורס וציון.

```
1.  INSERT  INTO CS_GRADES
2.                SELECT STUDENT_ID, COURSE_ID, GRADE
3.                FROM    GRADES  G JOIN COURSES  C ON COURSE_ID
4.                  WHERE  C.DEPARTMENT_ID =      'CS'
```

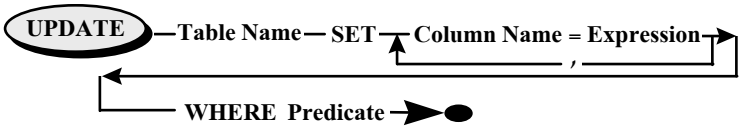
עדכון שורות (Update)

שפת SQL מאפשרת עדכון של הערכים בשורה בודדת או עדכון בו-זמני של מספר שורות המקיימות תנאי לוגי כלשהו.

עיקרון הפעולה:

מתבצעת סריקה של השורות. שורות המקיימות את התנאי המופיע במשפט WHERE תעודכנה, כלומר בעמודה המתאימה יוכנס הערך החדש.

תחביר הפקודה:



תרשים 8.47: תרשים תחביר לפקודת עדכון.

המשפט SET (קבע, הצב) מגדיר את הערך החדש של העמודות המופיעות בפקודה. כל עמודה יכולה להופיע רק פעם אחת בפקודת UPDATE. הביטוי במשפט SET יכול להכיל קבוע או ביטוי המכיל עמודה, אופרטורים אלגבריים וקבועים. העדכון יבוצע רק עבור שורות המקיימות את התנאי המופיע במשפט WHERE. התחביר של התנאי במשפט WHERE דומה לזה של שאילתה רגילה. התנאי יכול גם להיות גם תת-שאילתה.

דוגמה א' : עדכן את עיר המגורים של סטודנט מספר 105 מחיפה לירושלים.

```
1. UPDATE   STUDENTS
2.              SET   CITY = 'Jerusalem'
3.              WHERE    STUDENT_ID = '105'
```

דוגמה ב' : הוסף 5 נקודות לציון הסופי של כל הסטודנטים שלמדו בקורס M-100 בקיץ 1998.

```
1. UPDATE   GRADES
2.              SET       GRADE = GRADE + 5
3.              WHERE COURSE_ID = 'M-100' AND
4.                        SEMESTER = 'SUM1998'
```

דוגמה ג' : הוסף 10% לציון של כל הסטודנטים שלמדו בקורס מסוג מעבדה במחלקה למדעי המחשב.

```
1. UPDATE GRADES
2.          SET   GRADE = GRADE *  1.1
3.          WHERE  COURSE_ID  IN
4.                 (SELECT COURSE_ID
5.                  FROM     COURSES
6.                   WHERE  TYPE = 'LAB' AND
7.                          DEPARTMENT_ID = 'CS')
```

בדוגמה זו השתמשנו בתת-שאילתה המופעה בשורות 4 עד 7, המגדירה קבוצה של מספרי קורס. העדכון יבוצע עבור כל השורות מטבלת הציונים שמספר הקורס הוא אחד מתוך הקבוצה הזאת.

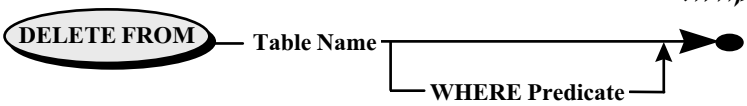
ביטול שורות (Delete)

שפת SQL מאפשרת לבטל מטבלה שורה בודדת או אוסף שורות, על פי תנאי לוגי כלשהו.

עיקרון פעולה:

מתבצעת סריקה של שורות הטבלה, וכתוצאה תבוטלנה כל השורות שמקיימות את התנאי שמוצג במשפט WHERE.

תחביר הפקודה:



תרשים 8.48: תרשים תחביר לפקודת ביטול.

משפט WHERE מגדיר את התנאי שכל השורות שתבוטלנה צריכות לקיים. התחביר של התנאי במשפט WHERE דומה לזה של שאילתה רגילה. התנאי יכול להיות גם תת-שאילתה. הפקודה DELETE ללא משפט WHERE תבטל את כל השורות בטבלה.

דוגמה א' : בטל את השורה של סטודנט מספר 210 בקורס B-10.

```
1. DELETE  GRADES
2.            WHERE  STUDENT_ID = '210' AND
3.                   COURSE_ID = 'B-10'
```

בדוגמה זו תבוטל שורה בודדת, מאחר ובשורה 2 ו-3 רשמנו את שדה המפתח של הטבלה.

דוגמה ב' : בטל את כל הציונים של סטודנט שמספרו 210.

```
1. DELETE  GRADES
2.            WHERE  STUDENT_ID = '210'
```

בדוגמה זו תבוטלנה כל השורות של סטודנט שמספרו 210.

דוגמה ג' : בטל את כל הציונים של קורס מסוג סמינר.

```
1. DELETE  GRADES
2.            WHERE  COURSE_ID  IN
3.                   (SELECT  COURSE_ID
4.                    FROM     COURSES
5.                    WHERE   TYPE = 'SEMIN')
```

בדוגמה זו, להגדרת השורות שתבוטלנה השתמשנו בתת-שאילתה. לא ניתן להתייחס בתת-השאילתה אל הטבלה שבה מבטלים שורות, כלומר בדוגמה שלנו לא ניתן להתייחס בתת-השאילתה לטבלת הציונים.

דוגמה ד' : בטל את כל השורות מטבלת הציונים עבור קורס M-100, M-200 ו- B-10.

```
1. DELETE  GRADES
2.         WHERE  COURSE_ID IN ('M-100', 'M-200', 'B-10')
```

בדוגמה זו השתמשנו בבדיקה לקיום ערך בתוך קבוצת ערכים. רק שורות המקיימות את התנאי תבוטלנה.

טיפול בערכים חסרים (Null Values)

המושג **ערך חסר** (Null Value) בבסיסי נתונים טבלאיים ובשפת SQL דורש הבהרה והתייחסות מיוחדת. מודל הנתונים הטבלאי תומך בערך Null, המצביע על כך שערך מסוים חסר, או לא ידוע. היכולת להבחין בערך מיוחד זה מאפשר הכנסת שורות לטבלה גם אם לא כל הערכים שלה ידועים. קיימת אבחנה ברורה בין הערך Null המציין את העובדה שהערך חסר או לא ידוע, לבין הערך Space המציין שהערך ידוע וערכו הוא התו המציין רווח, או לבין הערך אפס שהוא ערך לכל דבר.

בזמן הצגת השורות לא ניתן להבחין בין ערך חסר לבין רווח, למרות שמבחינת מערכת RDBMS אלה הם שני דברים (ערכים) שונים המיוצגים בבסיס הנתונים על ידי רצף שונה של סיביות. הדרך הנכונה להתייחס אל Null היא כאל מציין המצביע על העובדה שהערך חסר ולא כאל ערך כלשהו, כלומר זהו מחוון מיוחד עבור מערכת RDBMS. מבחינת מודל הנתונים הטבלאי ומערכות ניהול בסיסי נתונים, הצורך לטפל בדרך שונה בערכים חסרים נובע מכך שהקשרים בין הטבלאות מבוססים על הערכים המופיעים בעמודות ומהעובדה שהפקודות השונות של SQL מתייחסות לערכי העמודות בהשוואות שונות.

להדגמת המשמעות של ערכים חסרים, נניח שבטבלת הסטודנטים העמודה עיר מגורים לא חייבת להכיל ערך, וייתכן מצב שבו עיר המגורים של סטודנט אינה ידועה.

הטבלה שבתרשים מציגה מצב שהכתובת של סטודנט שמספרו 210 אינה ידועה, ולכן העמודה עיר מגורים מכילה את הערך Null. לעומת זאת, עבור הסטודנט שמספרו 245 הכתובת ידועה והיא מכילה תו רווח, שיתכן כמובן שהוזן בטעות, אולם מבחינת מערכת RDBMS זהו ערך חוקי.

STUDENT_ID	NAME	CITY	
105	Moshe	Haifa	
210	Dan		← Null
107	Eyal	Tel Aviv	
110	Ran	Haifa	
245	Yoel		← Space
240	Ayelet	Tel Aviv	
200	David	Tel Aviv	
310	Tova	Jerusalem	

תרשים 8.49: טבלת סטודנטים עם ערכים חסרים ועם ערך רווח.

אם נפעיל על טבלה זו שאילתה המציגה את כל השורות בטבלה, נקבל:

```
1. SELECT *
2. FROM    STUDENTS
```

STUDENT_ID	NAME	CITY
105	Moshe	Haifa
210	Dan	
107	Eyal	Tel Aviv
110	Ran	Haifa
245	Yoel	
240	Ayelet	Tel Aviv
200	David	Tel Aviv
310	Tova	Jerusalem

נשים לב לכך שלא ניתן להבחין בעת הצגת הטבלה התוצאתית בין ערך חסר לבין רווח, למרות שמבחינת המערכת הם מיוצגים באופן שונה. כדי לאפשר אבחנה בין ערכים חסרים לבין ערכים אחרים, מאפשרת SQL התייחסות ישירה אל ערכים חסרים על ידי שימוש במונח NULL. אם נרצה לשלוף את כל השורות למעט השורות בהן מופיע ערך חסר, נוכל לרשום:

```
1. SELECT *
2. FROM    STUDENTS
3. WHERE CITY NOT = NULL
```

STUDENT_ID	NAME	CITY
105	Moshe	Haifa
107	Eyal	Tel Aviv
110	Ran	Haifa
245	Yoel	
240	Ayelet	Tel Aviv
200	David	Tel Aviv
310	Tova	Jerusalem

בסיום, השורה של הסטודנט שמספרו 210 נעלמה מטבלת התוצאה.

לוגיקה תלת-ערכית (Three Valued Logic)

כאשר מטפלים בערכים חסרים או לא ידועים עלינו להתייחס לשני כללים חשובים:

- התוצאה של פעולה אריתמטית עם ערך חסר תמיד תהיה ערך חסר. משמעות הדבר היא, שאם העמודה Column1 מכילה ערך חסר, תוצאת הפעולות Column1 x 3, Column1 - 10, או Column1 + 5 תהיה בכל המקרים Null.
- השוואת עמודה המכילה ערך חסר עם עמודה אחרת, תוך שימוש באחד מהאופרטורים הרגילים להשוואה, כגון =, =NOT, >, < ואחרים, תיתן תוצאה "בלתי ידוע" (Unknown), ומשמעות הדבר שתוצאת ההשוואה אינה ידועה. כלומר, אם העמודה Column1 מכילה ערך חסר והעמודה Column2 מכילה ערך חוקי כלשהו,

התוצאה של ביטויים כגון Column1 = Column2, Column1 > Column2 או Column1 NOT= Column2 תהיה בכל המקרים לא ידועה (Unknown).

נשים לב שהכלל השני מרחיב את כללי הלוגיקה הבוליאנית בהם עסקנו עד כה ושבהם התוצאה של השוואה יכולה להיות רק אחת משתי האפשרויות: אמת (True) או שקר (False). מסיבה זו מקובל לקרוא ללוגיקה הבוליאנית הרגילה בשם לוגיקה דו-ערכית (Two Valued Logic). נניח ש-A ו-B הם ביטויים לוגיים כלשהם, ואז:

A = (GRADE_SEM BETWEEN 60 AND 70)
B = (COURSE_ID IN ('C-55', 'M-100'))

ביטוי A בודק אם ציון הסמסטר הוא בין 60 ל-70 והביטוי B בודק אם מספר הקורס הוא אחד משני ערכים. כמובן שהתוצאה של כל אחד מהביטויים האלה יכולה להיות אמת או שקר. הטבלה הבאה מדגימה את טבלת האמת של השוואה בין ביטויים בלוגיקה בוליאנית:

A	B	A AND B	A OR B	NOT A
True	True	True	True	False
True	False	False	True	False
False	True	False	False	True
False	False	False	True	True

תרשים 8.50: טבלת אמת רגילה.

עכשיו נראה את טבלת האמת עבור המצב שבו אחד הביטויים מכיל עמודה עם ערך Null ואז התוצאה שלה לא ידועה, כלומר Unknown. טבלת האמת החדשה תהיה כמפורט בתרשים 8.51.

כפי שניתן לראות, בעוד שטבלת האמת הרגילה מכילה רק ארבע אפשרויות הרי שהוספת אפשרות של תוצאה נוספת, Unknown, מוסיפה חמש אפשרויות חדשות. מקובל לקרוא ללוגיקה המטפלת בשלושה מצבים שונים בשם לוגיקה תלת-ערכית (Three Valued Logic).

A	B	A AND B	A OR B	NOT A
True	True	True	True	False
True	False	False	True	False
False	True	False	False	True
False	False	False	True	True
True	Unknown	Unknown	True	False
Unknown	True	Unknown	True	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown
Unknown	False	False	Unknown	Unknown
True	Unknown	False	Unknown	True

תרשים 8.51: טבלת אמת בלוגיקה תלת-ערכית.

נתבונן בשאילתה זו:

```
1. SELECT *
2. FROM    STUDENTS
3. WHERE (CITY LIKE 'A%') OR (CITY NOT LIKE 'A%')
```

במצב של לוגיקה בוליאנית, שאילתה זו אמורה להחזיר כתוצאה את כל השורות, מאחר וכל שם עיר מתחיל באות A או שאינו מתחיל באות A. אם באחת או יותר מהשורות העמודה שם עיר תכיל את הערך Null, התוצאה של שני הביטויים הלוגיים תהיה Unknown והשורה לא תישלף, מאחר ועל פי טבלת האמת הנ"ל התוצאה תהיה Unknown וכפי שכבר הסברנו, לטבלה התוצאתית נשלפות רק שורות המחזירות את התוצאה True.

טיפול בערכים חסרים על ידי פקודות SQL

שפת SQL מכילה התייחסות וצורת טיפול מיוחדת בערכים חסרים. נסקור את הפקודות השונות של השפה וכיצד הן מתייחסות לסוגיה של ערכים חסרים.

הוספת שורות עם ערכים חסרים

הפקודה INSERT מכילה אפשרות מיוחדת להכנסת ערך Null לתוך שורה חדשה.

```
1 . INSERT INTO GRADES
2 .   VALUES   ('210' ,'Medicine', Null)
```

פקודה זו מוסיפה שורה חדשה בטבלת הציונים עבור סטודנט שמספרו 210 ומספר קורס C-210. העמודה ציון הסמסטר תקבל ערך Null. אגב, עמודה שאינה מוזכרת במפורש בפקודה INSERT מקבלת כברירת מחדל את הערך Null, אלא אם נקבע לעמודה ערך ברירת מחדל כלשהו (הדרך לקבוע ברירות מחדל לעמודות תוצג בפרק הבא). לכן עמודת הציון הסמסטריאלי תקבל ערך Null, גם אם נרשום:

```
1. INSERT INTO GRADES
2. VALUES  ( '210' ,'C-200' ,'AUT1999' ,'A' ,85)
```

לצורך הדגמת שאר הפקודות וכיצד הן מטפלות בערכים חסרים, נשתמש בטבלה שבתרשים 8.52, שבה נוכל לראות שלוש שורות המכילות ערך חסר בעמודה ציון סמסטר.

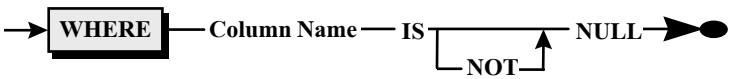
STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE	GRADE_SEM
מס. סטודנט	מס. קורס	סמסטר	מועד בחינה	ציון סופי	ציון סמסטר
105	C-55	SUM1998	A	58	70
210	M-100	AUT1999	A	90	90
105	M-100	SUM1998	B	75	50
105	C-200	AUT1999	A	90	85
210	C-200	AUT1999	A	85	
210	B-10	WIN1999	A	78	50
105	B-40	WIN1999	B	70	70
245	M-100	AUT1999	A	90	80
245	B-10	AUT1999	A	80	70
200	C-200	AUT1999	B	78	
200	B-10	AUT1999	A	70	65
245	B-40	WIN1998	A	85	95
200	M-100	SUM1998	B	90	
310	M-100	SUM1998	A	65	100

תרשים 8.52: טבלת ציונים עם ערכים חסרים.

בדיקת ערכים חסרים (Null Value Test)

לשליפת שורות שבהן ערך מסוים של עמודה מכיל את הערך Null, ניתן להשתמש באופרטור IS NULL או בשלילה שלו IS NOT NULL.

תחביר הפקודה:



תרשים 8.53: תרשים תחביר לבדיקת ערכים חסרים.

דוגמה : הצג את מספר הסטודנט של כל הסטודנטים שאין להם ציון סמסטר. יש להציג כל סטודנט פעם אחת בלבד, גם אם אין לו ציון סמסטר במספר קורסים.

```
1. SELECT   DISTINCT STUDENT_ID
2. FROM     GRADES
3. WHERE    GRADE_SEM  IS  NULL
```

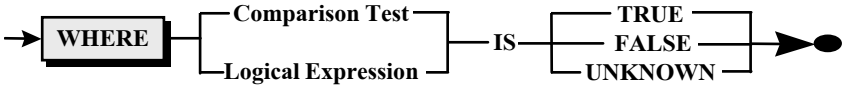
STUDENT_ID
210
200

הסטודנט שמספרו 200 הוצג רק פעם אחת, למרות שחסר לו ציון בשני קורסים, וזאת בגלל השימוש באפשרות DISTINCT המופיעה בשורה ראשונה.

בדיקת ביטוי לוגי

שפת SQL מאפשרת בדיקת תוצאה של ביטוי לוגי. בגלל התמיכה של השפה בלוגיקה תלת-ערכית, ניתן לבדוק שהתוצאה היא אחת משלוש האפשרויות: אמת, שקר ולא ידוע.

תחביר הפקודה:



תרשים 8.54: תרשים תחביר לבדיקת ביטוי לוגי.

דוגמה א' : הצג את כל השורות בהן המכפלה בין ציון סמסטר לבין ציון סופי אינה ידועה.

```
1. SELECT   *
2. FROM     GRADES
3. WHERE    (GRADE_SEM  * GRADE) IS UNKNOWN
```

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE	GRADE_SEM
210	C-200	AUT1999	A	85	
200	C-200	AUT1999	B	78	
200	M-100	SUM1998	B	90	

דוגמה ב' : הצג את השורות בהן מועד הבחינה הוא ב' והמכפלה בין ציון הסמסטר לציון הסופי אינה ידועה.

```
1. SELECT   *
2. FROM     GRADES
3. WHERE    (TERM = 'B') IS TRUE AND
4.          (GRADE_SEM  * GRADE) IS UNKNOWN
```

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE	GRADE_SEM
200	C-200	AUT1999	B	78	
200	M-100	SUM1998	B	90	

בדיקת טווח (Between Test)

שפת SQL מאפשרת לבדוק אם ערך כלשהו נמצא בתוך טווח ערכים רציף. אם הערך הנבדק הוא "ערך חסר", התוצאה המוחזרת אינה ידועה, ולכן שורות אלו לא תישלפנה.

דוגמה א' : יש לשלוף את כל השורות בהן ציון הסמסטר הוא בין 60 ל-70 ראה שרטוט בעמוד הבא).

נשים לב ששורות המכילות ערך חסר לא נשלפו.

```
SELECT  *
FROM    GRADES
WHERE   GRADE_SEM  BETWEEN 60 AND 70
```

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE	GRADE_SEM
105	C-55	SUM1998	A	58	70
105	B-40	WIN1999	B	70	70
245	B-10	AUT1999	A	80	70
200	B-10	AUT1999	A	70	65

דוגמה ב' : יש לשלוף שורות בהן הציון הסופי הוא בין 0.5 ל- 0.75 מציון הסמסטר.

```
1.  SELECT  *
2.  FROM    GRADES
3.  WHERE  GRADE  BETWEEN
5.             (0.5 * GRADE_SEM)  AND (0.7 * GRADE_SEM)
```

כל השורות בהן הביטוי הקובע את החסם התחתון או החסם העליון נותן תוצאה Unknown, לא תישלפנה.

צירוף טבלאות (Join)

בצירוף רגיל (Equi Join) של טבלאות שבו משווים את הערכים של שתי עמודות, שורות המכילות ערך Null בעמודה אחת או בשתי העמודות לא תישלפנה. נשים לב שגם אם שתי העמודות מכילות Null ולכאורה הביטוי Null = Null אמור להשתתף בצירוף, הרי שעל פי טבלת האמת התוצאה תהיה Unknown ולכן השורה לא תישלף. בצירוף מסוג Outer Join תישלפנה גם שורות המכילות ערך חסר.

פונקציות מובנות (Built In Functions)

בעיה מיוחדת שימוש בפונקציות מובנות מתרחשת כשבחלק מהשורות של הטבלה יש ערכים חסרים. תקן SQL2 קובע שעל הפונקציות המובנות להתעלם משורות המכילות ערך חסר.

דוגמה א' : נראה כיצד הפונקציה COUNT מתייחסת לערכים חסרים. הדוגמה הבאה מציגה את ההבדל בין התוצאה של ספירת השורות כתוצאה מהתעלמות הפונקציה משורות עם ערכים חסרים.

```
1.  SELECT COUNT (*),
2.             COUNT (GRADE_SEM)
3.  FROM    GRADES
```

COL 1	COL 2
14	11

ההבדל בין מספר השורות לבין מספר השורות בעמודה של הציונים נובע כמובן מהעובדה שבעמודת ציון הסמסטר יש שלוש שורות עם ערכים חסרים. המשמעות היא שהפונקציה COUNT מתעלמת משורות עם ערכים חסרים.

דוגמה ב' : נראה שימוש בפונקציה SUM ובצורת הטיפול שלה בערכים חסרים.

```
1. SELECT SUM (GRADE),
2.        SUM (GRADE_SEM),
3.        SUM (GRADE - GRADE_SEM)
4. FROM   GRADES
```

COL1	COL2	COL3
1104	825	26

נוכל לראות שהעמודה השלישית אינה נותנת את התוצאה הצפויה 279, כלומר את ההפרש בין שתי העמודות הראשונות, אלא תוצאה אחרת. הסיבה לכך נובעת מהעובדה שהפונקציה SUM מתעלמת מהשורות בהן אחד הערכים המשתתפים בחישוב הוא ערך חסר. באופן דומה, הפונקציה AVG לחישוב ממוצע תתעלם משורות בהן הערך חסר.

שאילתות עם הקבצות (Grouped Queries)

בשאילתה עם הקבצה על עמודה המכילה ערך חסר נוצר למעשה מצב מיוחד. המשמעות של Null היא ערך חסר, או לא ידוע, ולכן כל שורה המכילה ערך כזה צריכה להיות שייכת להקבצה משל עצמה, כי אם נקבץ יחד את השורות המכילות ערך חסר, המשמעות היא שכל הערכים שלהם שווים. כפי שראינו בטבלת האמת, הביטוי Null = Null נותן את הערך Unknown ולא True, ולכן מבחינה לוגית אסור לקבץ שורות אלו יחד. למרות האמור לעיל, תקן SQL2 בכל זאת קובע שכל השורות המכילות Null תקובצנה יחדיו, כדי לפשט את היישום.

דוגמה : הצג לכל ציון סמסטר את מונה מספר הסטודנטים שקיבלו את אותו ציון. את הרשימה יש להציג לפי סדר יורד של המונה ובתוכו לפי סדר יורד של הציון.

```
1. SELECT   GRADE_SEM,
2.          COUNT (*) AS GRADE_COUNT
3. FROM     GRADES
4. GROUP BY GRADE_SEM
5. ORDER BY GRADE_COUNT DESC,
6.          GRADE_SEM DESC
```

GRADE_SEM	GRADE_COUNT
70	3
50	2
100	1
95	1
90	1
85	1
80	1
65	1
Null	3

שים לב שהשורה האחרונה מונה את מספר הסטודנטים שציון הסמסטר שלהם חסר.

סיכום

בפרק זה סקרנו את פקודות SQL לטיפול בנתונים. שפת SQL מורכבת מארבע פקודות בסיסיות בלבד לטיפול בנתונים, אך ניתן לראות מתוך הסקירה שמספר האפשרויות שלה גדול מאוד והשפה רבת עוצמה. פקודת SQL אחת הכתובה במספר שורות בודדות יכולה להחליף תוכנית יישום ארוכה ומורכבת שהיתה מחייבת תוכניתן מקצועי למאמץ ניכר. הפקודה בעלת העוצמה הגדולה ביותר היא SELECT, אשר מגלמת כמעט את כל תפישת המודל הטבלאי. יחד עם פקודה זו, מכילה SQL את כל שאר הפקודות הדרושות לתחזוקת הנתונים בבסיס הנתונים, כמו הוספה, עדכון וביטול.

לשפה גמישות רבה וכוללת דרכים שונות לבטא את אותה שאילתה. ניתן למשל לכתוב את תנאי הצירוף במשפט WHERE, ניתן לכתוב אותו במשפט FROM במספר דרכים שונות (USING, ON או NATURAL JOIN אם לעמודות הצירוף שמות זהים), ניתן להשתמש בתנאי מורכב או לנסח את אותו התנאי בתת-שאילתה וכד'. לגמישות זו יתרונות וחסרונות. היתרון העיקרי הוא ביכולת להתאים את צורת כתיבת השאילתה לסגנון הנוח ביותר לכל אחד. עם זאת, היכולת לבטא את אותה בקשה במספר רב של דרכים יכולה ליצור בלבול. עקרונית, מעבד SQL צריך לפתח את תוכנית הגישה לנתונים גם אם צורת כתיבת השאילתה היא שונה. במציאות, המצב אינו תמיד כך. שאילתה המנוסחת בצורה של תנאי במשפט WHERE ושאילתה המשתמשת בתת-שאילתה, מייצרות תוכניות גישה שונות העלולות לספק זמני תגובה שונים מאוד. לא פעם קורה שניסוח שונה של שאילתה מביא לשיפור או הרעה משמעותיים בביצועי השאילתה.

חלק גדול מעצמת שפת SQL נמצא בתוך מעבד השפה וברעיון שרכיב זה מגלם. במקום שמפתח התוכנה יכיר את המבנה המדויק של בסיס הנתונים והקשרים בין הנתונים, בסביבה של מערכת RDBMS הוא מנסח את בקשתו בצורה פשוטה יחסית ומעבד SQL מתרגם את בקשת המשתמש לתוכנית גישה לבסיס הנתונים, תוכנית המבוססת על מבנה בסיס הנתונים, קיום אינדקסים על הטבלאות, הקשרים בין הטבלאות וכן סטטיסטיקות המנוהלות על ידי המערכת. תוכנית זו מבצעת בדרך כלל את הגישות היעילות ביותר לטיפול בנתונים.

שאלות חזרה ותרגילים

שאלות חזרה

1. הסבר מהו המונח "תואם איחוד" ומתי הוא רלוונטי.
2. מתי יש להשתמש במשפט Having בפקודת SQL?
3. מהו ההבדל בין Group By לבין Order By בפקודת SQL?
4. מתי יש צורך להשתמש בשמות נרדפים (Aliases) בפקודות SQL?
5. הסבר באיזה מצבים חובה להשתמש בתת-שאילתה ובאיזה מצבים ניתן להשתמש ב-Join ולא בתת-שאילתה.
6. מדוע לדעתך מרשה SQL הופעת שורות כפולות בטבלת התוצאה, למרות שזה נוגד את כללי האלגברה הטבלאית ואת ההגדרה המתמטית של יחס?
7. מדוע יש חשיבות מיוחדת לצורת הטיפול בערכים חסרים (Null) במודל הטבלאי?
8. לפניך המשפט SELECT A B. כיצד ניתן לקבוע אם A ו-B מייצגים עמודות, או ש-B הוא שם נרדף ל-A?

תרגילים

1. נתונה הסכימה הטבלאית הבאה:

- **מלון** (קוד מלון, שם מלון, עיר, כתובת, דרגת המלון, מספר טלפון, מספר חדרים)
- **חדר במלון** (קוד מלון, מספר חדר, סוג חדר, מספר מיטות, מחיר ללילה בדולרים)
- **אורח במלון** (מספר זהות או דרכון, שם אורח, עיר, כתובת, ארץ)
- **הזמנת חדר** (קוד מלון, מספר חדר, מתאריך, עד תאריך, צורת תשלום)

כתוב את השאילתות הבאות בשפת SQL:

א. הצג רשימת כל המלונות בעיר טבריה ומיין אותה לפי שם המלון.

ב. הצג את רשימת החדרים מסוג חדר ענק שהמחיר ללילה הוא בין 150 ל- 200 דולר ללילה. לכל חדר יש להציג את שם המלון, העיר בה ממוקם המלון, מספר החדר במלון והמחיר ללילה. יש למיין את רשימת החדרים מהמחיר היקר ביותר עד הנמוך ביותר.

ג. הצג את רשימת כל החדרים עם שתי מיטות לפחות במלונות שיש בהם מעל 100 חדרים. מיין את הרשימה לפי שם המלון ומספר החדר.

ד. הצג את כל ההזמנות לחדרים מסוג מסוים שצורת התשלום עבורם בחודשיים האחרונים היתה במזומן.

ה. הצג את רשימת שמות האורחים המתארחים כרגע במלון המלך דוד או הילטון ירושלים. מיין את הרשימה לפי עיר המגורים של האורחים.

ו. מהו המחיר הממוצע של חדר במלונות בדרגת חמישה כוכבים.

ז. מהו מספר הימים הממוצע ששוהה אורח במלון בדרגת ארבעה כוכבים בעיר אילת.

ח. כמה מלונות בעלי ארבעה או חמישה כוכבים יש באילת או בחיפה ושיש בהם בין 140 ל- 200 חדרים. מיין את הרשימה בסדר יורד של מספר המלונות.

ט. הצג את רשימת כל החדרים במלון דן פנורמה בעיר תל אביב ושנכון לתאריך נתון אין להם הזמנה.

י. הצג את מספר החדרים בכל אחד מהמלונות בעלי שלושה כוכבים בחיפה. הצג את הרשימה לפי סדר יורד של מחיר החדר.

יא. מהו החדר בעל מספר ההזמנות הגדול ביותר בכל אחד מהמלונות.

יב. מהו המספר הממוצע של הזמנות חדרים בחודש יולי 1999 בכל אחד מהמלונות בעלי ארבעה או חמישה כוכבים.

יג. הצג את רשימת כל האורחים הגרים בירושלים והנמצאים בתאריך נתון במלון הנסיכה באילת בחדרים שהמחיר שלהם הוא גדול מ- 200 דולר ללילה. מיין את הרשימה לפי מחיר החדר ושם האורח.

2. השאילתות הבאות מתייחסות לבסיס הנתונים של חנות המוסיקה המתואר בשאלה 5 שבפרק 4.

א. הצג לכל מנוי את רשימת כל האלבומים שהלקוח רכש מתחילת המנוי, ואת סוג האלבום.

ב. הצג את רשימת כל השירים באלבומים שהופקו על ידי חברת EMI.

ג. הצג את רשימת שמות המנויים וכתובתם לכל הלקוחות שרכשו אלבום המכיל שיר שנכתב על ידי Paul MaCartney.

3. נתונה הסכימה הטבלאית הבאה:

- **לקוחות** (קוד לקוח, שם לקוח, עיר)
- **פריטים** (קוד פריט, שם פריט, סוג פריט, מחיר פריט)
- **הזמנות** (קוד לקוח, קוד פריט, תאריך הזמנה, כמות מוזמנת)

כתוב את השאילתות הבאות בשפת SQL:

א. הצג רשימת הלקוחות שהזמינו פריטים מסוג CD Players. לכל לקוח יש להציג את שם הלקוח, שם הפריט וערך ההזמנה. ערך ההזמנה מחושב על ידי המכפלה של מחיר הפריט בכמות המוזמנת. את הרשימה יש למיין לפי שם הלקוח ובתוכו לפי ערך ההזמנה בסדר יורד.

ב. הצג את כל הצמדים של לקוחות הגרים באותה העיר.

ג. הצג את מספר ההזמנות לכל לקוח בהן מופיע פריט 1242. יש להציג רק לקוחות שהזמינו יותר מ 10 הזמנות של הפריט 1242.

ד. הצג את שמות כל הלקוחות ועיר המגורים שלהם שהזמינו פריט כלשהו שבשם הפריט מופיע רצף התווים 'elec' וסוג הפריט הוא Video או CD Player.

ה. נניח שעיר המגורים של הלקוח יכולה להכיל את הערך Null. כתוב שאילתה המציגה את כל ההזמנות ללקוחות שעיר המגורים שלהם לא ידועה.

ו. הצג את רשימת הלקוחות שהזמינו לפחות פריט אחד שערך ההזמנה גבוה מהערך הממוצע של ההזמנות שהוזמנו על ידי לקוח ששמו Eyal. ערך הזמנה מחושב על ידי המכפלה של מחיר הפריט בכמות המוזמנת.

פרק 9

הגדרת בסיס הנתונים (Data Definition)

1. מבוא
2. אמינות ושלמות בסיס הנתונים (Data Integrity)
3. שמות וכינויים (SQL Identifiers)
4. טיפוסי נתונים (Data Types)
5. הגדרת בסיס נתונים חדש
6. הגדרת מרחב ערכים (Domain Definition)
7. הגדרת טבלה (Table Create)
8. ביטול טבלה מבסיס הנתונים (Drop Table)
9. שינוי טבלה (Alter Table)
10. אילוצים ברמת בסיס הנתונים (Assertions)
11. אינדקסים (Indexes)
12. סיכום
13. שאלות חזרה ותרגילים

מבוא

עד כה הנחנו שבסיס הנתונים מוגדר וקיים. בפרק זה נסקור את פקודות SQL הקשורות להגדרת בסיס הנתונים. פקודות אלו נקראות באופן כללי **שפה להגדרת נתונים** – **DDL** (Data Definition Language). הן משמשות בדרך כלל את מנהל בסיס הנתונים להקמת וביטול אובייקטים שונים של בסיס הנתונים – טבלאות, אינדקסים וכד' – ומאפשרות הגדרת טבלאות חדשות, אילוצים שונים שעל הטבלאות לקיים, וקשרים בין טבלאות שונות. כמו כן הן מאפשרות שינוי מבנה טבלאות קיימות, ביטול טבלאות, הוספת אינדקסים, ביטולם וכד'.

נוסף להגדרת הטבלאות השונות, אפשר להגדיר מיגוון כללים שמטרתם לאפשר למערכת RDBMS להבטיח שבסיס הנתונים יהיה אמין ושלם. הגדרת אילוצי אמינות אלה ברמת בסיס הנתונים, ולא ברמת תוכניות היישום, הינה בעלת חשיבות מרובה. הגדרה חד פעמית תוך אבטחת האכיפה על ידי מערכת ניהול בסיסי הנתונים הינה דרך נוחה ויעילה וחשוב יותר – דרך המבטיחה את קיום האילוצים בכל עת. בפרק זה נרחיב את הדיון בסוגי האילוצים השונים וכיצד ניתן לבטא אותם בשפה להגדרת הנתונים – DDL.

המערכות המסחריות מאפשרות לבצע את הפקודות הסטנדרטיות להגדרת בסיס הנתונים ובנוסף הן מאפשרות לבצע מיגוון פעולות, כגון הקצאת שטחי דיסק והגדרת שיטות שונות לאחסון הטבלאות. פקודות אלו אינן חלק מתקן SQL2, אלא ייחודיות לכל מערכת מסחרית, לא נסקור אותן כאן. בפרק זה נעסוק בפקודות התקניות בלבד.

בפרק זה

- נציג את פקודות SQL להגדרת בסיס הנתונים והנתונים המאוחסנים בו.
- נסביר כיצד להגדיר כללים ואילוצים שיבטיחו את אמינות ושלמות בסיס הנתונים.
- נציג דרכי אכיפה מובנים להבטחת שלמות ואמינות בסיס הנתונים.
- נסביר את האפשרויות והאילוצים בעת הגדרת מרחב ערכים, ביטול ושינוי של טבלה.
- נציג מהם אינדקסים, ונסביר את תרומתם ותועלתם לניהול והפעלת בסיס הנתונים.

אמינות ושלמות בסיס הנתונים (Data Integrity)

כללי אמינות ושלמות

לפני שנציג את פקודות SQL המשמשות להגדרת בסיס הנתונים, נרחיב את הדיון בנושא אמינות ושלמות בסיס הנתונים. לאחר מכן, במהלך הצגת הפקודות השונות להגדרת נתונים, נראה כיצד נושא זה בא לידי ביטוי בשפת SQL.

כפי שראינו, בסיס נתונים טבלאי מורכב מ**אוסף טבלאות** נפרדות **הקשורות** ביניהן ב**קשרים לוגיים** שונים.

מצב בסיס הנתונים **Database State**	**מצב *בסיס הנתונים* יוגדר כאוסף כל הערכים בכל הטבלאות בנקודת זמן מסוימת.**

מכיון שבסיס הנתונים מייצג מציאות מסוימת, מצב בסיס הנתונים צריך להיות עקבי, אמין ושלם. משמעות הדבר היא שהערכים בטבלאות השונות צריכות לייצג את המציאות ברמת האמינות הגבוהה ביותר. נציג מספר דוגמאות להמחשת הנושא:

- אם נקבע שלכל קורס יש מספר קורס חד-ערכי, לא נרצה למצוא בטבלת קורסים שני קורסים שונים שמספרם M-100.
- אם במכללה קיים כלל שקובע שלכל מחלקה יש רק ראש מחלקה אחד וראש מחלקה לא יכול לשמש כראש מחלקה בו-זמנית במספר מחלקות, לא נרצה למצוא בבסיס הנתונים מצב שד״ר ישראל עומד בראש שתי מחלקות.
- אם נקבע שקורס יוכל להיות מסוג שיעור, מעבדה או סמינר, לא נרצה למצוא בבסיס הנתונים קורסים מסוג אחר.
- מאחר וכל ציון מתייחס למספר סטודנט ומספר קורס מסוים, לא נרצה למצוא מצב שבו הציון שמופיע בטבלת ציונים שייך לסטודנט שלא קיים בטבלת סטודנטים או לקורס שלא קיים בטבלת קורסים.
- מאחר וכל ציון יכול להיות מספר בין 0 ל-100, לא נרצה למצוא בעמודה זו מספר כגון 120 או מספר שלילי כמו -55 או מחרוזת תווים כגון AA.
- אם נקבע שסטודנט לא יכול להבחן יותר מאשר פעמיים באותו קורס, לא נרצה מצב שבו טבלת ציונים מכילה שלושה ציונים שונים של סטודנט בקורס מסוים.

כפי שניתן לראות מאוסף דוגמאות אלו, בסיס נתונים אמין ושלם הוא בסיס נתונים שמכיל אוסף של ערכים חוקיים. הערכים בטבלאות והקשרים בין הטבלאות צריכים לקיים אוסף של אילוצים וכללים שמקובל לקרוא להם בשם **כללי אמינות ושלמות הנתונים** (Data Integrity Rules). מונח זה מתייחס לאוסף כל החוקים והאילוצים המגדירים את המצב הנכון והתקין של בסיס הנתונים.

כללי אמינות ושלמות נתונים **Data Integrity Rules**	**כללי אמינות ושלמות הנתונים הם אוסף של כללים וחוקים המגדירים את האילוצים שבסיס הנתונים חייב לקיים בכל נקודת זמן.**

עקרונית, ניתן להגדיר ולאכוף את כללי האמינות והשלמות בתוך תוכניות היישום השונות, כלומר לקבוע שזו אחריות היישום לשמור על הכללים ולא להפר אותם. קל לומר, קשה לבצע. משמעות הדבר היא שכל תוכנית יישום שניגשת לטבלה כלשהי ומעדכנת אותה – מוסיפה שורות חדשות, מעדכנת שורות קיימות, מבטלת שורות – חייבת להכיר את כללי האמינות והשלמות, שלעיתים יכולים להיות מורכבים למדי, ולוודא שהיא אינה מפרה אותם. בשיטה זו, השמירה על האמינות והשלמות של בסיס הנתונים נתונה בידי מפתחי היישום, וכל טעות או תקלה מקרית יכולים לפגוע במצב בסיס הנתונים.

התאוששות ממצב שבו בסיס נתונים לא אמין הינה קשה, ולעיתים אפילו בלתי אפשרית. היא עלולה לגרום במקרים קיצוניים לחוסר אמון של המשתמשים במידע המופק מהיישום. גישה נכונה יותר לשמירה על אמינות ושלמות בסיס הנתונים, היא להגדיר את הכללים והאילוצים בעת הגדרת בסיס הנתונים ולהשאיר למערכת RDBMS את מלאכת האכיפה. מאחר ומערכת זו שולטת בכל עת על כל הגישות לבסיס הנתונים, ההגדרות נעשות פעם אחת בלבד, אין צורך לחזור עליהן ביישומים שונים והאחריות על מצב בסיס הנתונים עוברת מהיישום אל המערכת המנהלת את הנתונים. נדגיש שלא כל כללי האמינות והשלמות שנדרשים על ידי היישומים נתמכים באופן ישיר על ידי שפת SQL, ולכן במציאות נמצא שחלק מהאילוצים מוגדר בתוך בסיס הנתונים וחלק אחר מוגדר ביישום עצמו.

קטגוריות של כללי אמינות ושלמות בסביבת SQL

מקובל לחלק את כללי האמינות והשלמות הנתמכים על ידי שפת SQL למספר קטגוריות עיקריות.

נתוני חובה (Required Data)

קביעת העמודות שחייבות להכיל בכל עת ערך כלשהו. עמודות אלו לא יכולות לקבל את הערך Null, המציין שהערך חסר. לדוגמה ניתן להגדיר שציון הסמסטר חייב להופיע בכל שורה בטבלת ציונים או שמספר הקורס בטבלת קורסים חייב להופיע, מאחר והוא המפתח העיקרי של הטבלה והוא מזהה באופן חד-ערכי כל שורה.

אילוצי מרחב ערכים (Domain Integrity)

לכל עמודה יש טיפוס נתונים משלה: נומרי שלם, נומרי עשרוני, תווים, תאריך וכד'. עצם הגדרת טיפוס הנתונים של העמודה מהווה הצהרה על הערכים שהעמודה יכולה לקבל ובמובן זה ניתן להתייחס אל ההגדרה כאל אילוץ. לדוגמה עמודה מטיפוס נומרי שלם יכולה לקבל את כל טווח המספרים הנתמך על ידי המערכת, ממספר שלם שלילי נמוך מאוד ועד מספר שלם חיובי גדול מאוד. לדוגמה, ב-DB2 עמודה המוגדרת כטיפוס INTEGER תופסת ארבעה בתים ויכולה לקבל את כל המספרים השלמים אותם ניתן לייצג על ידי 31 סיביות (סיבית נוספת מיועדת עבור הסימן). לעומתה, עמודה המוגדרת כטיפוס SMALLINT תופסת רק שני בתים ולכן יכולה לקבל מספרים שלמים אותם ניתן לייצג על ידי 15 סיביות. עמודה המוגדרת כ-CHAR(3) יכולה אמנם להכיל תווים, מספרים וסימנים מיוחדים, אבל רק שלושה תווים כאלה.

קביעת טיפוס הנתונים של עמודה קובע גם איזה ערכים העמודה אינה יכולה לקבל. למשל, עמודה המוגדרת כ-INTEGER אינה יכולה לקבל ערך AB או ערך 125.32, מאחר ואלה אינם מספרים שלמים. עמודה מטיפוס תאריך לא יכולה לקבל מספר שלילי וכד'.

נוסף לאילוץ הנובע באופן עקיף מעצם הגדרת טיפוס הנתונים של העמודה ואורכה, יש מצבים בהם נרצה לצמצם את מרחב הערכים שעמודה יכולה לקבל. לאילוצים אלה אנו קוראים **אילוצי מרחב ערכים** (Domain Constraint). למשל העמודה ציון היא בעלת טיפוס נתונים נומרי שלם, אבל יכולה להכיל רק את טווח המספרים השלמים שבין 0 ל-100 כי זהו הטווח שציון יכול לקבל. אין כל משמעות לציון -20 או לציון 234, למרות שאלה מספרים שלמים. דוגמה אחרת יכולה להיות העמודה מועד הבחינה שהינה בעלת טיפוס נתונים של מחרוזת תווים באורך של תו אחד. למרות שמחרוזת כזאת יכולה להכיל כל תו או מספר או סימן מיוחד, בסביבה העסקית – ובמקרה זה, במכללה – עמודה זו יכולה להכיל רק את הערכים A, B או C המציינים שהסטודנט נבחן במועד א', ב' או ג'. ערך כגון + או 7, שהם ערכים חוקיים לעמודה מסוג CHAR הם חסרי כל משמעות בהקשר של מועד בחינה. דוגמה נוספת לאילוץ על מרחב ערכים יכולה להיות אילוץ החד-ערכיות. אילוץ זה קובע שכל ערך מתוך מרחב הערכים יכול להופיע פעם אחת בלבד בתוך העמודה. למשל, בעמודה מספר קורס, שהיא מטיפוס CHAR, הערך M-100 יכול להופיע פעם אחת בלבד.

שלמות הטבלה (Table Integrity)

לטבלה יש מפתח עיקרי (Primary Key). העמודות המשתתפות במפתח העיקרי, לא יכולות להכיל Null וחייבות להכיל ערך חד-ערכי (Unique), מאחר והמפתח משמש לזיהוי חד-ערכי של השורות בטבלה.

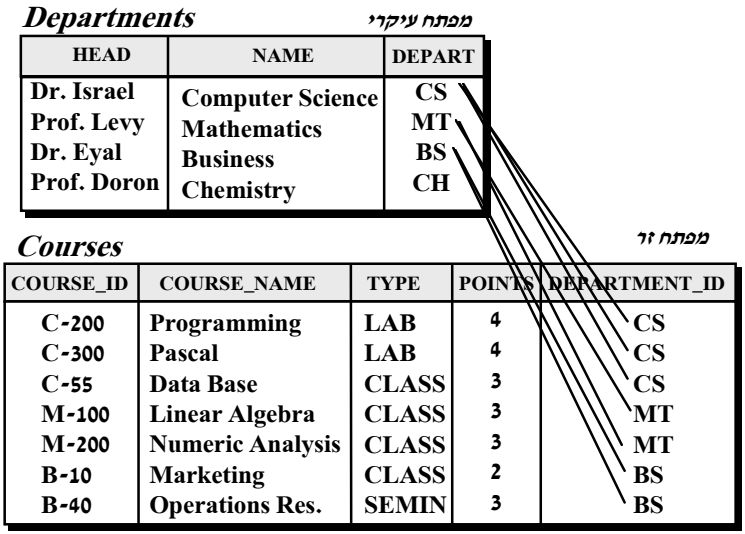
שלמות הקשרים בין טבלאות
(Referential Integrity)

בין הטבלאות של בסיס הנתונים מתקיימים קשרים שונים.

אילוץ שלמות הקשרים **Referential Integrity**	***אילוץ שלמות הקשרים*, הנקרא לעיתים בקיצור RI, מגדיר את צורת ניהול הקשרים בין הטבלאות ומבטיח שמערכת RDBMS תשמור על אמינות ושלמות הקשרים ביניהן בכל נקודת זמן.**

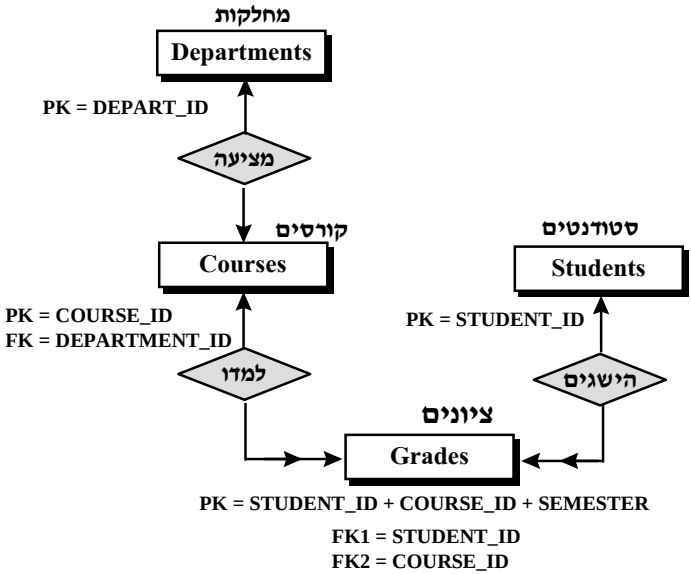
נחזור לדוגמה של טבלת מחלקות וטבלת קורסים כדי להציג את המפתח הזר. למען פשטות הצגת תרשים 9.1, הפכנו את סדר הופעת העמודות בטבלת מחלקות.

העמודה מספר מחלקה בטבלת קורסים היא מפתח זר, והעמודה מספר מחלקה בטבלת מחלקות היא מפתח עיקרי. התרשים מציג קשר אב/בן המתקבל בין שתי הטבלאות. כל ערך המופיע בעמודת המפתח הזר חייב להופיע בעמודת המפתח העיקרי, אבל לא להיפך. למשל, למחלקה עם קוד CH אין אף קורס, ולכן אין אף שורה בטבלת קורסים המכילה ערך זה.



תרשים 9.1: מפתח זר ועיקרי.

תרשים 9.2 מציג את המפתחות העיקריים והזרים בכל אחת מהטבלאות המשתתפות בבסיס הנתונים לדוגמה.



תרשים 9.2: מפתחות עיקריים וזרים בבסיס הנתונים לדוגמה.

האילוץ מתייחס לנקודות זמן שונות: מה תהיה ההתנהגות בעת הוספת שורה חדשה לטבלת הבן, מה תהיה ההתנהגות בעת ביטול שורה מתוך טבלת האב, ומה תהיה ההתנהגות בעת עדכון שורה בטבלת האב ובטבלת הבן.

- **אילוץ בעת הוספה:** אילוץ זה קובע שמפתח זר יכול לקבל ערך מתוך הערכים הקיימים בטבלת האב. מערכת RDBMS תבדוק בעת ההוספה שהערך המופיע במפתח הזר של טבלת הבן, כבר קיים בתוך העמודה של המפתח העיקרי של טבלת האב. אם הערך לא קיים, המערכת לא תרשה לבצע את הוספת השורה החדשה.
- **אילוץ בעת ביטול:** אילוץ זה מגדיר כיצד יש להתנהג אם מבטלים שורה בטבלת האב ומה לעשות עם השורות בטבלת הבן. מאחר וקיים קשר דו-סטרי בין שתי הטבלאות, ביטול של שורה בטבלת האב יכולה להביא לניתוק הקשר ולמעשה להפרת האילוץ. ניתן להגדיר מצבים בהם נרשה את הביטול ומצבים בהם לא נרשה זאת.
- **אילוץ בעת עדכון:** עדכון המפתח הזר בטבלת הבן או המפתח העיקרי בטבלת האב יכולה לגרום לניתוק הקשר. גם כאן יש להגדיר איזה מצבים מותרים ואיזה אסורים.

שלמות הקשרים (RI) יכולה להתייחס לקשרים בין טבלאות שונות וגם לקשרים של טבלה עם עצמה – קשר רפלקסיבי. השיטה שבה מגדירים את RI יכולה להיות על ידי **שלמות קשרים מוצהרת** (Declarative RI) בעת הגדרת בסיס הנתונים, תוך שימוש במנגנון של **פרוצדורות מאוחסנות** (Stored Procedures) בתוך בסיס הנתונים או מנגנון של **מזניק** (Trigger) שניתן להפעילם בכל פקודת עדכון. במקרה זה מקובל לקרוא לשיטה זו בשם **שלמות קשרים פרוצדורלית** (Procedural RI). הפרוצדורות מופעלות אוטומטית בעת פעולות עדכון של בסיס הנתונים.

לדוגמה, בעת ביטול שורה בטבלת המחלקות, מופעלת אוטומטית פרוצדורה המוחקת את כל השורות של אותה מחלקה בטבלת הציונים. תקן SQL2 מבוסס על Declarative RI ולא על שימוש בפרוצדורות.

כללים וחוקים עסקיים (Business Rules)

לעיתים יש צורך להגדיר כללי שלמות ואמינות הנובעים מהיישום עצמו או מכללי הארגון. כללים אלה יכולים להיות מורכבים מאוד ולעיתים משתתפים בכללים אלה מספר טבלאות שונות. שפת SQL מאפשרת את הגדרת האילוצים האלה על ידי חוקים הנקראים **הכרזות** (Assertions), המאוחסנים בקטלוג בסיס הנתונים ומופעלים באופן אוטומטי על ידי מערכת RDBMS כאשר מתבצעים שינויים בטבלאות הרלוונטיות. היתרון של מימוש החוקים העסקיים האלה על ידי הגדרתם כהכרזות הוא בהגדרה החד-פעמית שלהם ובאכיפתם על ידי המערכת ולא על ידי היישומים, ללא צורך בתיאורם ובתכנותם מחדש בכל תוכנית המעדכנת את בסיס הנתונים. יחד עם זאת, מאחר ומנגנון זה אינו מתיימר להוות שפת תכנות שלמה, לא ניתן להגדיר כל כלל באמצעותו, ולכן יש צורך באכיפת הכללים בתוך תוכניות יישום. בנוסף להכרזות אלו מאפשר בסיס הנתונים הפעלת פרוצדורות מיוחדות באופן אוטומטי על ידי מערכת RDBMS בעת עדכון של טבלה כלשהי: זהו מנגנון המזניק (Trigger) שיכול להכיל פקודות SQL רגילות ובחלק מהמקרים גם לוגיקה.

שמות וכינויים (SQL Identifiers)

תקן השפה מגדיר את הכללים לבניית שמות של אובייקטים שונים בבסיס הנתונים – טבלאות, עמודות וכד', שמקובל לכנות אותם **שמות SQL** (SQL Identifiers). התקן מגדיר שכל שם או כינוי חייב להיות מורכב מאותיות גדולות A עד Z, אותיות קטנות a עד z, ספרות 0 עד 9 וקו תחתי (_). התקן מגדיר מספר אילוצים על השמות:

- האורך המקסימלי של שם הוא 128 תווים.
- כל שם חייב להתחיל באות.
- שם אינו יכול להכיל רווחים.

למרות ההגדרה המקובלת הזו של השמות, קיים שוני בין המערכות המסחריות בכל הקשור לקביעת שמות.

טיפוסי נתונים (Data Types)

תקן SQL מגדיר מספר טיפוסי נתונים חוקיים והם:

טיפוס הנתונים	תיאור
CHAR (N)	מחרוזת תווים בעלת אורך קבוע של n תווים. המחרוזת יכולה להכיל כל ערך: נומרי, תו אלפא, סימנים מיוחדים.
VARCHAR (N)	מחרוזת תווים בעלת אורך משתנה כאשר n הוא האורך המירבי.
DEC (N,M)	מספר עשרוני בעל n ספרות, מתוכן m ספרות אחרי הנקודה העשרונית. לדוגמה DEC(7,2) הינו מספר בן 7 ספרות כאשר 5 הן לפני הנקודה העשרונית ו-2 אחריה.
INTEGER	מספר שלם התופס מילה שלמה במחשב ולכן יכול להגיע לערכים גבוהים מאוד.
SMALLINT	מספר שלם התופס חצי מילה במחשב ולכן יכול להכיל ערכים קטנים יותר מאשר INTEGER.
BIT (N)	מחרוזת של סיביות באורך קבוע של n.
BIT VARYING (N)	מחרוזת של סיביות באורך משתנה כאשר n הוא האורך המירבי.
DATE	תאריך קלנדרי. נתמכים מספר פורמטים שונים של תאריכים. לדוגמה YYYY/MM/DD.
TIME	זמן שעון במבנה HH:MM:SS
TIMESTAMP	חותמת זמן המכיל שילוב של התאריך והשעה.
BOOLEAN	משתנה היכול להכיל ערך אמת או שקר.
BLOB	קיצור של Binary Large Object, משתנה היכול להכיל אובייקט גדול מאוד, למשל תמונה, הקלטה של קול, מפה וכד'.
MONEY	משתנה המכיל סכום כסף. ניתן להגדיר את המבנה ואת צורת ההצגה ואז שפת SQL דואגת להציג את הסכום בצורה נכונה. למשל עם סימן $ בצד הנכון.

יצרני מערכות RDBMS הוסיפו על תקן זה טיפוסי נתונים כהבנתם. לדוגמה, מערכת DB2 Universal Database של יבמ תומכת בטיפוס נתונים מסוג BIGINT עבור מספרים שלמים גדולים בעלי 64 סיביות, ב-REAL עבור מספרים עשרוניים קטנים ו-DATALINK המצביע על אובייקט המאוחסן מחוץ לבסיס הנתונים.

הגדרת בסיס נתונים חדש

בסיס נתונים מוגדר כאוסף של טבלאות בעלות משמעות לוגית כלשהי. לכל אוסף כזה של טבלאות מקובל לקרוא בשם **סכימה** (Schema) ולתת לו שם, כמו EMPLOYEE_DB, COMPANY_SCHEMA או כל שם אחר. מערכת אחת לניהול בסיסי נתונים יכולה לנהל מספר סכימות שונות. תקן SQL2 אינו קובע כיצד מגדירים בסיס נתונים חדש ולכן המערכות המסחריות עושות זאת בצורות שונות: חלקן על ידי שימוש בפקודה מיוחדת, חלקן על ידי שימוש בתוכנית שירות מיוחדת, חלקן כחלק מתהליך התקנת המערכת וכד'. התקן גם אינו מגדיר כיצד נבנה **קטלוג המערכת** (System Catalog) ומשאירה נושא זה פתוח למפתחי המערכות לניהול בסיסי הנתונים.

התקן מגדיר פקודה להגדרת סכימה ולביטול סכימה.

תחביר הפקודה להגדרת סכימה:

1. **CREATE SCHEMA name**

פקודה זו בונה סכימה חדשה בעלת השם name.

תחביר הפקודה לביטול סכימה:

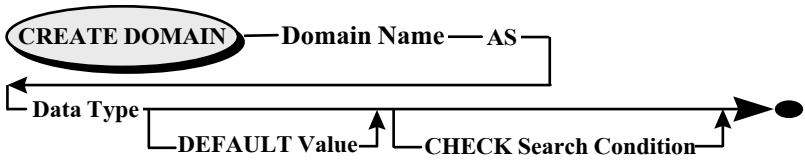
1. **DROP SCHEMA name RESTRICT/CASCADE**

פקודה זו מבטלת סכימה קיימת. פרמטר RESTRICT, שהוא ברירת המחדל, קובע שהסכימה חייבת להיות ריקה, כלומר אסור שתכיל אובייקטים כלשהם כדי שניתן יהיה לבטלה. הפרמטר CASCADE קובע שכל האובייקטים הקיימים בסכימה יבוטלו בעת ביטול הסכימה.

הגדרת מרחב ערכים (Domain Definition)

תקן SQL2 מאפשר הגדרת מרחב ערכים בצורה מפורשת, ולא רק במהלך הגדרת טיפוס הנתונים של עמודה. טיפוס הנתונים של עמודה מגדיר למעשה את מרחב הערכים שעמודה מסוימת יכולה לקבל. למשל, הגדרת העמודה ציון כבעלת טיפוס נתונים INTEGER קובעת שעמודה זו יכולה להכיל רק מספרים שלמים. מעבר לטיפוס הנתונים, ניתן להגדיר גם אילוצים נוספים על מרחב הערכים, כמו למשל שציון יכול להיות מספר שלם בין 0 ל-100 בלבד, ולא כל מספר שלם. אותה עמודה יכולה להופיע בטבלאות שונות ולכן נוח לפעמים להגדיר את מרחב הערכים של נתון מסוים פעם אחת ולהשתמש בהגדרה זו בכל המקומות האחרים. נציין שחלק מהמערכות המסחריות אינן תומכות בהגדרת מרחב הערכים.

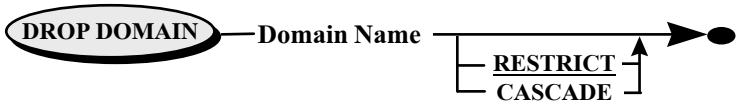
תחביר הפקודה להוספת מרחב ערכים:



תרשים 9.3: פקודה להגדרת מרחב ערכים.

הפרמטר DEFAULT קובע את ברירת המחדל, כלומר את הערך שיוכנס לתוך העמודה אם נתון זה חסר. הפרמטר CHECK מאפשר את הגדרת האילוצים על מרחב הערכים. ניתן לבדוק קיום מתוך קבוצה סופית של ערכים, טווח ערכים, או ניסוח של האילוץ על ידי שאילתת SQL.

תחביר הפקודה לביטול מרחב ערכים:



תרשים 9.4: תרשים תחביר לביטול הגדרת מרחב ערכים.

הפרמטר RESTRICT, שהוא גם ברירת המחדל, קובע שניתן לבטל מרחב ערכים רק אם אין אף עמודה בטבלה כלשהי שמתבססת על מרחב ערכים זה. הפרמטר CASCADE קובע שטיפוס הנתונים של כל העמודות המתבססות על מרחב ערכים זה יוחלף לטיפוס הנתונים של מרחב הערכים לפני הביטול.

דוגמה א' : הגדר מרחב ערכים STUDENT_ID בעל טיפוס נתונים מחרוזת תווים.

```
1.  CREATE DOMAIN STUDENT_ID AS CHAR (5)
```

דוגמה ב' : הגדר מרחב ערכים בשם EXAM_TERM בעל טיפוס נתונים מחרוזת שיכול להכיל רק את הערכים A, B או C.

```
1.  CREATE DOMAIN EXAM_TERM AS CHAR(1) DEFAULT 'A'
2.          CHECK (VALUE IN ('A', 'B', 'C'))
```

דוגמה ג' : הגדר מרחב ערכים GRADES, טיפוס נתונים נומרי שלם וטווח ערכים 0-100.

```
1.  CREATE DOMAIN GRADES AS INTEGER
2.          CHECK (VALUE BETWEEN 0 AND 100)
```

דוגמה ד' : הגדר מרחב ערכים DEPARTMENT_ID בעל טיפוס נתונים מחרוזת תווים שיכול לקבל ערכים רק מתוך עמודת DEPART בטבלת מחלקות.

```
1.  CREATE DOMAIN DEPARTMENT_ID AS CHAR (2)
2.          CHECK (VALUE IN (SELECT  DEPART FROM DEPARTMENTS))
```

נשים לב שבדוגמה זו בדיקת הערכים מתבצעת באמצעות שאילתת SQL.

הגדרת טבלה (Create Table)

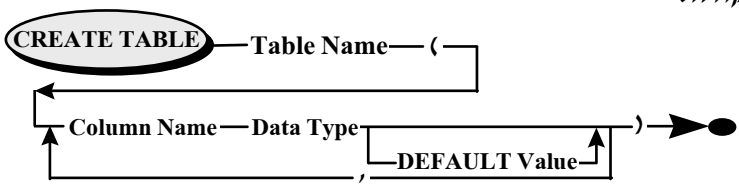
הגדרת טבלה חדשה בבסיס הנתונים מבוצעת באמצעות הפקודה CREATE TABLE. פקודה זו מכילה מספר רב של פרמטרים שרובם מתייחס לאילוצי אמינות ושלמות. הפקודה תוצג בשלבים, תחילה המבנה הבסיסי והפשוט שלה להגדרת טבלה חדשה ללא אילוצי אמינות ובהמשך נציג פרמטרים מתקדמים יותר, ונוסיף אילוצים באופן הדרגתי.

הגדרת טבלה חדשה

פקודה זו מוסיפה טבלה חדשה לבסיס הנתונים. טבלה זו תכיל את העמודות על פי המוגדר בפקודה, כאשר כל עמודה תוכל להכיל ערכים מטיפוס מסוים. מספר העמודות המקסימלי שניתן להגדיר בכל טבלה שונה וייחודי למערכות RDBMS השונות. לדוגמה, מערכת DB2 Universal Database של יבמ תומכת בטבלאות בעלות 500 עמודות, אם משתמשים בדפים (pages) באורך 4KB, ותומכת ב- 1,012 עמודות כשמשתמשים בדפים באורך 8KB.

תחזוקת הטבלה, כלומר הכנסת שורות חדשות, עדכון ערכים בשורות קיימות וביטול שורות, תבוצע באמצעות פקודות SQL רגילות לטיפול בנתונים: INSERT, DELETE ו-UPDATE.

תחביר הפקודה:



תרשים 9.5: תרשים תחביר לפקודה CREATE TABLE בסיסית.

טיפוס הנתונים של העמודה חייב להיות אחד מבין טיפוסי הנתונים הנתמכים על ידי שפת SQL. בעיקרון, בעת הוספת שורה חדשה לטבלה, כל עמודה שלא הוגדר עבורה ערך כלשהו, תקבל באופן אוטומטי ערך חסר (Null Value). הפרמטר DEFAULT מאפשר להגדיר ערך ברירת מחדל בעת פתיחת שורה חדשה. ערך ברירת המחדל יינתן רק באם העמודה אינה מכילה ערך כלשהו בפקודה INSERT.

דוגמה: הגדר את טבלת ציונים. לעמודה מועד בחינה יש לקבוע ברירת מחדל A.

```
1.  CREATE TABLE GRADES
2.          (STUDENT_ID   CHAR (5),
3.           COURSE_ID    CHAR (5,
4.           SEMESTER     CHAR (6),
5.           TERM         CHAR (1)   DEFAULT 'A',
6.           GRADE        SMALLINT,
7.           GRADE_SEM    SMALLINT)
```

בשורה 5 נקבע לעמודה TERM ערך ברירת מחדל A ולכן בעת הוספת עמודה חדשה, אם לא ניתן ערך לעמודה זו, היא תקבל את הערך A.

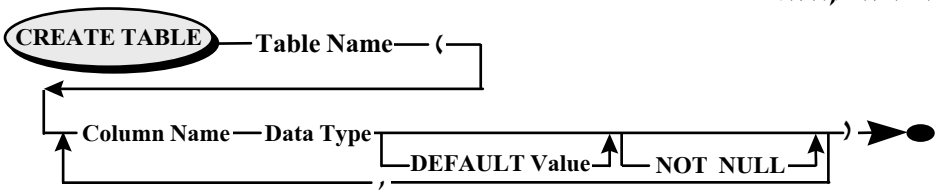
נתוני חובה (Required Data)

חלק מהעמודות המופיעות בטבלאות הן עמודות חובה, כלומר הן חייבות להכיל ערך כלשהו. לדוגמה ניתן לקבוע שמספר קורס הוא נתון חובה בטבלת ציונים. משמעות הדבר היא שאסור שתמצאנה שורות בטבלת ציונים בהן העמודה מספר קורס מכילה Null. זהו כלל אמינות פשוט ובסיסי ביותר. הדרך לבטא כלל זה היא על ידי הגדרת העמודה כ- NOT NULL, בעת הגדרת הטבלה.

עיקרון פעולה:

פקודה זו בונה טבלה ריקה חדשה. טבלה זו תכיל את העמודות על פי המוגדר בפקודה. הגדרת עמודה כ- NOT NULL גורמת למערכת RDBMS לבדוק בעת הוספת שורה חדשה שהעמודה מכילה ערך כלשהו, ולבדוק שבעת עדכון הטבלה לא יוכנס ערך Null לעמודה. חובה להגדיר אילוץ זה בעת ההגדרה הראשונה של הטבלה. לא ניתן להוסיף אילוץ זה לטבלה שכבר יש בה נתונים.

תחביר הפקודה:



תרשים 9.6: תרשים תחביר להגדרת טבלה חדשה עם נתוני חובה.

הפרמטר NOT NULL מבטיח שהעמודה לא תכיל ערך Null, כי זהו נתון חובה, כמו למשל מספר תעודת זהות בשורת הסטודנט.

דוגמה : הגדר את הטבלה ציונים. העמודות מספר סטודנט ומספר קורס הן עמודות חובה. לעמודה מועד בחינה יש לקבוע ברירת מחדל A.

```
1.  CREATE TABLE GRADES
2.          (STUDENT_ID CHAR (5)    NOT NULL,
3.           COURSE_ID   CHAR (5)    NOT NULL,
4.           SEMESTER     CHAR (6),
5.           TERM          CHAR (1)  DEFAULT 'A',
6.           GRADE         SMALLINT,
7.           GRADE_SEM  SMALLINT)
```

נשים לב שבשורות 2 ו-3 אנו מגדירים שתי עמודות כ- NOT NULL ולכן המערכת לא תרשה שיוכנס ערך Null לעמודות אלו, הן בעת הוספת שורה חדשה והן בעת עדכון שורה בטבלה.

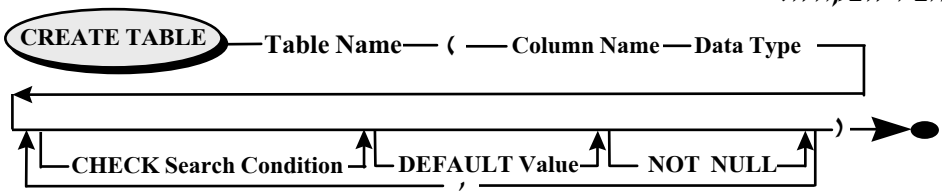
אילוץ מרחב ערכים (Domain Constraint)

לכל עמודה מוגדר טיפוס נתונים מסוים. בדרך כלל, העמודה יכולה להכיל אחד מבין כל הערכים החוקיים במרחב הערכים המוגדר על ידי טיפוס הנתונים. למשל אם העמודה היא בעלת טיפוס נתונים INTEGER היא תוכל להכיל ערך כלשהו מתוך טווח המספרים השלמים הנתמך, בין אם הוא מספר חיובי או שלילי. לפעמים יש צורך להגדיר אילוץ נוסף על מרחב ערכים זה, אילוץ המצמצם את הערכים האפשריים.

עיקרון פעולה:

הגדרת עמודה עם אילוץ מרחב ערכים, מבטיחה שמערכת RDBMS תבדוק בעת הוספת שורה חדשה או בעת עדכון השורה, שהערך המופיע בעמודה הוא במרחב הערכים המוגדר. במידה ולא, המערכת תדחה את הפקודה תוך מתן הודעה מתאימה.

תחביר הפקודה:



תרשים 9.7: תרשים תחביר עם בדיקות אילוץ מרחב ערכים

תנאי החיפוש (Search Condition) יכול להגדיר (א) בדיקת קיום בתוך קבוצת ערכים כלשהי; (ב) בדיקת טווח רציף של ערכים; ו-(ג) שאילתה המבצעת בדיקה אם הערך קיים בטבלה כלשהי.

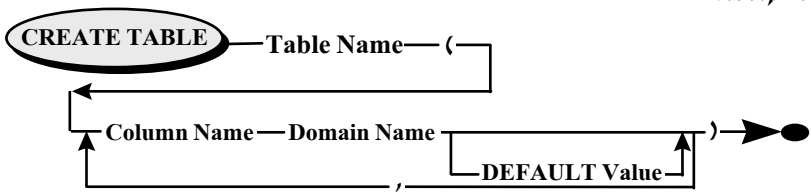
דוגמה : הגדר את טבלת ציונים. לעמודה מועד בחינה יש לקבוע אילוץ מרחב ערכים המגדיר שהיא יכולה לקבל אחד מבין הערכים A, B או C. ברירת המחדל של העמודה יהיה A. יש לקבוע אילוץ שהציון הסופי וציון הסמסטר יהיה בין 0 ל-100.

```
1.  CREATE TABLE GRADES
2.          (STUDENT_ID CHAR (5)     NOT NULL,
3.           COURSE_ID   CHAR (5)     NOT NULL,
4.           SEMESTER    CHAR (6),
5.           TERM        CHAR (1)     DEFAULT 'A'
6.                                    CHECK (VALUE IN (  'A', 'B', 'C'),
7.           GRADE       SMALLINT CHECK (VALUE BETWEEN 0 AND 100),
8.           GRADE_SEM   SMALLINT CHECK (VALUE BETWEEN 0 AND 100))
```

שימוש במרחב ערכים במקום טיפוס נתונים

במקום להגדיר את טיפוס הנתונים ואת האילוצים בפקודה CREATE TABLE ניתן להגדיר את מרחב הערכים בנפרד מהגדרת הטבלה ולהשתמש בשם מרחב הערכים בעת הגדרת העמודות השונות.

תחביר הפקודה:



תרשים 9.8: תרשים תחביר להגדרת עמודה על בסיס של מרחב ערכים.

דוגמה : הגדר את טבלת ציונים על סמך מרחבי הערכים.

הגדרת מרחבי ערכים

```
1. CREATE DOMAIN STUDENT_ID AS CHAR (5)
2. CREATE DOMAIN EXAM_TERM AS CHAR (1 )
3.            DEFAULT'A'
4.            CHECK (VALUE IN ('A' ,'B' ,'C' ))
5. CREATE DOMAIN GRADE AS  SMALLINT
6.            CHECK (VALUE BETWEEN  0  AND100)
```

הגדרת טבלה

```
1. CREATE TABLE GRADES
2.            (STUDENT_ID  STUDENT_ID    NOT NULL,
3.             COURSE_ID    CHAR (5)      NOT NULL,
4.             SEMESTER     CHAR (6),
5.             TERM         EXAM_TERM ,
6.             GRADE        GRADE,
7.             GRADE_SEM    GRADE)
```

נשים לב לכך ששם העמודה ושם המרחב יכולים להיות זהים, ויותר מעמודה אחת יכולה להתבסס על מרחב ערכים אחד.

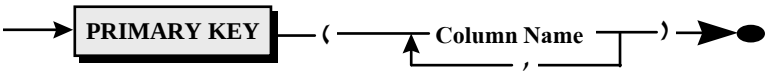
הגדרת מפתח עיקרי (Primary Key)

הגדרת המפתח העיקרי של הטבלה מתבצעת על ידי המשפט PRIMARY KEY שנמצא בפקודה CREATE TABLE.

עיקרון פעולה:

הגדרת עמודה אחת או יותר כמפתח עיקרי של טבלה גורמת למערכת RDBMS לבנות באופן אוטומטי אינדקס חד משמעי, Unique, על עמודות אלו כדי להבטיח את הזיהוי החד-ערכי של השורה בטבלה. בעת הוספת שורה חדשה לטבלה, המערכת תבטיח את הזיהוי החד-ערכי ותדחה שורה המכילה ערך שכבר קיים בתוך הטבלה.

תחביר הפקודה:



תרשים 9.9: תרשים תחביר להגדרת המפתח העיקרי של הטבלה.

בכל פקודה CREATE TABLE יכול להופיע משפט PRIMARY KEY אחד בלבד. מפתח עיקרי לא יכול להכיל ערך חסר ולכן כל עמודה המשתתפת במפתח העיקרי חייבת להכיל את האילוץ NOT NULL ההופך את העמודות המשתתפות במפתח עיקרי לעמודות חובה.

דוגמה : הגדר את טבלת ציונים עם מפתח עיקרי מספר סטודנט, מספר קורס וסמסטר.

```
1.  CREATE TABLE GRADES
2.              (STUDENT_ID CHAR (5)     NOT NULL,
3.               COURSE_ID   CHAR (5)    NOT NULL,
4.               SEMESTER    CHAR (6)    NOT NULL,
5.               TERM        CHAR (1)    DEFAULT 'A'
6.                                       CHECK (VALUE IN ( 'A', 'B', 'C'),
7.               GRADE       SMALLINT    CHECK (VALUE BETWEEN 0 AND 100),
8.               GRADE_SEM   SMALLINT    CHECK (VALUE BETWEEN 0 AND 100)
9.  PRIMARY KEY  (STUDENT_ID, COURSE_ID, SEMESTER)
```

שורה 9 מגדירה את המפתח העיקרי של הטבלה.

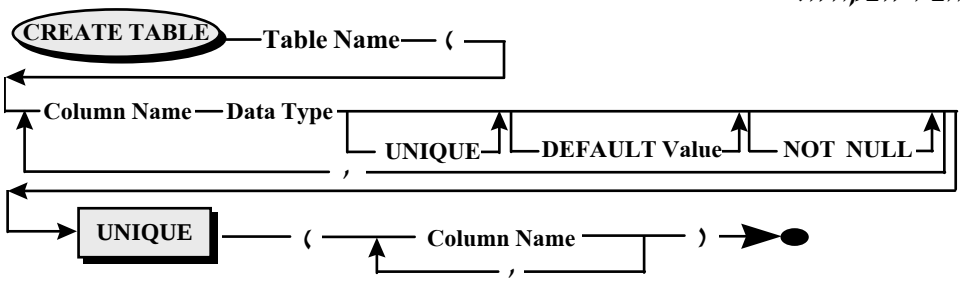
אילוץ חד-ערכיות (Unique Constraint)

בנוסף לאילוץ החד-ערכיות הקיים באופן טבעי עבור המפתח העיקרי של הטבלה, קיימת אפשרות לבקש שעמודות נוספות יקיימו אילוץ של חד-ערכיות, כלומר לא תופיע שורה נוספת באותה טבלה שבה מופיעים אותם ערכים בעמודות אלו. הגדרה זו מאפשרת להגדיר **מפתחות נוספים** (Candidate Keys) לטבלה.

עיקרון פעולה:

המערכת תבנה אינדקס חד-ערכי (Unique) על העמודות המבוקשות, כדי להבטיח זיהוי חד-ערכי. ניסיון הוספה או עדכון המכיל ערכים שכבר מופיעים בשורה כלשהי יידחה.

תחביר הפקודה:



תרשים 9.10: תרשים תחביר להגדרת עמודה חד-ערכית.

אם אילוץ החד-ערכיות הוא על עמודה בודדת, ניתן להשתמש בפרמטר UNIQUE המופיעה ליד שם העמודה. לעומת זאת, אם יש להבטיח חד-ערכיות של אוסף עמודות, יש להשתמש במשפט UNIQUE. לא כל המערכות המסחריות תומכות בצורה זו להגדרת אילוץ חד-ערכי, אלא משתמשות למימוש אילוץ החד-ערכיות בהגדרת **אינדקס חד-ערכי** – CREATE UNIQUE INDEX.

דוגמה א' : הגדר את טבלת מחלקות. יש להבטיח שראש מחלקה יופיע פעם אחת בלבד.

```
1. CREATE TABLE DEPARTMENTS
2.          (DEPART CHAR (5)    NOT NULL,
3.           NAME    CHAR (25),
4.           HEAD    CHAR (30)  NOT NULL UNIQUE,
5. PRIMARY KEY (DEPART)
```

בשורה 4 מופיע האילוץ של חד-ערכיות ראש מחלקה. ניתן לרשום את האילוץ הזה גם בצורה הבאה:

```
1. CREATE TABLE DEPARTMENTS
2.          (DEPART CHAR (5)   NOT NULL,
3.           NAME    CHAR (25),
4.           HEAD    CHAR (30)  NOT NULL,
5. UNIQUE  (HEAD),
6. PRIMARY KEY  (DEPART)
```

דוגמה ב' : הגדר את טבלת מחלקות. יש להבטיח שהצירוף של מחלקה וראש מחלקה יופיע פעם אחת בלבד.

```
1. CREATE TABLE DEPARTMENTS
2.          (DEPART CHAR (5)    NOT NULL,
3.           NAME    CHAR (25),
4.           HEAD    CHAR (30)    NOT NULL,
5. UNIQUE  (HEAD, DEPART),
6. PRIMARY KEY  (DEPART)
```

שלמות קשרים בין טבלאות (Referential Integrity)

מנקודת מבט מערכת RDBMS, בסיס הנתונים מורכב מאוסף של טבלאות הקשורות ביניהן על ידי קשרי אב/בן המבטאים **יחסים** (Relations). בדומה לאחריות המערכת לשמור על שלמות ואמינות הערכים המנוהלים בטבלה הבודדת, יש לה גם אחריות לנהל ולשמור על שלמות הקשרים בין הטבלאות השונות, כדי להבטיח שבבסיס הנתונים הקשרים, או היחסים האלה, יהיו נכונים.

אם נתבונן לרגע ב"בסיס הנתונים לדוגמה" המוצג בסוף פרק 7, נראה שהוא מורכב מארבע טבלאות ומשלושה קשרים בין הטבלאות. להדגמת הגדרת הקשרים ואילוצי השלמות והאמינות של הקשרים, נשתמש בדוגמה מתרשים 9.1 שבראש הפרק והמתייחסת לשתי טבלאות בלבד: מחלקות וקורסים.

עיקרון פעולה:

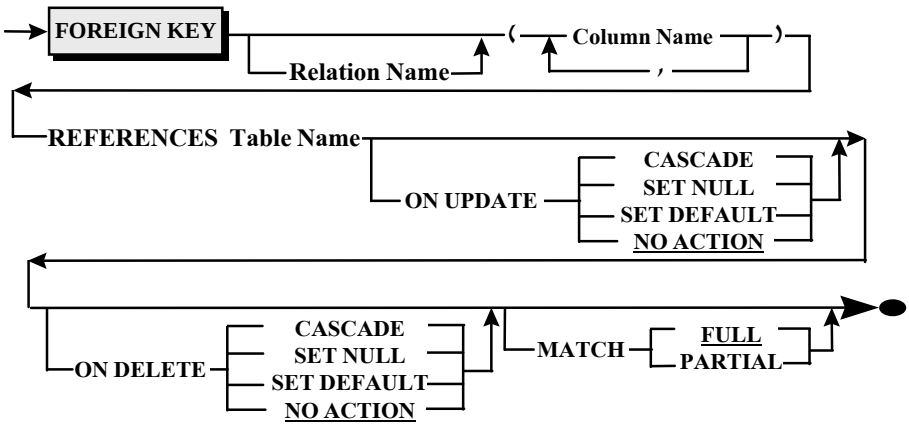
הגדרת שלמות הקשרים בין טבלאות מתבצע על ידי משפט FOREIGN KEY בתוך הפקודה CREATE TABLE. נתייחס לעיקרון הפעולה בשתי נקודות זמן שונות: בזמן הגדרת הטבלה ובזמן עדכון תוכן הטבלה. אפשר לראות זאת בתרשים 9.11.

- **בדיקות בעת הגדרת הטבלה:** בזמן העיבוד של הפקודה CREATE TABLE המכילה משפט FOREIGN KEY, מתבצעת בדיקה שהמפתח הזר אכן מופיע כמפתח עיקרי בטבלת האב ושטיפוס הנתונים של שתי העמודות זהה. מסיבה זו יש להגדיר תחילה את טבלאות האב ורק לאחר מכן – את טבלאות הבן.
- **בדיקות בזמן עדכון תוכן הטבלה:** בזמן ביצוע הפקודה INSERT להוספת שורה חדשה לטבלת הבן, תתבצע בדיקה שהערך של המפתח הזר קיים בעמודת המפתח העיקרי בטבלת האב. בזמן ביטול שורה מטבלת האב או בזמן עדכון של שורה בטבלת האב, תשמור המערכת שהעדכון לא ישבש את הקשר. ניתן לבחור במספר חלופות לצורת הטיפול בכל השורות בטבלת הבן ושהביטול או העדכון בטבלת האב נוגע להן.

ניתן להגדיר **שם לאילוץ** (Relation Name), אבל השימוש בו ב-SQL מצומצם מאוד ומשמש בעיקר לתיעוד ולזיהוי הודעות שגיאה. מילת המפתח MATCH מאפשרת להגדיר שתי דרכים לבדיקת התאמת המפתח הזר עם המפתח העיקרי, ובעיקר עבור מצבים בהם המפתח הזר יכול להכיל Null.

- **MATCH FULL** – ערכי המפתח הזר יכולים להיות Null או חייבים להכיל ערכים. זו ברירת המחדל.
- **MATCH PARTIAL** – ערכי המפתח הזר חייבים להיות Null או שיש לפחות שורה אחת בטבלת האב שיכולה לספק את האילוץ, אילו כל שאר הערכים החסרים היו מוחלפים בערכים כלשהם.

תחביר הפקודה:

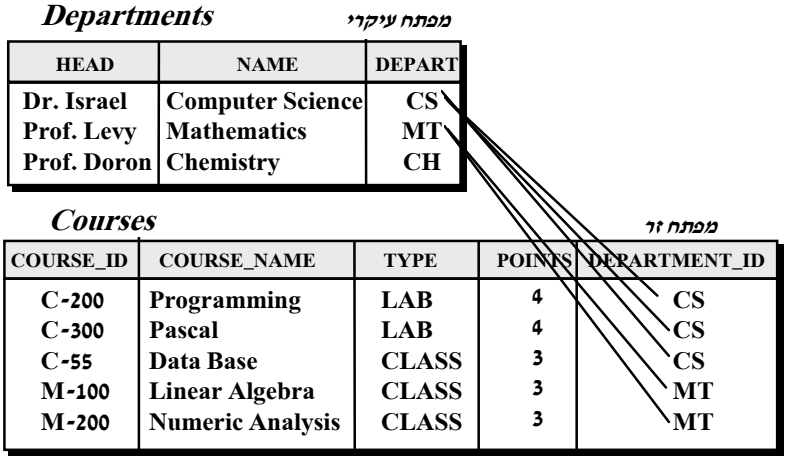


תרשים 9.11: תרשים תחביר למשפט FOREIGN KEY.

מילת המפתח ON DELETE מאפשרת לבחור במספר דרכים לפעולה בעת ביטול שורה:

- **DELETE CASCADE** – בעת ביטול רשומה בטבלת האב יש לבטל באופן אוטומטי את **כל** השורות בטבלת הבן המתייחסות לשורה המבוטלת. תופעת **המפל** (Cascade) אומרת שביטול שורה בטבלת בן כלשהי עשויה לגרום לביטול שורות בטבלאות נוספות, אם טבלת הבן היא עצמה טבלת אב בקשר אחר, וכך הלאה, כמו במפל רב-דרגות. תרשים 9.12 מציג את מצב בסיס הנתונים לאחר ביטול שורת המחלקה למינהל עסקים מטבלת מחלקות.

נוכל לראות שמערכת RDBMS ביטלה באופן אוטומטי את כל השורות בטבלה קורסים המתייחסות לשורת המחלקה שבוטלה.



תרשים 9.12: מצב בסיס הנתונים לאחר ביטול שורת מחלקה עם DELETE CASCADES.

- **DELETE SET NULL** – בעת ביטול שורה בטבלת האב יש לקבוע את הערך Null לכל השורות בטבלאות הבן המתייחסות לשורה המבוטלת. כמובן שניתן לבחור באפשרות זו רק אם בטבלת הבן עמודת המפתח הזר אינה מוגדרת כ- NOT NULL. תרשים 9.13 מציג את מצב בסיס הנתונים לאחר ביטול שורת המחלקה למינהל עסקים.

נוכל לראות בתרשים ששתי השורות בטבלת קורסים שהתייחסו לשורת המחלקה שבוטלה, קיבלו ערך Null בעמודה מספר מחלקה.

Departments *מפתח עיקרי*

HEAD	NAME	DEPART
Dr. Israel	Computer Science	CS
Prof. Levy	Mathematics	MT
Prof. Doron	Chemistry	CH

Courses *מפתח זר*

COURSE_ID	COURSE_NAME	TYPE	POINTS	DEPARTMENT_ID
C-200	Programming	LAB	4	CS
C-300	Pascal	LAB	4	CS
C-55	Data Base	CLASS	3	CS
M-100	Linear Algebra	CLASS	3	MT
M-200	Numeric Analysis	CLASS	3	MT
B-10	Marketing	CLASS	2	Null
B-40	Operations Res.	SEMIN	3	Null

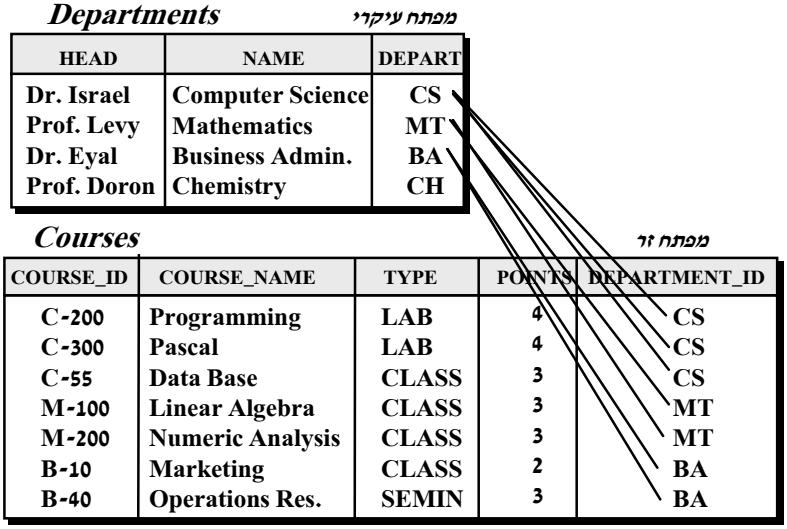
תרשים 9.13: מצב בסיס הנתונים לאחר ביטול שורת מחלקה עם DELETE SET NULL.

- **DELETE SET DEFAULT** – כשמבטלים שורה בטבלת האב יש לקבוע את ערך ברירת המחדל לכל השורות בטבלאות הבן המתייחסות לשורה המבוטלת. ניתן לבחור באפשרות זו, רק אם בטבלת הבן עמודת המפתח הזר מכילה הגדרת DEFAULT לקביעת ברירת מחדל. על ברירת המחדל להיות ערך קיים בעמודת המפתח העיקרי.

- **DELETE NO ACTION** – בעת ביטול שורה בטבלת האב תתבצע בדיקה אם קיימות שורות בטבלאות הבן ואם נמצאו כאלו, פעולת הביטול לא תבוצע. זו ברירת המחדל של פעולת הביטול. במקרה זה, על היישום לבטל תחילה את כל השורות בטבלאות הבן באמצעות הפקודה DELETE רגילה ורק לאחר מכן לבטל את השורה בטבלת האב. שיטה זו מבטיחה ששורות בטבלת הבן לא תבוטלנה באופן מקרי וללא כוונה תחילה של היישום. בתקן הישן, לאפשרות זו קראו RESTRICTED. בדוגמה שלנו, המערכת לא תרשה לבטל את שורת המחלקה למינהל עסקים, מאחר וקיימות שורות בטבלת קורסים המתייחסות לשורה שמבקשים לבטל.

מילת המפתח ON UPDATE מאפשרת לבחור במספר דרכי פעולה בעת עדכון שורה. לצורך העניין, שינוי הערך של העמודה המכילה את המפתח העיקרי בטבלת האב, גורם להפעלת האפשרות שנבחרה.

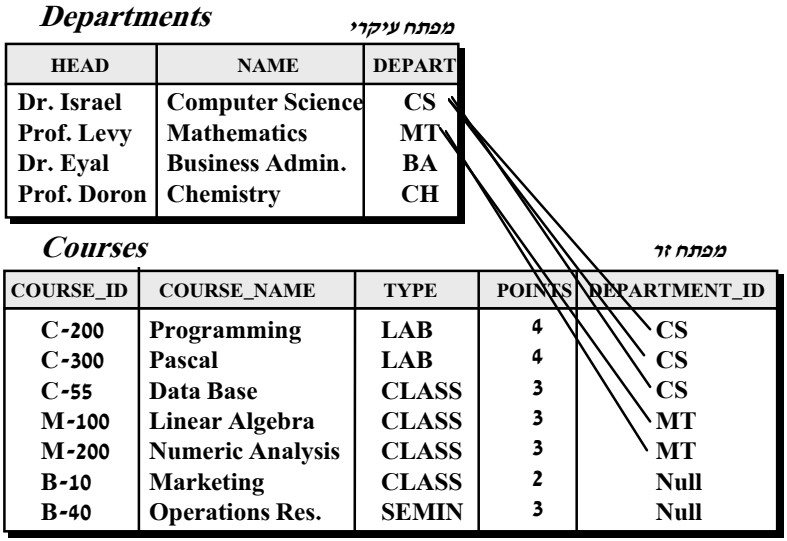
- **UPDATE CASCADE** – בעת עדכון שורה בטבלת האב יוכנס הערך החדש לכל השורות בטבלת הבן המתייחסות לשורה המעודכנת. תופעת המפל (Cascade) משמעותה שעדכון שורה בטבלת הבן, יכולה לגרום לעדכון שורות בטבלאות נוספות, אם טבלת הבן היא בעצמה טבלת אב בקשר אחר וכך הלאה, כמו במפל רב-דרגות. תרשים 9.14 מציג את מצב בסיס הנתונים לאחר שביצענו עדכון של שורת המחלקה למינהל עסקים. מספר המחלקה הוחלף מ-BS ל-BA ושם המחלקה הוחלף מ-Business ל- Business Admin.



תרשים 9.14: מצב בסיס הנתונים לאחר עדכון שורת מחלקה עם UPDATE CASCADE.

נוכל לראות שמערכת RDBMS עדכנה באופן אוטומטי את המפתח הזר בטבלת הקורסים, ובמקום הערך BS מופיע עכשיו הערך BA.

- **UPDATE SET NULL** – אפשרות זו קובעת שבעת עדכון שורה בטבלת האב, יש לקבוע את הערך Null לכל השורות בטבלאות הבן המתייחסות לשורה המעודכנת. כמובן שאפשרות זו תקפה רק אם בטבלת הבן, עמודת המפתח הזר אינה מוגדרת כ- NOT NULL. תרשים 9.15 מציג את מצב בסיס הנתונים לאחר שעדכנו את מספר המחלקה למינהל עסקים מ-BS ל-BA.



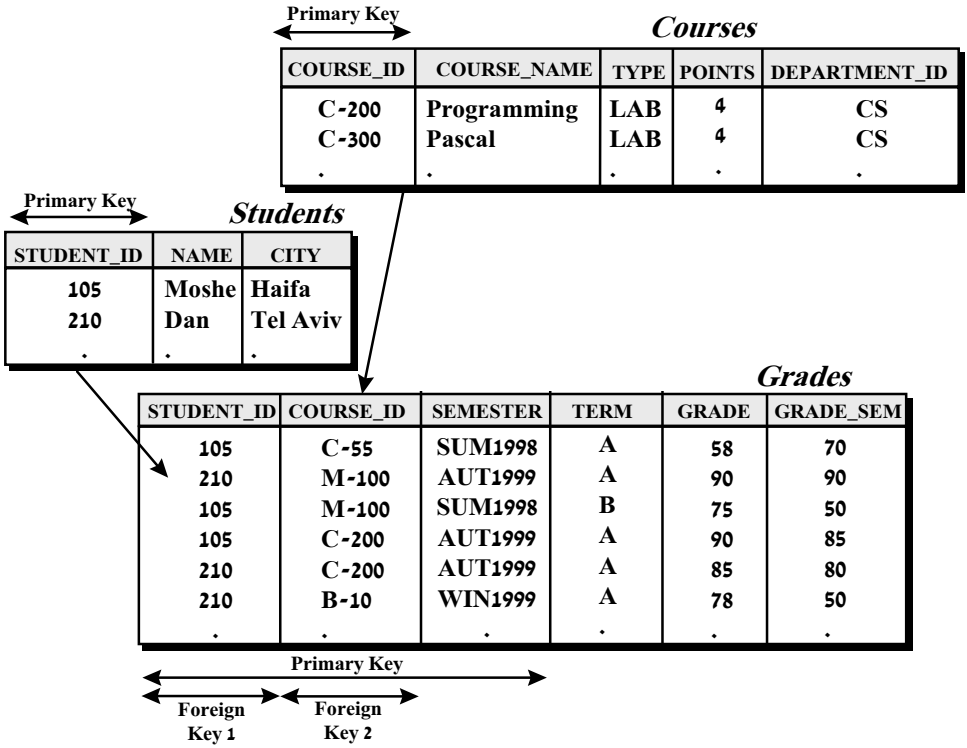
תרשים 9.15: מצב בסיס הנתונים לאחר עדכון שורת מחלקה עם UPDATE SET NULL.

נשים לב שבטבלת מחלקות בוצע העדכון למספר החדש, ובשורות המתייחסות למחלקה זו הוחלף הקוד BS בערך Null.

- **UPDATE SET DEFAULT** – בעת עדכון שורה בטבלת האב יש לקבוע את ערך ברירת המחדל לכל השורות בטבלאות הבן המתייחסות לשורה המבוטלת. כמובן שניתן לבחור אפשרות זו רק אם בטבלת הבן עמודת המפתח הזר מכילות הגדרת DEFAULT לקביעת ברירת מחדל. הערך של ברירת המחדל חייב להיות ערך קיים בתוך עמודת המפתח העיקרי.
- **UPDATE NO ACTION** – בעת עדכון שורה בטבלת האב תתבצע בדיקה אם קיימות שורות בטבלת הבן. אם נמצאו כאלו, פעולת העדכון לא תבוצע. זו ברירת המחדל של פעולת העדכון. במקרה זה, על היישום לבטל תחילה את כל השורות בטבלת הבן באמצעות פקודה DELETE רגילה, ורק לאחר מכן לעדכן את השורה בטבלת האב. בתקן הישן, אפשרות זו קרויה RESTRICTED. בדוגמה שלנו, המערכת לא תרשה לעדכן את מספר המחלקה בשורת המחלקה מינהל עסקים, מכיון שקיימות שורות בטבלה קורסים המתייחסות לשורה שמבקשים לעדכן.

דוגמה: יש להגדיר את אילוץ שלמות הקשרים (RI) עבור הטבלה ציונים. תרשים 9.16 מציג את המפתחות העיקריים והזרים של שלוש הטבלאות הרלוונטיות.

נשים לב שבטבלה ציונים שבתרשים 9.16 יש מפתח עיקרי המורכב משלוש עמודות וגם שני מפתחות זרים: מספר סטודנט שהוא המפתח עיקרי בטבלת סטודנטים ו- מספר קורס שהוא מפתח עיקרי בטבלת קורסים. יש להגדיר עבור שני המפתחות הזרים את האפשרות NO ACTION עבור ביטול ו-SET NULL עבור עדכון.



תרשים 9.16: מפתחות עיקריים וזרים בבסיס הנתונים לדוגמה.

נוכל לראות ששורות 10 עד 13 מגדירות את המפתח הזר הראשון ושורות 14 עד 16 מגדירות את המפתח הזר השני.

```
1 . CREATE TABLE GRADES
2 .            (STUDENT_ID CHAR (5)  NOT NULL,
3 .             COURSE_ID    CHAR (5) NOT NULL,
4 .             SEMESTER     CHAR (6) NOT NULL,
5 .             TERM         CHAR (1 ) DEFAULT'A'
6 .                                   CHECK (VALUE IN ('A' ,'B' ,'C' )),
7 .             GRADE      SMALLINT CHECK (VALUE BETWEEN      0  AND100),
8 .             GRADE_SEM  SMALLINT CHECK (VALUE BETWEEN  0  AND100),
9   PRIMARY KEY  (STUDENT_ID, COURSE_ID, SEMESTER),
10.  FOREIGN KEY   (STUDENT_ID) REFERENCES STUDENTS
12 .                              ON DELETE NO ACTION
13 .                               ON UPDATE SET NULL,
14.  FOREIGN KEY   (COURSE_ID) REFERENCES COURSES
15 .                              ON DELETE NO ACTION
16 .                            ON UPDATE SET NULL   )
```

אילוצים ברמת הטבלה (Check)

עד כאן הצגנו את האפשרות להגדיר אילוצים שונים ברמת העמודה הבודדת, כמו למשל בדיקת טווח ערכים או בדיקת הערכים מול קבוצה מוגדרת מראש. לעיתים יש צורך בהגדרת אילוצים מורכבים יותר, אילוצים ברמת הטבלה ולא ברמת הערכים המופיעים בעמודה מסוימת. שפת SQL מאפשרת הגדרת אילוצים אלה בזמן הגדרת טבלה.

עיקרון פעולה:

האילוצים המוגדרים על ידי משפט CHECK בפקודה CREATE TABLE, נבדקים באופן אוטומטי על ידי המערכת, בכל עדכון של הטבלה. אם האילוץ מתקיים, עדכון הטבלה מתבצע. אם האילוץ מופר כתוצאה מהעדכון, פעולת העדכון נדחית עם הודעת שגיאה מתאימה.

תחביר הפקודה:

תרשים 9.17: תחביר המשפט לבדיקת אילוץ ברמת הטבלה.

ניתן לתת לאילוץ שם חד-ערכי המזהה אותו. למרות שמתן שם לאילוץ הוא רשות, מומלץ לתת שמות לאילוצים, מאחר ובשלב מאוחר יותר ניתן יהיה להשתמש בשם זה כדי לבטל את האילוץ על ידי הפקודה ALTER TABLE, המאפשרת שינויים של הגדרת הטבלה. יתרון נוסף למתן שמות לאילוצים הוא שניתן לקבוע האם האילוץ ייבדק מייד (Immediate) לאחר כל פקודת עדכון של טבלה או שהוא ייבדק באופן מושהה (Deferred), רק בעת סיום מוצלח של תנועה. נושא זה יפורט בפרק העוסק בעיבוד תנועות בסביבת SQL.

תנאי החיפוש (Search Condition) מוגדר כשאילתה רגילה שלפנייה מופיע התנאי EXISTS (קיים) או התנאי NOT EXISTS (אינו קיים).

דוגמה : נניח שהמכללה קבעה כלל, שסטודנט אינו יכול להיבחן במשך כל לימודיו ביותר מ-100 בחינות, בכל הקורסים שהוא לומד.

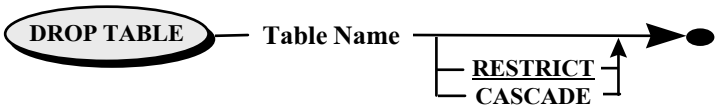
נוכל לראות שבשורות 16 עד 20 הוספנו להגדרת הטבלה את המשפט CHECK עם בדיקת קיום של מספר סטודנט כפי שמופיע בהקבצה. אם התוצאה של השאילתה הזאת יהיה True, הוספת שורה חדשה תידחה בגלל הדרישה NOT EXISTS.

```
1.  CREATE TABLE GRADES
2.              (STUDENT_ID CHAR (5)      NOT NULL,
3.               COURSE_ID   CHAR (5)     NOT NULL,
4.               SEMESTER    CHAR (6)     NOT NULL,
5.               TERM        CHAR (1)     DEFAULT 'A'
6.                                       CHECK (VALUE IN ('A', 'B', 'C'),
7.               GRADE        SMALLINT CHECK (VALUE BETWEEN 0 AND 100),
8.               GRADE_SEM  SMALLINT CHECK (VALUE BETWEEN 0 AND 100))
9.   PRIMARY KEY  (STUDENT_ID,  COURSE_ID, SEMESTER)
10. FOREIGN KEY   (STUDENT_ID) REFERENCES STUDENTS
11.                            ON DELETE NO ACTION
12.                            ON UPDATE SET NULL)
13. FOREIGN KEY   (COURSE_ID)  REFERENCES COURSES
14.                            ON DELETE NO ACTION
15.                            ON UPDATE SET NULL)
16. CONSTRAINT MAX_NUMBER_OF_EXAMS
17.         CHECK (NOT EXISTS     (SELECT      STUDENT_ID
18.                            FROM        GRADES
19.                            GROUP BY STUDENT_ID
20.                            HAVING      COUNT (*) > 100))
```

ביטול טבלה מבסיס הנתונים (Drop Table)

פקודה זו מאפשרת לבטל טבלה קיימת בבסיס הנתונים. אם הטבלה מכילה שורות, הן תבוטלנה יחד עם כל ההגדרות הקשורות לטבלה והמופיעות בקטלוג המערכת.

תחביר הפקודה:



תרשים 9.18: תרשים תחביר לביטול טבלה

הפרמטר RESTRICT קובע שאם יש טבלאות אחרות המתייחסות אל הטבלה המבוטלת כאל טבלת אב, הביטול לא יבוצע. הפרמטר CASCADE קובע שאם יש טבלאות אחרות המתייחסות אל הטבלה המבוטלת כאל טבלת אב, הן תבוטלנה. ביטול טבלאות אחרות אלו עשוי לגרור ביטולים נוספים, במידת הצורך.

דוגמה : בטל את טבלת הציונים.

```
1.  DROP TABLE GRADES RESTRICTED
```

הבחירה בפרמטר RESTRICTED מיותרת במקרה זה, כי בדוגמה שלנו הטבלה ציונים משמשת רק כטבלת בן בשני קשרים שונים – עם הטבלה קורסים ועם הטבלה סטודנטים – אבל אינה משמשת כטבלת אב באף קשר אחר.

שינוי טבלה (Alter Table)

לעיתים יש צורך להכניס שינוי במבנה לטבלה לאחר שזו כבר הוגדרה ואולי אף מאוכלסת בנתונים. ניתן לבצע מטלה זו באמצעות הפקודה ALTER TABLE המאפשרת לבצע שינויים מסוימים בטבלה קיימת וביניהם: הוספת עמודה חדשה, ביטול עמודה קיימת, הוספת אילוץ לעמודה קיימת, ביטול אילוץ מעמודה, קביעת ברירת מחדל לעמודה קיימת וביטול ברירת מחדל מעמודה. מסיבות שונות, חלק ממערכות RDBMS אינן תומכות באופן מלא בכל הפרמטרים לשינוי הגדרת הטבלאות. מה שמוצג בהמשך מבוסס על תקן SQL2, שכפי שכבר הודגש מספר פעמים, עדיין אינו נתמך באופן מלא על ידי כל המערכות המסחריות.

הוספת עמודה

פקודה זו מאפשרת להוסיף עמודה חדשה לטבלה קיימת. בעיקרון, האפשרויות הקיימות עבור עמודה חדשה בפקודה ALTER TABLE דומות לאלו שקיימות עבור עמודה בעת הגדרת טבלה חדשה. העמודה החדשה מתווספת כעמודה אחרונה ומקבלת ערך Null. אם הטבלה עוד ריקה, ניתן להוסיף עמודה חדשה עם אילוץ NOT NULL. אם הטבלה מכילה כבר שורות, לא ניתן להוסיף עמודה חדשה עם אילוץ NOT NULL.

תחביר הפקודה:

תרשים 9.19: תרשים תחביר להוספת עמודה חדשה

דוגמה: יש להוסיף לטבלת סטודנטים עמודה חדשה שתכיל את תאריך הלידה של הסטודנט.

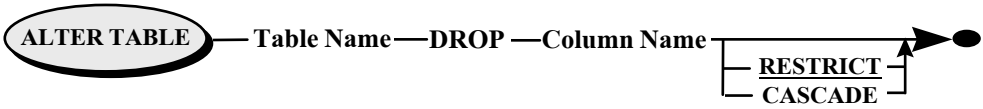
```
1. ALTER TABLE STUDENTS
2.         ADD BIRTH_ DATE    DATE
```

העמודה החדשה תאריך לידה, תתוסף כעמודה אחרונה, לאחר העמודה CITY. לאחר ביצוע הפקודה הטבלה תכיל עמודה חדשה, ובכל השורות הקיימות של עמודה זו יוצב הערך Null. אין זה אומר שבפועל מערכת RDBMS קובעת ערך זה לכל השורות. מה שקורה הוא, שהגדרת העמודה החדשה מצטרפת להגדרת הטבלה, ורק כאשר פקודת עדכון תעדכן שורה קיימת, העמודה החדשה תקבל את הערך החסר. מסיבה זו ניתן להוסיף עמודות לטבלאות קיימות בצורה קלה ונוחה, גם אם הן מכילות כבר נתונים.

ביטול עמודה קיימת

פקודה זו מאפשרת לבטל עמודות מתוך טבלה. חלק מהמערכות המסחריות אינן תומכות בפקודה זו. אם זה המצב, ניתן לוותר על ביטול העמודה ולהשאירה בטבלה, תוך בזבוז מסוים של שטח דיסק. אם העמודה מכילה Null בכל השורות, ניתן לומר שהשארתה לא מנוצלת היא כמעט ללא השפעה. אם בכל זאת רוצים לבטל את העמודה, יש להשתמש ב**תוכנית שירות לפריקה** (Unload Utility) של הטבלה מבסיס הנתונים **ובתוכנית שירות לטעינה** (Load Utility) לאחר שמגדירים את הטבלה מחדש, וללא העמודה שרצינו לבטל.

תחביר פקודה:



תרשים 9.20: תרשים תחביר לביטול עמודה קיימת.

הפרמטר RESTRICT מונע את ביטול העמודה אם העמודה שיש לבטל משמשת למטרת ייחוס כלשהו – למשל היא מופיע במשפט FOREIGN KEY בטבלה כלשהי. הפרמטר CASCADE מבטל את העמודה ואת כל ההצבעות עליה מכל שאר האובייקטים בבסיס הנתונים.

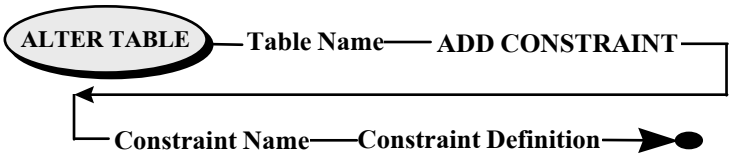
דוגמה : בטל את העמודה תאריך לידה מטבלת סטודנטים.

```
1. ALTER TABLE STUDENTS
2.         DROP BIRTH_ DATE
```

הוספת אילוץ

פקודה זו מאפשרת להוסיף אילוץ ברמת הטבלה. צורת הגדרת האילוץ דומה להגדרת אילוצים על עמודות בעת הגדרת טבלה חדשה.

תחביר פקודה:



תרשים 9.21: תרשים תחביר להוספת אילוץ.

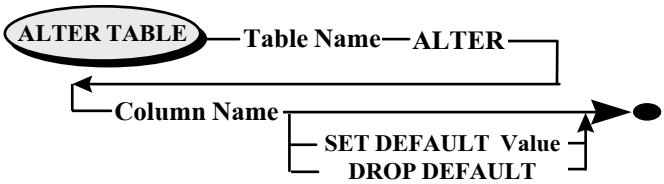
דוגמה : הוסף אילוץ של בדיקת חד-ערכיות לעמודה CITY בטבלת סטודנטים.

```
1. ALTER TABLE STUDENTS
2.         ADD CONSTRAINT UNIQUE (CITY)
```

שינוי ברירת מחדל לעמודה

פקודה זו מאפשרת שינוי ברירת מחדל של טבלה, או ביטול ברירת מחדל של עמודה.

תחביר פקודה:



תרשים 9.22: תרשים תחביר לשינוי ברירת מחדל.

דוגמה : יש לשנות את ברירת המחדל של העמודה TERM בטבלת ציונים מ-A ל-B. נבצע פעולה זו על ידי שתי פקודות ALTER TABLE, אחת כדי לבטל את ברירת המחדל הנוכחית ופקודה שנייה כדי לקבוע את הערך החדש.

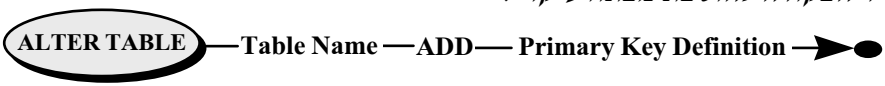
ALTER TABLE GRADES ALTER TERM DROP DEFAULT

ALTER TABLE GRADES ALTER TERM SET DEFAULT 'B'

הוספה או ביטול של מפתח עיקרי

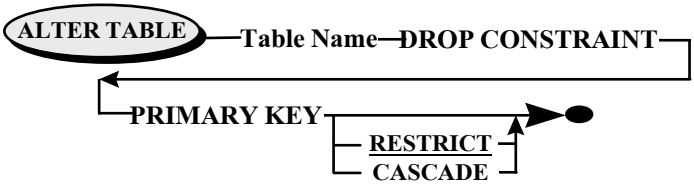
ניתן להוסיף לטבלה מפתח עיקרי, אם בשלב הקמת הטבלה לא הוגדר לה מפתח עיקרי. לא ניתן להוסיף מפתח עיקרי לטבלה שמכילה נתונים. כמו כן ניתן לבטל את המפתח העיקרי של הטבלה.

תחביר הפקודה להוספת מפתח עיקרי:



תרשים 9.23: תרשים תחביר להוספת מפתח עיקרי.

תחביר הפקודה לביטול מפתח עיקרי:



תרשים 9.24: תרשים תחביר לביטול מפתח עיקרי.

הפרמטר RESTRICT קובע שאם המפתח העיקרי מופיע בקשר אב/בן כלשהו, יש למנוע את הביטול. הפרמטר CASCADE יבטל את המפתח העיקרי ואת כל הקשרים שבו הוא מופיע.

דוגמה א' : יש להוסיף מפתח עיקרי לטבלת סטודנטים (בהנחה שבעת הגדרת הטבלה לא הוגדר המפתח העיקרי של הטבלה).

```
1. ALTER TABLE STUDENTS
2.        ADD PRIMARY KEY STUDENT_ID
```

דוגמה ב' : יש לבטל את מפתח עיקרי של טבלת סטודנטים.

```
1. ALTER TABLE STUDENTS
2.        DROP PRIMARY KEY
```

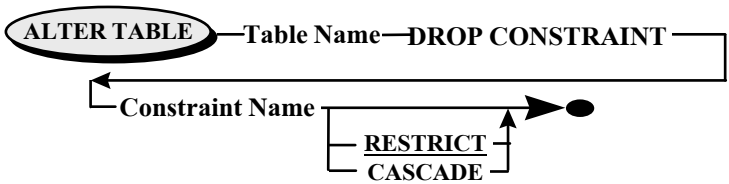
הוספת או ביטול מפתח זר

ניתן להוסיף לטבלה מפתח זר בכל נקודת זמן. כמו כן ניתן לבטל את המפתח העיקרי של הטבלה.

תחביר הפקודה להוספת מפתח עיקרי:

תרשים 9.25: תרשים תחביר להוספת מפתח זר.

תחביר הפקודה לביטול מפתח זר:



תרשים 9.26: תרשים תחבר לביטול מפתח זר.

שינויים מורכבים בהגדרת טבלאות

לעיתים, יש צורך לבצע שינוי משמעותי בטבלה (לדוגמה הרחבת עמודה, שינוי אילוצים מסוימים וכד') כאשר לא כל הפרמטרים נתמכים על ידי המערכת ולעיתים גם לא על ידי תקן השפה. לביצוע מטלה זו ניתן להגדיר טבלה חדשה באמצעות הפקודה CREATE TABLE ולטעון אותה במהירות ובנוחות על ידי הפקודה INSERT.

דוגמה : נניח שכתוצאה משינוי מדיניות המכללה נוצר הצורך להרחיב את העמודה של מועד בחינה ולאפשר מספר צירופים נוספים. במקום מחרוזת עם תו אחד יש צורך להרחיב את ההגדרה למחרוזת בת שני תווים ולשנות את הערכים האפשריים בהתאם. כל ערך קיים, כגון A יהפוך ל-A1 וכד' ויש לאפשר שני ערכים חדשים A2 ו-B2. נבצע שינוי זה במספר שלבים.

❖ נגדיר טבלה זמנית חדשה בשם GRADES_TEMP. בטבלה זו הרחבנו את הגדרת העמודה TERM וסילקנו את בדיקת התחום. כמו כן, בטבלה הזמנית לא הגדרנו את המפתח העיקרי ואת הקשרים עם טבלאות אחרות.

```
1. CREATE TABLE GRADES_TEMP
2.          (STUDENT_ID CHAR (5)   NOT NULL,
3.           COURSE_ID   CHAR (5)   NOT NULL,
4.           SEMESTER    CHAR (6)   NOT NULL,
5.           TERM        CHAR (2) ,
6.           GRADE       SMALLINT CHECK (VALUE BETWEEN 0 AND 100),
7.           GRADE_SEM   SMALLINT CHECK (VALUE BETWEEN 0 AND 100))
```

❖ נעתיק את הטבלה הקיימת לטבלה הזמנית.

```
1. INSERT INTO GRADES_TEMP
2.    SELECT  *
3.    FROM    GRADES
```

❖ על ידי שימוש בפקודות עדכון נעדכן את הטבלה הזמנית, כך שכל הערכים הקיימים יוחלפו בערכים החדשים.

```
1. UPDATE GRADES_TEMP
2.    SET     TERM = A1
3.    WHERE   TERM = 'A'
```

❖ נבטל את הטבלה GRADES, לאחר שנוודא שכל השורות אכן הועתקו כהלכה לטבלת הזמנית.

```
1. DROP TABLE GRADES
```

❖ נגדיר מחדש את הטבלה GRADES שכוללת הפעם את האילוץ הנכון. ניתן לעשות זאת גם על ידי הפקודה ALTER TABLE, אבל לצורך ההדגמה הנחנו שהמערכת אינה תומכת בשינוי אילוצים.

```
1.  CREATE TABLE GRADES
2.          (STUDENT_ID CHAR (5)  NOT NULL,
3.           COURSE_ID   CHAR (5) NOT NULL,
4.           SEMESTER    CHAR (6) NOT NULL,
5.           TERM        CHAR (2) DEFAULT 'A'
6.                       CHECK (VALUE IN ('A1', 'A2', 'B1','B2', 'C1') ,
7.           GRADE       SMALLINT CHECK (VALUE BETWEEN  0 AND 100),
8.           GRADE_SEM SMALLINT CHECK (VALUE BETWEEN  0 AND 100)
9.   PRIMARY KEY (STUDENT_ID, COURSE_ID, SEMESTER, TERM)
10.  FOREIGN KEY  (STUDENT_ID) REFERENCES STUDENTS
11.                            ON DELETE NO ACTION
12.  FOREIGN KEY (COURSE_ID)   REFERENCES COURSES
13.                            ON DELETE NO ACTION)
```

❖ נעתיק את כל השורות מהטבלה הזמנית אל הטבלה החדשה.

```
1. INSERT INTO GRADES
2.     SELECT  *
3.     FROM     GRADES_TEMP
```

❖ נבטל את הטבלה הזמנית, לאחר שנוודא שכל השורות הועתקו כהלכה לטבלה ציונים החדשה.

```
1. DROP TABLE GRADES_TEMP
```

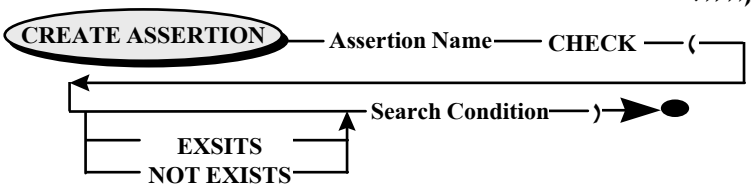
אילוצים ברמת בסיס הנתונים – הכרזות (Assertions)

עד כאן הצגנו הגדרת אילוצים שונים ברמת העמודה בטבלה וברמת הטבלה הבודדת, תוך שימוש בפרמטר CHECK של הפקודה CREATE TABLE. שפת SQL תומכת בסוג נוסף של אילוץ, שמוגדר ברמת בסיס הנתונים כולו ולא ברמת הטבלה הבודדת. משמעות הדבר היא שבהגדרת האילוץ יכולים, אבל לא חייבים, להשתתף מספר טבלאות.

עיקרון פעולה:

מערכת RDBMS עוקבת אחר כל פעולה של שינוי – הוספה, עדכון, ביטול – בכל אחת מהטבלאות שעבורן הוגדר אילוץ. ברגע שמופעלת פקודה לשינוי טבלה, המערכת מפעילה את האילוץ ומבצעת את הבדיקה. אם האילוץ יופר כתוצאה מפעולת השינוי, המערכת תדחה את הפקודה ותחזיר הודעת שגיאה מתאימה. יש לשים לב לסוגיה של הביצועים. הפעלת האילוצים השונים צורכת משאבי מחשב ויכולה להשפיע על זמני התגובה, לכן יש לבחון היטב את נושא האילוצים ברמת בסיס הנתונים.

תחביר הפקודה:



תרשים 9.27: תרשים תחביר להגדרת אילוץ ברמת בסיס הנתונים.

דוגמה : נניח שהוחלט להגביל את מספר הסטודנטים שלומדים בכל הקורסים מסוג מעבדה, כך שבכל סמסטר לא ייבחנו יותר מאשר 50 סטודנטים בקורסים מסוג מעבדה.

```
1. CREATE ASSERTION LIMIT_LAB_STUDENTS
2. CHECK (NOT EXISTS (SELECT     STUDENT_ID
3.                    FROM       GRADES  NATURAL JOIN COURSES
4.                    WHERE      TYPE = 'LAB'
5.                    GROUP BY STUDENT_ID
6.                    HAVING     COUNT (*) > 50))
```

אינדקסים (Indexes)

המודל הטבלאי אינו מתייחס לצד הפיסי של איחסון הנתונים אלא רק לצד הלוגי ולכן שפת SQL אינה כוללת התייחסות להיבטים פיסיים. למשל, הפקודה SELECT העוסקת בשליפת נתונים אינה מכילה כל התייחסות לאינדקסים. כתוצאה מכך, אחת התכונות היפות של המודל הטבלאי היא שקיום או אי קיום אינדקס אינו משפיע על נכונות הפקודות.

קיום האינדקס משפיע רק על מהירות הביצוע של הפקודה ולא על נכונותה. היעדר מנגנון האינדקס, מחייב את המערכת לבצע **סריקות מלאות של טבלאות** (Full Table Scan), דבר שהוא מאוד לא יעיל ובזבזני מבחינת משאבי המחשב. במובן זה, ניתן לומר שמנגנון האינדקס הוא מצד אחד שקוף מבחינת שפת SQL ומצד שני – חשוף מאוד מבחינת מערכת RDBMS. מנגנון זה מהווה את אחד המרכיבים העיקריים בתהליך האופטימיזציה וביכולת לבצע את הפקודות במהירות וביעילות.

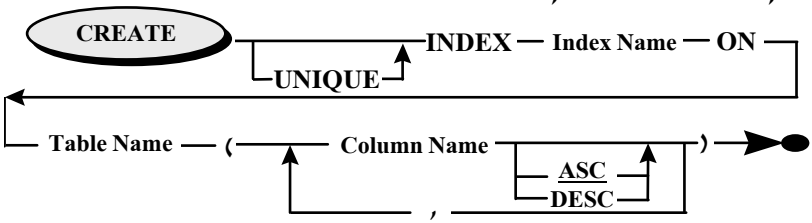
בעת הגדרת האינדקסים עבור הטבלאות השונות צריך להתחשב בכך שיחד עם השיפור בזמן עיבוד הפקודות האינדקס גם גורם לתקורה: הוא תופס מקום בדיסק ומאריך את זמן עדכון הטבלאות. כל הוספת שורה או ביטול שורה מחייבים עדכון של כל האינדקסים של אותה טבלה, בנוסף לעדכון הנתונים בטבלה עצמה. מסיבה זו יש להגדיר אינדקסים רק עבור עמודה או אוסף של עמודות שפונים אליהן בתדירות גבוהה, או שדרכן ניגשים לטבלה. לדוגמה, העמודה מספר קורס, העמודה מספר סטודנט והצירוף של מספר קורס, מספר סטודנט וסמסטר – אלו הן דוגמאות לעמודות שסביר להניח שיוגדר להן אינדקס. לעומת עמודות אלו, לא סביר להגדיר אינדקס לעמודה ציון סופי. אם יתברר שהעמודה ציון סופי משתתפת במספר רב של שאילתות, בהחלט ייתכן שכדאי לבנות אינדקס לעמודה זו כדי לזרז את הגישות.

מערכת RDBMS בונה באופן אוטומטי אינדקס עבור כל מפתח עיקרי של טבלה, מתוך הנחה שעמודה זו תשמש בתדירות גבוהה כאמצעי גישה לנתוני הטבלה. כמו כן, המערכת מייצרת אינדקס עבור עמודות שאינן מפתח עיקרי אולם הוגדר עבורן אילוץ של חד-ערכיות (Unique).

מערכות RDBMS המסחריות מסופקות כיום עם כלים שתפקידם להמליץ למפתח המערכת איזה אינדקסים כדאי לו לבנות לשיפור הביצועים, על פי פרופיל השאילתות האופייני. לדוגמה, מערכת SQL Server של מיקרוסופט מכילה את הרכיב Index Tuning Wizard, ומערכת DB2 של יבמ מכילה את הרכיב Index SmartGuide.

למרות ששפת SQL ותקן SQL מתעלמים ממנגנון האינדקס, כל המערכות המסחריות משתמשות במנגנון זה ולכן הוסיפו פקודה מיוחדת להגדרת אינדקס חדש ולביטול אינדקס קיים.

תחביר הפקודה להוספת אינדקס:



תרשים 9.28: תרשים תחביר להגדרת אינדקס חדש לטבלה.

הפרמטר UNIQUE מגדיר אם העמודה, או העמודות, חייבות להכיל ערכים חד-ערכיים או שמותר לערכים להופיע מספר פעמים. בדרך כלל אם בונים אינדקס עבור מפתח זר, הערכים חוזרים על עצמם מספר פעמים ולכן אין לרשום UNIQUE. הפרמטר ASC קובע שהאינדקס ימוין בסדר עולה, והפרמטר DESC קובע שהאינדקס ימוין סדר יורד. ברירת המחדל היא ASC.

דוגמה א' : הגדר אינדקס בשם STD_IDX לטבלת סטודנטים.

1. CREATE UNIQUE INDEX STD_IDX ON STUDENTS (STUDENT_ID)

דוגמה ב' : הגדר אינדקס בשם GRADE_IDX על מספר סטודנט ומספר קורס בטבלת ציונים.

1. CREATE UNIQUE INDEX GRADE_IDX ON GRADES (STUDENT_ID, COURSE_ID)

תחביר הפקודה לביטול אינדקס קיים:

תרשים 9.29: תרשים תחביר לביטול אינדקס.

דוגמה ג' : בטל את האינדקס STD_IDX של טבלת סטודנטים.

1. DROP INDEX STD_IDX

סיכום

בפרק זה הצגנו את **שפת הגדרת הנתונים** של SQL, המכילה אוסף פקודות המשמשות להגדרת בסיס הנתונים. בנוסף לעצם הגדרת הטבלאות והעמודות שלהן, פקודות אלו מאפשרות לקבוע את כללי האמינות והשלמות שיאכפו ישירות על ידי המערכת לניהול בסיסי נתונים. בין כללים אלה נזכיר את האפשרות לקבוע טווח ערכים עבור עמודות, את היכולת להגדיר כללי אמינות מורכבים יחסית דרך שאילתה המופיעה במשפט CHECK וגם את כללי שלמות הקשרים (Referential Integrity), שחשיבותם רבה ובכוחם להבטיח שבסיס הנתונים יהיה אמין ועקבי גם לאחר עדכונים.

בנוסף לאילוצים ברמת הטבלה הבודדת, ניתן להגדיר אילוצים וכללי אמינות ברמת בסיס הנתונים. כללים אלה מוגדרים באמצעות ההכרזות (Assertions) המוגדרות על ידי שאילתה רגילה ועל ידי היכולת לבדוק אם השאילתה מתקיימת או אינה מתקיימת.

בנוסף לטבלאות ולאילוצים, מאפשרת שפת SQL להגדיר אינדקסים לטבלאות. למרות שתקן SQL אינו מתייחס למנגנון האינדקסים ששייך לצורת המימוש, רוב המערכות המסחריות מספקות פקודות מיוחדות לבניית או לביטול אינדקסים. חשיבות האינדקסים נובעת כמובן מהיכולת שלהם להשפיע בצורה דרמטית על זמני התגובה לביצוע הפקודות השונות. מעבד SQL משתמש במנגנון האינדקס כדי לשפר את זמני התגובה.

שאלות חזרה ותרגילים

שאלות חזרה

1. הגדר שלושה אילוצים שונים. הסבר כיצד ניתן לבטא אילוצים אלה במודל הטבלאי.
2. הסבר כיצד ניתן ליישם שלמות קשרים בשפת SQL.
3. כיצד ניתן לבצע שינויים בטבלה הקיימת כבר בבסיס הנתונים? סקור את סוגי השינויים שניתן לבצע בטבלאות.
4. סקור את טיפוסי הנתונים הנתמכים על ידי שפת SQL.
5. מה המשמעות של Delete Cascades ושל Delete No Action בזמן הגדרת שלמות הקשרים בין טבלאות?
6. מדוע כדאי שמערכת RDBMS תתמוך באופן ישיר ברעיון מרחב הערכים (Domain)? איזה יתרונות יש לכך?
7. הגדר את המושגים: מרחב ערכים, מפתח עיקרי, מפתח זר, אילוץ אמינות.

תרגילים

1. כתוב את הפקודות להגדרת בסיס הנתונים המופיע בשאלה בתרגיל 1 בפרק 8.

2. נתונה הסכימה הטבלאית הבאה:

- **יצרן** (מספר יצרן, שם יצרן, עיר, כתובת, ארץ)
- **מחשב נייח** (שם מודל, מספר יצרן, מהירות מעבד, גודל דיסק, גודל זיכרון, גודל מסך, מחיר בדולרים)
- **מחשב נייד** (שם מודל, מספר יצרן, מהירות מעבד, גודל דיסק, סוג מסך, גדול זיכרון, מחיר בדולרים)
- **מדפסות** (שם מודל, מספר יצרן, סוג מדפסת, מהירות הדפסה, צורת חיבור, מחיר בדולרים)

עליך לכתוב:

א. פקודות להגדרת בסיס הנתונים.

ב. אילוץ שסוג המדפסת יכול להיות מטריצה, לייזר או הזרקת דיו בלבד.

ג. פקודה להוספת יצרן חדש שנתוניו הם:

מספר יצרן = CMPQ, שם יצרן = Compaq, עיר = Dallas,
כתובת = 102 Main Street, ארץ = USA.

3. כתוב את פקודות SQL המתאימות להגדרת בסיס הנתונים לדוגמה שבפרק 7. יש להגדיר את אילוצי האמינות הבאים:

א. סוג קורס יכול להיות אחד מבין הערכים הבאים: Class, Lab, Seminar, Workshop.

ב. מספר נקודות הזכות שקורס יכול להעניק הוא בין 1 ל-6.

ג. מועד בחינה יכול להיות A, B או C.

ד. ציון סמסטר וציון סופי יכולים להיות בין 0 ל-100.

פרק 10

טבלאות מדומות ושימושיהן (Views)

1. מבוא
2. מהי טבלה מדומה וכיצד היא נבנית
3. הגדרת טבלה מדומה (Create View)
4. טבלאות מדומות אופקיות (Horizontal Views)
5. טבלאות מדומות אנכיות (Vertical Views)
6. טבלאות מדומות משולבות – שורות ועמודות
7. טבלאות מדומות עם צירוף (Joined Views)
8. טבלאות מדומות עם פונקציות מובנות והקבצות (Grouped View)
9. תהליך שילוב שאילתות (View Resolution)
10. עדכון טבלאות מדומות (View Updatability)
11. ביטול טבלה מדומה (Drop View)
12. השוואה בין טבלה מדומה לבין תת-סכימה
13. סיכום
14. שאלות חזרה ותרגילים

מבוא

פרק זה מציג את אחד הנושאים הייחודיים של שפת SQL, את **הטבלאות המדומות** (View Table, ובקיצור – View). בפרקים קודמים הצגנו את השפה לטיפול בנתונים ואת השפה להגדרת נתונים והנחנו שבסיס הנתונים מורכב מאוסף של טבלאות. מקובל לכנות טבלאות אלו גם בשם **טבלאות בסיסיות** (Base Tables) או **טבלאות המקור** (Source Tables). בפרק זה נציג סוג נוסף של טבלה המנוהל בבסיס הנתונים, **טבלה מדומה** (View).

הטבלה המדומה היא טבלה שרק ההגדרה שלה, במבנה של שאילתת SQL רגילה, נשמרת בבסיס הנתונים. הטבלה ממומשת על ידי הפעלת השאילתה רק בעת הצורך ולכן היא טבלה מדומה ולא ממשית. לטבלאות המדומות מספר רב של יתרונות – יכולת הצגה פשוטה יותר של הנתונים למשתמשים שונים, יכולת להתאים למשתמשים שונים נקודות מבט ייחודיות לצרכים שונים, יכולת לספק רמה גבוהה של בקרת גישות ואבטחת מידע ועוד. כל היתרונות האלה הפכו את הטבלאות המדומות לאחד המנגנונים הנפוצים בסביבה של בסיסי נתונים טבלאיים.

בפרק זה

- נציג את הפקודה להגדרת טבלה מדומה, CREATE VIEW ומיגוון רחב של דוגמאות להגדרה ושימוש בטבלאות מדומות.
- נלמד לעדכן בסיס הנתונים באמצעות טבלאות מדומות. כפי שנראה קיימים מצבים מסוימים בהם ניתן לעדכן את בסיס הנתונים באמצעות טבלאות מדומות, אולם רוב הטבלאות המדומות אינו תומך בעדכון.
- נבחן את סוגיית עדכון בסיס הנתונים באמצעות טבלאות מדומות, את האילוצים והאפשרויות.
- נציג יתרונות וחסרונות של שימוש בטבלאות מדומות.

מהי טבלה מדומה וכיצד היא נבנית

טבלה מדומה **View**	***טבלה מדומה* הינה טבלה שאינה קיימת באופן ממשי בבסיס הנתונים אלא היא תוצאה דינמית של הפקודה SELECT.**

הטבלה המדומה נוצרת רק בעת הפנייה אליה, וכוללת נתונים שנגזרים מטבלת בסיס אחת או יותר. מאחר והטבלה המדומה אינה מכילה נתונים, מקובל לומר שטבלה מדומה היא דינמית להבדיל מטבלאות הבסיס שהן טבלאות סטטיות המאוכלסות בנתונים. כל פנייה אל הטבלה המדומה יכולה ליצור טבלה מדומה שונה, בהתאם לתוכן הרגעי של טבלאות הבסיס מהן נגזרים הנתונים. בסיס הנתונים מאחסן את נתוני טבלאות הבסיס ורק את ההגדרות של הטבלאות המדומות. מנקודת המבט של המשתמש, טבלה מדומה

נראית בדיוק כמו טבלה אמיתית והוא יכול לבצע עליה שאילתות שונות, באופן דומה לצורת בניית שאילתות על טבלאות הבסיס.

דרך נוספת להתייחס אל טבלה מדומה היא כאל פילטר המאפשר הצגה (ולעיתים גם עדכון) של שורות או של עמודות מסוימות מטבלאות בסיסיות. זו הסיבה לשם **View**, כלומר **נקודת מבט על הנתונים**, או **תצוגת נתונים**. ניתן לומר שטבלאות מדומות מאפשרות "תפירת" נקודות מבט ייחודיות לכל משתמש, תוך פישוט הגישה לבסיס הנתונים.

נראה עכשיו כיצד הטבלאות המדומות נבנות. נניח לרגע שבנינו טבלה שהיא התוצאה של השאילתה הבאה – הצג את שמות הסטודנטים ואת כתובתם עבור כל הסטודנטים שלמדו בקורס C-200. שאילתה זו מבצעת Join בין שתי טבלאות בסיס, טבלת סטודנטים וטבלת קורסים.

```
1. SELECT   NAME, CITY
2. FROM     STUDENTS S, GRADES G
3. WHERE    COURSE_ID = 'C-200' AND
4.                S.STUDENT_ID = G.STUDENT_ID
```

PROG_STUDENTS

NAME	CITY
Moshe	Haifa
Dan	Tel Aviv
David	Tel Aviv

נניח שלטבלת התוצאה נתנו שם PROG_STUDENTS ואנו מאפשרים לראש המחלקה למדעי המחשב לגשת אליה. אם הוא ירצה לראות את **כל** הסטודנטים שלומדים תכנות ועיר מגוריהם היא תל אביב, הוא יבנה את השאילתה הבאה:

```
1. SELECT  *
2. FROM    PROG_STUDENTS
3. WHERE   CITY = 'Tel Aviv'
```

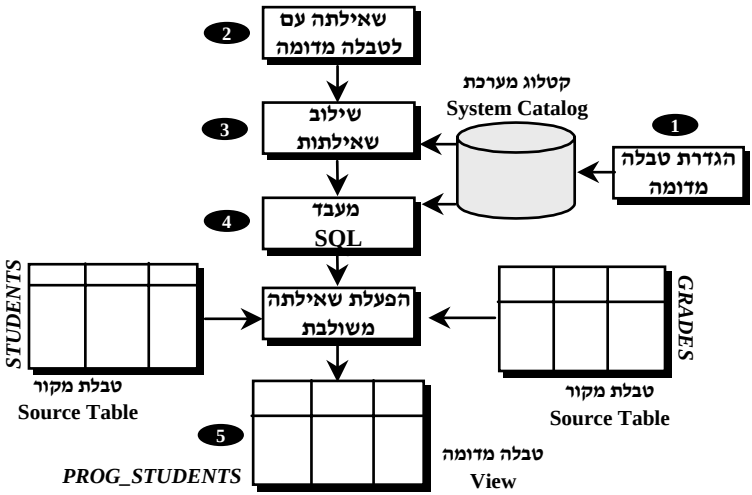
NAME	CITY
Dan	Tel Aviv
David	Tel Aviv

מנקודת המבט שלו, טבלת PROG_STUDENTS היא טבלה רגילה לכל דבר והוא יכול להפעיל עליה את השאילתות הדרושות לו. בפועל, מאחורי הקלעים, מתבצעת שאילתה המשלבת את השאילתה לבניית הטבלה PROG_STUDENTS ואת השאילתה של ראש המחלקה. השאילתה שבאמת מבוצעת תהיה:

```
1. SELECT  NAME, CITY
2. FROM    STUDENTS, GRADES
3. WHERE   COURSE_ID = 'C-200' AND
4.                STUDENTS.STUDENT_ID = GRADES.STUDENT_ID
5.                AND CITY = 'Tel Aviv'
```

שלבים בבניית טבלה מדומה

נסקור את השלבים במימוש הטבלה המדומה. מקובל להתייחס לארבעת השלבים הבאים בתהליך המימוש: שלב הגדרת הטבלה המדומה, שלב הפעלת השאילתה של המשתמש, שלב שילוב השאילתות ושלב הביצוע.



תרשים 10.1: שלבים בבניית טבלה מדומה.

תרשים 10.1 מתאר באופן סכמתי את שלושת השלבים האלה. נפרט כל אחד מהם בהמשך.

- **שלב 1 – הגדרת הטבלה המדומה**: בשלב הראשון מגדירים את הטבלה המדומה, באמצעות הפקודה המיוחדת CREATE VIEW. פקודה זו מכילה שאילתה המשמשת לבניית הטבלה המדומה. לכל טבלה מדומה ניתן שם חד-ערכי, בדומה לשם החד-ערכי שניתן לכל טבלת בסיס. שם זה ישמש בהמשך לצורך פנייה אל הטבלה המדומה. לאחר בדיקת תקינות ההגדרה, ההגדרה מוכנסת אל הקטלוג של מערכת RDBMS. נשים לב שמעבר לאחסון הגדרת הטבלה המדומה בקטלוג, בסיס הנתונים לא מבצע שום פעולה נוספת. ההגדרה אינה עוברת אפילו שלב של אופטימיזציה ובניית תוכנית גישה לבסיס הנתונים.

- **שלב 2 – הפעלת שאילתת המשתמש:** הגישה אל הטבלה המדומה מתרחשת כאשר משתמש (או תוכנית יישום) משגר פקודת SQL שבמשפט FROM שלה מופיע שם של טבלה מדומה. מנקודת מבט המשתמש, אין כל הבדל בין שאילתה הפונה לטבלאות בסיס ובין שאילתה הפונה לטבלאות מדומות. אגב, מותר לפנות באותה שאילתה גם לטבלאות בסיס וגם לטבלאות מדומות.

- **שלב 3 – שילוב שאילתות (View Resolution)**: מערכת RDBMS שולפת את הגדרת הטבלה המדומה מתוך הקטלוג ומתחילה בביצוע תהליך של שילוב השאילתה המגדירה את הטבלה המדומה והשאילתה של המשתמש. שתי השאילתות משולבות על ידי הוספת מילת קשר AND ביניהן. תהליך זה נקרא View Resolution. בגמר ביצוע

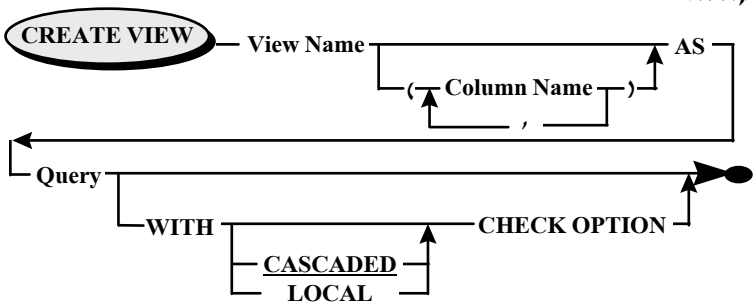
תהליך השילוב, מתקבלת שאילתה חדשה. תהליך השילוב בין הגדרת הטבלה המדומה ושאילתת המשתמש, מתבצע באופן שקוף מאחורי הקלעים ולכן, מנקודת המבט של המשתמש, נדמה לו שהוא הפעיל שאילתה על טבלת בסיס. שלב זה מפורט בהמשך.

- **שלב 4 – בניית תוכנית גישה**: עם גמר תהליך השילוב, נוצרת שאילתה חדשה. שאילתה זו נשלחת אל מעבד SQL כשאילתה רגילה. השאילתה עוברת את כל השלבים ששאילתה צריכה לעבור, כולל תהליך של אופטימיזציה והכנת תוכנית הגישה לבסיס הנתונים.
- **שלב 5 – ביצוע השאילתה המשולבת**: מייד עם גמר הכנת תוכנית הגישה, היא מופעלת ומתחילה לייצר את השורות של הטבלה התוצאתית. מאחר ותהליך בניית השאילתה המשולבת מתבצע רק בעת שיגור השאילתה החדשה, הטבלה הנוצרת היא טבלה דינמית המבוססת על התוכן הרגעי של טבלאות המקור באותו זמן.

הגדרת טבלה מדומה (Create View)

הפקודה CREATE VIEW משמשת להגדרת טבלה מדומה חדשה. ההגדרה נבדקת מבחינת תקינותה ונרשמת בקטלוג המערכת. אין מגבלה למספר הטבלאות המדומות שניתן להגדיר.

תחביר הפקודה:



תרשים 10.2: תרשים תחביר להגדרת טבלה מדומה.

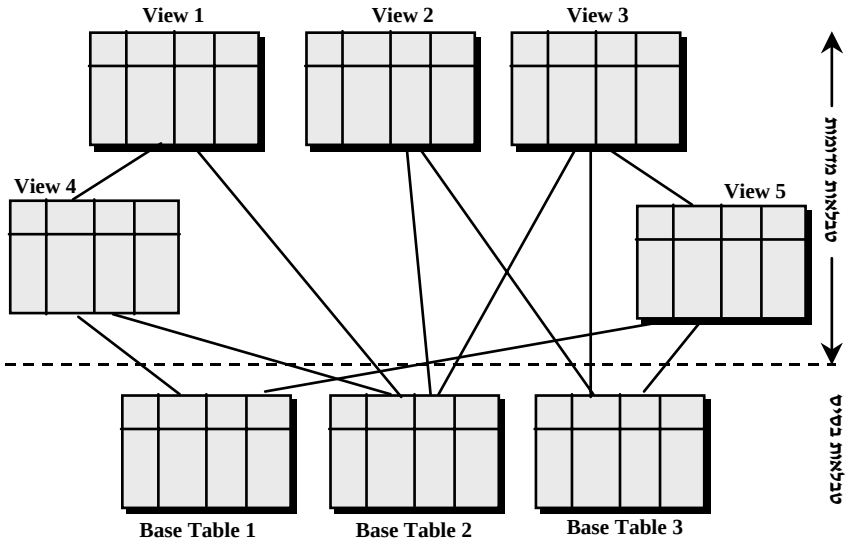
לכל טבלה מדומה יש שם חד-ערכי, View Name, המשמש לאחר מכן להתייחסות אליה. שמות העמודות בטבלה המדומה יכולים להיות זהים לשמות העמודות בטבלאות הבסיס, או שניתן לקבוע שמות חדשים תוך שימוש בפרמטר Column Name. אם קובעים שמות חדשים לעמודות של הטבלה המדומה, מספר העמודות עבורם קובעים שם חדש חייב להיות זהה למספר העמודות הנשלפות על ידי השאילתה המשמשת לבניית הטבלה המדומה.

השאילתה המשמשת לבניית הטבלה המדומה, View Query, צריכה להיות שאילתת SQL חוקית עם מספר מגבלות:

א. השאילתה אינה יכולה להגדיר סדר מיון על ידי המשפט ORDER BY;

ב. השאילתה אינה יכולה לבצע איחוד, כלומר אינה יכולה להכיל משפט UNION.

במשפט FROM של השאילתה ניתן להתייחס הן לטבלאות מקור והן לטבלאות מדומות אחרות, כלומר ניתן לבנות טבלה מדומה המבוססת על טבלה מדומה אחרת. תרשים 10.3 מדגים מצב שבו טבלה מדומה 2 מבוססת רק על טבלאות הבסיס 2 ו-3, ואילו טבלה מדומה 1 מבוססת על טבלת הבסיס 2 ועל טבלה מדומה 4. למצב זה אנו קוראים **היררכיה** של טבלאות מדומות.



תרשים 10.3: היררכיה של טבלאות מדומות.

הפרמטר WITH CHECK OPTION בפקודה מציין שהטבלה המדומה היא בת-עדכון. תפקידו למנוע עדכון של טבלה מדומה עם תוכן שאינו עונה על התנאים המופיעים בשאילתה המשמשת להגדרתה. היעדר פרמטר זה יאפשר עדכון של טבלה מדומה גם אם העדכון אינו מקיים את תנאי השאילתה. מכיון שטבלה מדומה יכולה להיות מבוססת על טבלאות מדומות אחרות, הפרמטר WITH LOCAL CHECK OPTION יגרום לבדיקת העדכון רק מול התנאי של הטבלה המדומה המתעדכנת, והפרמטר WITH CASCADED CHECK OPTION ידרוש שהעדכון יקיים את התנאים של כל הטבלאות המדומות בהיררכיה. אם לא מצוין אחרת, זו ברירת המחדל.

טבלאות מדומות אופקיות (Horizontal Views)

טבלה מדומה אופקית היא טבלה המבוססת על שאילתה הבוחרת את כל העמודות מטבלאות המקור ומכילה תנאי לבחירת שורות מתוך טבלת המקור. טבלה מדומה מסוג זה מאפשרת לנו לתפור לכל משתמש טבלה מדומה משלו, טבלה המכילה רק את השורות המענינות אותו או שהוא מורשה לגשת אליהן.

דוגמה א' : יש להגדיר טבלה מדומה בשם HAIFA_STUDENTS המכילה את כל העמודות של טבלת סטודנטים ואת השורות של סטודנטים שעיר מגוריהם חיפה בלבד.

```
1. CREATE VIEW HAIFA_STUDENTS AS
2.        SELECT   *
3.        FROM     STUDENTS
4.        WHERE    CITY = 'Haifa'
```

אם נרצה לראות את הטבלה המדומה נפעיל עליה שאילתה רגילה.

```
1. SELECT *
2. FROM    HAIFA_STUDENTS
```

STUDENT_ID	NAME	CITY
105	Moshe	Haifa
110	Ran	Haifa
245	Yoel	Haifa

דוגמה ב' : כמו בדוגמה א', אבל יש לקבוע את שם העמודה הראשונה כ- STUDENT_NUMBER.

```
1. CREATE VIEW HAIFA_STUDENTS (STUDENT_NUMBER, NAME, CITY) AS
2.        SELECT *
3.        FROM    STUDENTS
4.        WHERE CITY = 'Haifa'
```

נשים לב שמספר השמות המופיעים בהגדרת הטבלה המדומה זהה למספר העמודות בטבלת סטודנטים. אם נרצה לראות את הטבלה המדומה נפעיל עליה שאילתה רגילה.

```
1. SELECT *
2. FROM    HAIFA_STUDENTS
```

STUDENT_NUMBER	NAME	CITY
105	Moshe	Haifa
110	Ran	Haifa
245	Yoel	Haifa

דוגמה ג' : יש להגדיר טבלה מדומה חדשה המבוססת על הטבלה המדומה HAIFA_STUDENTS_100 וכוללת רק סטודנטים שמספרי הזיהוי שלהם בין 100 ל-199.

```
1. CREATE VIEW HAIFA_STUDENTS_100 AS
2.        SELECT   *
3.        FROM     HAIFA_STUDENTS
4.        WHERE    STUDENT_ID BETWEEN 100 AND 199
```

למעשה, יכולנו לבנות טבלה מדומה זו באופן ישיר, אולם המטרה היתה להציג את היכולת לבנות היררכיה של טבלאות מדומות. המסקנה היא, שטבלה מדומה יכולה להתייחס לטבלה מדומה אחרת.

טבלאות מדומות אנכיות (Vertical Views)

טבלה מדומה אנכית היא טבלה המבוססת על שאילתה שבוחרת אוסף חלקי של עמודות מתוך טבלת המקור ובוחרת את כל השורות של טבלת המקור. טבלה מדומה מסוג זה מאפשרת לנו לתפור לכל משתמש טבלה מדומה משלו, טבלה המכילה רק את העמודות המעניינות אותו או שהוא מורשה לגשת אליהן.

דוגמה : יש להגדיר טבלה מדומה בשם STUDENT_1 שתכיל רק את העמודות של מספר הסטודנט ושם הסטודנט מתוך טבלת סטודנטים.

```
1. CREATE VIEW STUDENT_1 AS
2.        SELECT   STUDENT_ID, NAME
3.        FROM     STUDENTS
```

אם נרצה לראות את הטבלה המדומה, נפעיל עליה שאילתה רגילה.

```
1. SELECT *
2. FROM    STUDENT_1
```

STUDENT_ID	NAME
105	Moshe
210	Dan
107	Eyal
110	Ran
245	Yoel
240	Ayelet
200	David
310	Tova

טבלאות מדומות
משולבות – שורות ועמודות

טבלה מדומה משולבת היא טבלה המבוססת על שאילתה שבוחרת אוסף כלשהו של עמודות מתוך טבלאות המקור וכן תנאי לבחירת השורות מתוך טבלאות המקור.

דוגמה א': יש להגדיר טבלה מדומה בשם HAIFA_STD_NAMES המכילה את מספר הסטודנט ואת שם הסטודנט עבור כל הסטודנטים הגרים בחיפה. מאחר וממילא בטבלה המדומה העמודה של העיר תכיל תמיד את הערך חיפה, אין הגיון להציג אותה.

```
1. CREATE VIEW HAIFA_STD_NAMES AS
2.         SELECT   STUDENT_ID, NAME
3.         FROM     STUDENTS
4.         WHERE    CITY = 'Haifa'
```

אם נפעיל עכשיו את השאילתה הקודמת נקבל:

```
1. SELECT *
2. FROM    HAIFA_STD_NAMES
```

STUDENT_ID	NAME
105	Moshe
110	Ran
245	Yoel

דוגמה ב' : יש להגדיר טבלה מדומה בשם HAIFA_STD_NAMES1, אשר דומה לדוגמה הקודמת, אך לקבוע שמות חדשים לעמודות של הטבלה המדומה.

```
1. CREATE VIEW HAIFA_STD_NAMES1
2.                (STUDENT_NUMBER, STUDENT_NAME) AS
3.                  SELECT   STUDENT_ID, NAME
4.                  FROM     STUDENTS
5.                  WHERE    CITY = 'Haifa'
```

אם נפעיל עכשיו את השאילתה נקבל:

```
1. SELECT *
2. FROM    HAIFA_STD_NAMES1
```

STUDENT_NUMBER	STUDENT_NAME
105	Moshe
110	Ran
245	Yoel

נשים לב ששמות העמודות בטבלה המדומה שונים משמות העמודות בטבלת הבסיס.

דוגמה ד' : יש להגדיר טבלה מדומה בשם BEST_GRADES שתציג את מספר הסטודנט והציון עבור כל הסטודנטים שקיבלו במבחן הסמסטר ציון גבוה מהציון הגבוה ביותר שסטודנט כלשהו קיבל במבחן הסופי.

```
1. CREATE VIEW BEST_GRADES AS
2.        SELECT STUDENT_ID, GRADE
3.        FROM   GRADES
4.        WHERE GRADE_SEM >
5.                (SELECT MAX  (GRADE)
6.                 FROM    GRADES)
```

זו דוגמה להגדרת טבלה מדומה המבוססת על שאילתה עם תת-שאילתה. אם נפעיל שאילתה להצגת הטבלה המדומה נקבל:

```
1. SELECT *
2. FROM    BEST_GRADES
```

STUDENT_ID	GRADE
245	85
310	65

נשים לב עד כמה פשוטה השאילתה מנקודת מבט המשתמש. הוא אינו יכול להבחין שמאחורי שאילתה פשוטה זו מסתתרת שאילתה מורכבת.

טבלאות מדומות עם צירוף (Joined Views)

טבלה מדומה עם צירוף מבוססת על שאילתה המכילה צירוף בין מספר טבלאות. היתרון הגדול של טבלאות מדומות המכילות צירוף הוא שמנקודת המבט של המשתמש, נדמה לו שהוא עובד עם טבלה אחת פשוטה ועליה הוא יכול להגדיר את תנאי השליפה שלו, מבלי להיות מודע כלל לעובדה שהשאילתה שלו דורשת ביצוע של מספר צירופים. טבלאות אלו נפוצות מאוד במקרים בהם יש צורך לאפשר למשתמשי קצה לגשת לבסיס הנתונים ולבצע שאילתות שונות. על ידי הגדרת הטבלה המדומה המכילה בחירת שורות, בחירת עמודות וצירוף בין טבלאות, ניתן לדמות בסיס נתונים פשוט מבחינת המשתמש למרות שמאחורי הטבלה המדומה מסתתרת הגדרה מורכבת למדי.

דוגמה א' : יש להגדיר טבלה מדומה בשם CS_GRADES שתכיל את שמות הסטודנטים וכל נתוני הציונים של הסטודנטים, עבור קורסים של המחלקה למדעי המחשב בלבד.

```
1. CREATE VIEW CS_GRADES AS
2.        SELECT  S.NAME, S.STUDENT_ID, C.COURSE_NAME,
3.                C.COURSE_ID, G.SEMESTER, G.TERM,
4.                G.GRADE, G.GRADE_SEM
5.        FROM    STUDENTS S, COURSES C, GRADES G
6.        WHERE   G.STUDENT_ID = S.STUDENT_ID AND
7.                G.COURSE_ID = C.COURSE_ID AND
8.                C.DEPARTMENT_ID = 'CS'
```

נבנה עכשיו שאילתה שתציג את כל השורות של הטבלה המדומה.

```
1. SELECT   *
2. FROM     CS_GRADES
```

NAME	STUDENT_ID	COURSE_NAME	COURSE_ID	SEMESTER	TERM	GRADE	GRADE_SEM
Moshe	105	Data Base	C-55	SUM1998	A	58	70
Moshe	105	Programming	C-200	AUT1999	A	90	85
Dan	210	Programming	C-200	AUT1999	A	85	80
David	200	Programming	C-200	AUT1999	B	78	50

נשים לב כמה פשוטה השאילתה הזאת מנקודת מבט המשתמש, למרות שהביצוע שלה דורש צירוף בין שלוש טבלאות בסיס. נראה שאילתה נוספת על טבלה CS_GRADES. בשאילתה זו נבקש להציג את רשימת שמות הסטודנטים שלמדו בסתיו *1999* ונבקש לקבל את הרשימה ממוינת לפי שם סטודנט.

```
1. SELECT     NAME
2. FROM       CS_STUDENTS
3. WHERE      SEMESTER = 'AUT1999'
4. ORDER BY   NAME
```

NAME
Dan
David
Moshe

דוגמה ב': הצג את מספר הסטודנט, מספר הקורס, מועד הבחינה והציון עבור כל הסטודנטים המופיעים בטבלה המדומה HAIFA_STUDENTS, כפי שהוגדרה מקודם.

```
1. SELECT STUDENT_NUMBER,
2.        COURSE_ID, TERM, GRADE
3. FROM   HAIFA_STUDENTS H, GRADES G
4. WHERE H.STUDENT_NUMBER = G.STUDENT_ID
5.        AND SEMESTER = 'AUT1999'
```

STUDENT_NUMBER	COURSE_ID	TERM	GRADE
105	C-200	A	90
245	M-100	A	90
245	B-10	A	80

זו דוגמה לשאילתה רגילה שבה מופיעה במשפט FROM גם טבלה מדומה וגם טבלת בסיס ומבוצע ביניהן צירוף.

טבלאות מדומות עם פונקציות מובנות והקבצות (Grouped View)

טבלה מדומה יכולה להכיל גם עמודות שהן תוצאה של פונקציה מובנית כלשהי. יש לשים לב שלא ניתן להשתמש בטבלאות מדומות המכילות פונקציות מובנות לצורך הגדרת היררכיה של טבלאות מדומות, כלומר לא ניתן להגדיר טבלה מדומה שבמשפט FROM שלה מופיעה טבלה מדומה המכילה פונקציה מובנית.

דוגמה : בנה טבלה מדומה בשם TOTAL_POINTS המכילה את סה"כ הנקודות שצבר כל סטודנט. צבירת נקודות תבוצע רק אם הסטודנט קיבל ציון מעל 60 בקורס. הנקודות המצטברות מחושבות על ידי המכפלה של הציון במספר נקודות הזכות שהקורס מעניק מחולק ל- 100.

```
1.  CREATE VIEW  CUMMULATIVE_POINTS
2.                    (NAME, ID, CUMM_POINTS) AS
3.        SELECT   S.NAME, G.STUDENT_ID,
4.                 SUM (POINTS * GRADE / 100),
5.        FROM     STUDENTS S, COURSES C, GRADES G
6.        WHERE    G.STUDENT_ID = S.STUDENT_ID AND
7.                 G.COURSE_ID = C.COURSE_ID AND
8.                 G.GRADE >= 60
9.        GROUP BY  NAME, STUDENT_ID
```

במקרה זה, הטבלה המדומה מכילה עמודה שכלל אינה קיימת בבסיס הנתונים. אם נרצה לראות את תוכן הטבלה עלינו לבנות את השאילתה הבאה :

```
1.  SELECT *
2.  FROM    CUMMULATIVE_POINTS
```

NAME	ID	CUMM_POINTS
Moshe	105	7.95
David	200	7.22
Dan	210	6.88
Yoel	245	2.55
Tova	310	1.85

תהליך שילוב השאילתות (View Resolution)

הגדרת הטבלה המדומה הינה הגדרה סטטית המאוחסנת בקטלוג המערכת, ולמעשה אינה גורמת לשום פעולה נוספת. הטבלה המדומה נבנית רק אם שאילתה כלשהי מכילה התייחסות אליה ובמשפט FROM של השאילתה מופיע שם הטבלה המדומה. בעת בניית השאילתה ולפני העברתה לאופטימיזציה, משולבת ההגדרה של הטבלה המדומה בתוך השאילתה של המשתמש. תהליך זה קרוי **שילוב שאילתות** (View Resolution).

הבה נראה כיצד מתבצע שילוב זה:

נתבסס על הטבלה המדומה HAIFA_STUDENTS, שנבנתה על ידי פקודה זו:

```
1. CREATE VIEW HAIFA_STUDENTS AS
2.        SELECT *
3.        FROM   STUDENTS
4.        WHERE CITY = 'Haifa'
```

עכשיו נניח שהפעלנו את השאילתה שהופיעה בדוגמה ב' בסעיף המסביר את הטבלאות המדומות המכילות צירוף.

```
1. SELECT   STUDENT_NUMBER, COURSE_ID, TERM, GRADE
2. FROM     HAIFA_STUDENTS H, GRADES G
3. WHERE    H.STUDENT_NUMBER = G.STUDENT_ID AND
4.          SEMESTER = 'AUT1999'
```

נעקוב לרגע אחר תהליך השילוב, תהליך המורכב ממספר שלבים:

- שמות העמודות במשפט SELECT מוחלפות בשמות של העמודות בטבלת הבסיס תוך הוספת AS עם שם העמודה מהטבלה המדומה. דבר זה יבטיח שבטבלת התוצאה תתקבל עמודה עם השם כפי שנקבע בעת שהטבלה המדומה הוגדרה.

שם בטבלת הבסיס — שם בטבלה מדומה

SELECT STUDENT_ID AS STUDENT_NUMBER , COURSE_ID, TERM, GRADE

- שמות הטבלאות המדומות במשפט FROM מוחלפים בשמות של טבלאות בסיס. בדוגמה שלנו, שם הטבלה המדומה HAIFA_STUDENTS מוחלף בשם הטבלה STUDENTS.

FROM STUDENTS H, GRADES G

- במשפט WHERE מתווספים כל התנאים של השאילתה המגדירה את הטבלה המדומה אל השאילתה של המשתמש תוך שימוש באופרטור AND. במידת הצורך מוחלפים שמות של עמודות בטבלה המדומה עם שמות של טבלאות הבסיס.

WHERE H.STUDENT_ID = G.STUDENT_ID AND SEMESTER = 'AUT1999'
AND H.CITY = 'Haifa'

אופרטור השילוב — תנאי להגדרת הטבלה המדומה — תנאי של השאילתה

- אם בשאילתה של המשתמש מופיעים גם משפטי GROUP BY ו-HAVING, הם יועתקו אל השאילתה החדשה.
- אם בשאילתה של המשתמש מופיע משפט ORDER BY להגדרת סדר המיון, הוא יועתק אל השאילתה החדשה.

בסופו של תהליך השילוב מתקבלת השאילתה החדשה הזו:

```
1.  SELECT   STUDENT_ID AS STUDENT_NUMBER,
2.           COURSE_ID, TERM, GRADE
3.  FROM     STUDENTS H, GRADES G
4.  WHERE    H.STUDENT_ID = G.STUDENT_ID AND
5.           SEMESTER = 'AUT1999' AND
6.           H.CITY = 'Haifa'
```

שאילתה חדשה זו, היא שתעבור את ההידור והמיטוב (אופטימיזציה) ולא השאילתה שנכתבה על ידי המשתמש. מאחר וכל תהליך שילוב השאילתה מתבצע מאחורי הקלעים וללא ידיעת המשתמש, הוא מקבל את התחושה כאילו פעל מול טבלאות אמיתיות.

עדכון טבלאות מדומות (View Updatability)

כפי שראינו, הטבלה המדומה היא למעשה **שאילתה** שהגדרתה נשמרת מראש בקטלוג המערכת ובעת הפנייה אליה, השאילתה של המשתמש משולבת יחד עם ההגדרה לקבלת שאילתה חדשה. שאילתה חדשה זו מופעלת ובונה את הטבלה התוצאתית המוצגת למשתמש. כל עוד מדובר בשליפת נתונים שיטה זו עובדת היטב. כאשר מבקשים גם לאפשר למשתמש לעדכן את בסיס הנתונים דרך הטבלאות המדומות, נתקלים במספר בעיות.

מגבלות בעדכון טבלה מדומה

מספר מגבלות נובעות מכך שטבלה מדומה הינה למעשה נגזרת של טבלאות בסיסיות. ראינו שבטבלה המדומה לא מופיעות בהכרח כל העמודות ולפעמים העמודות המופיעות בטבלה המדומה כלל אינן מופיעות בטבלאות הבסיס. לפיכך נוצרות בעיות המביאות לידי כך שלא תמיד ניתן להבטיח את יכולת העדכון דרך הטבלאות המדומות. נציג מספר דוגמאות של טבלאות מדומות ובאמצעותן נסקור את הבעיות השונות.

דוגמה א': ההגדרה הבאה מציגה טבלה מדומה אופקית, כלומר כל השורות מופיעות אבל לא כל העמודות.

```
1.  CREATE VIEW STUDENT_1 AS
2.          SELECT   STUDENT_ID, NAME
3.          FROM     STUDENTS
```

במקרה זה הטבלה המדומה מכילה רק שתי עמודות מתוך הטבלה הבסיסית. מאחר והעמודה המכילה את המפתח העיקרי מופיעה בטבלה המדומה, ניתן לבצע עדכון דרך הטבלה המדומה. למשל אם נרשום:

```
1.  INSERT INTO STUDENT_1  (STUDENT_ID, NAME)
2.          ('720', 'Doni')
```

פקודה זו תפתח שורה חדשה בטבלת הסטודנטים. מאחר והעיר אינה מופיעה, העמודה הזאת תקבל את הערך Null בשורה החדשה. אם בהגדרת טבלת המקור העמודה עיר מוגדרת כ- NOT NULL, פקודת ההוספה תדחה, למרות שמבחינת המשתמש הוא הכניס ערכים חוקיים. מצב זה מבלבל מאוד מבחינת המשתמש, מאחר והוא כלל אינו מכיר את העמודה עיר. מכאן נובע, שכדי שטבלה מדומה תהיה בת עדכון היא חייבת להכיל לפחות את המפתח העיקרי ואת כל העמודות שמוגדרות כ- NOT NULL.

דוגמה ב': ההגדרה הבאה מציגה טבלה מדומה אנכית, כלומר כל העמודות מופיעות אבל לא כל השורות.

```
1. CREATE VIEW HAIFA_STUDENTS AS
2.        SELECT   *
3.        FROM     STUDENTS
4.        WHERE    CITY = 'Haifa'
```

במקרה זה הטבלה מכילה את כל העמודות של טבלת סטודנטים ואת כל השורות שבהן הערך של העמודה עיר הוא חיפה. אם המשתמש רוצה להוסיף סטודנט חדש הוא יכתוב:

```
1. INSERT INTO HAIFA_STUDENTS
2.        ('720', 'Doni', 'Haifa')
```

פקודה זו תקינה לכן העדכון יבוצע. נבדוק לרגע מה היה קורה לו המשתמש היה כותב את פקודת ההוספה הבאה:

```
1. INSERT INTO HAIFA_STUDENTS
2.        ('720', 'Doni', 'Jerusalem')
```

כאן מתעוררת שאלה מעניינת. הטבלה המדומה מכילה רק שורות של 'חיפה' והמשתמש מנסה להוסיף שורה עם הערך 'ירושלים'. כאן נכנס לפעולה הפרמטר WITH CHECK OPTION של הפקודה CREATE VIEW. אם פרמטר זה אינו מופיע, השורה החדשה תוכנס אל הטבלה הבסיסית. אם המשתמש יבצע שאילתה מייד לאחר הוספת השורה, הוא לא יראה אותה, כי היא אינה מקיימת את התנאי של השאילתה המשמשת להגדרת הטבלה המדומה. כדי למנוע מצב זה, יש להגדיר את הטבלה המדומה בצורה זו:

```
1. CREATE VIEW HAIFA_STUDENTS AS
2.        SELECT   *
3.        FROM     STUDENTS
4.        WHERE    CITY = 'Haifa'
5. WITH CHECK OPTION
```

הפעם, תתבצע בדיקה של השורה החדשה ורק אם היא מקיימת את התנאי של השאילתה היא תוכנס אל הטבלה. צורת עבודה זו זהה למקרה של עדכון שורה, כלומר אם משתמש היה מנסה לבצע עדכון של שורה:

```
1. UPDATE HAIFA_STUDENTS
2.        SET CITY = 'Jerusalem'
3.        WHERE STUDENT_ID = 105
```

אם הפרמטר WITH CHECK OPTION אינו קיים קיים, עדכון זה יבוצע ולא יידחה.

דוגמה ג' : ההגדרה הבאה מציגה את הטבלה המדומה שמכילה עמודות הנובעות מפונקציות מובנות.

```
1.  CREATE VIEW CUMMULATIVE_POINTS
2.                  (NAME, ID, CUMM_POINTS) AS
3.      SELECT  S.NAME, G.STUDENT_ID,
4.              SUM (POINTS * GRADE / 100),
5.      FROM    STUDENTS S, COURSES C, GRADES G
6.      WHERE   G.STUDENT_ID = S.STUDENT_ID AND
7.              G.COURSE_ID = C.COURSE_ID AND
8.              G.GRADE >= 60
9.      GROUP BY  NAME, STUDENT_ID
```

נניח שמשתמש ינסה לרשום את הפקודה הזו :

```
1.  UPDATE CUMMULATIVE_POINTS
2.      SET CUMM_POINTS = 7.88
3.      WHERE STUDENT_ID = 105
```

במקרה זה לא ניתן לבצע את העדכון כלל, מאחר והעמודה CUMM_POINTS אינה קיימת אלא היא תוצאה של חישוב כלשהו על עמודות אחרות.

דוגמה ד' : ההגדרה הבאה מציגה את הטבלה המדומה HAIFA_STD_100 המבוססת על טבלה מדומה אחרת.

```
1.  CREATE VIEW HAIFA_STUDENTS_100 AS
2.      SELECT  *
3.      FROM    HAIFA_STUDENTS
4.      WHERE   STUDENT_ID BETWEEN 100 AND 199
5.  WITH LOCAL CHECK OPTION
```

נשים לב שבפקודת הגדרת הטבלה המדומה מופיעה האופציה WITH LOCAL CHECK OPTION. נניח שמשתמש ינסה לרשום את הפקודה הבאה :

```
1.  UPDATE HAIFA_STUDENTS_100
2.      SET CITY = 'Jerusalem'
3.      WHERE STUDENT_ID = 105
```

הפקודה מקיימת את התנאי המופיעה בהגדרת הטבלה המדומה : מספר סטודנט בין 100 ל-200 ולכן העדכון יבוצע. מאחר וקיימת כאן היררכיה של טבלאות מדומות, הפקודה הזאת הפרה למעשה את התנאי של הטבלה המדומה הראשונה, HAIFA_STUDENTS ולכן אם עכשיו המשתמש יבקש לראות את הטבלה לאחר העדכון הוא לא יראה אותה, מאחר והיא שולפת רק סטודנטים הגרים בחיפה. כדי למנוע מצב זה, יש להגדיר את הטבלה המדומה בצורה הבאה :

```
1.  CREATE VIEW HAIFA_STUDENTS_100 AS
2.      SELECT  *
3.      FROM    HAIFA_STUDENTS
4.      WHERE   STUDENT_ID BETWEEN 100 AND 199
5.  WITH CASCADED CHECK OPTION
```

הפעם הוגדר הפרמטר WITH CASCADED CHECK OPTION, ולכן תתבצע בדיקה שהעדכון מקיים את התנאים של כל ההיררכיה של הטבלאות המדומות. במקרה זה העדכון יידחה. זוהי אגב ברירת המחדל ולכן אם מופיע WITH CHECK OPTION לא תתבצע בדיקה לעומק כל ההיררכיה.

כללים לעדכון טבלה מדומה

ניתן להביא עוד מספר רב של דוגמאות מדוע עדכון דרך טבלאות מדומות הינו בעייתי ולעיתים גם בלתי אפשרי. בעיקרון, כדי שטבלה מדומה תהיה בת-עדכון, צריכה להיות אפשרות להגיע מהטבלה המדומה אל כל שורה בטבלה הבסיסית, ומכל עמודה בטבלה הבסיסית אל כל עמודה בטבלה הבסיסית. תקן SQL1 הגדיר מספר מצבים שהגדרת טבלה מדומה צריכה לקיים כדי שתהיה בת-עדכון והם:

- כל עמודה בטבלה המדומה היא עמודה בטבלה בסיסית. מכאן נובע שאם עמודה כלשהי היא תוצאה של פונקציה מובנית או תוצאה של ביטוי כלשהו בין עמודות (כמו מכפלה של שתי עמודות), לא ניתן לבצע עדכון.
- במשפט FROM המגדיר את הטבלה המדומה מופיעה רק טבלה אחת. מכאן נובע שטבלה מדומה הכוללת צירוף אינה בת עדכון. אם מתקיימת היררכיה של טבלאות מדומות, כלומר טבלה מדומה מבוססת על טבלה מדומה אחרת, כל הטבלאות הנמצאות בהיררכיה חייבות להתייחס אך ורק לטבלה בסיסית אחת.
- במשפט SELECT לא מופיע הפרמטר DISTINCT שמיועד למנוע הופעת שורות כפולות.
- במשפט WHERE לא מופיעה תת-שאילתה.
- בשאילתה אין משפטי GROUP BY ו-HAVING.
- העדכון חייב לשמר את כל כללי שלמות הקשרים של הטבלה הבסיסית. כלומר, אם בטבלה הבסיסית קיים מפתח זר, העדכון חייב לקיים את האילוץ.

כפי שניתן לראות התקן מגביל מאוד את המצבים בהם ניתן לבצע עדכון באמצעות טבלה מדומה. תקן SQL2 מרחיב הגדרות אלו ומאפשר מספר מצבי עדכון נוספים. עקרונית, קיימים מצבים נוספים בהם ניתן לבצע את העדכון למרות שהם חורגים מהמוגדר בתקן. נתבונן לרגע על ההגדרה הבאה של טבלה מדומה:

```
1.  CREATE VIEW GRADE_DIFF
2.                  (STUDENT_ID, COURSE_ID, SEMESTER,
3.                   TERM, GRADE_DIFF) AS
4.        SELECT STUDENT_ID, COURSE_ID, SEMESTER,
5.                TERM, GRADE - GRADE_SEM
6.        FROM    GRADES
7.        WHERE   COURSE_ID LIKE 'M%'
```

הטבלה המדומה מכילה עמודה מחושבת אחת, GRADE_DIFF ולכן לפי התקן הטבלה אינה בת עדכון. אבל מה קורה אם רוצים לעדכן דרך הטבלה המדומה הזאת את מועד הבחינה.

נבצע את הפקודה הבאה:

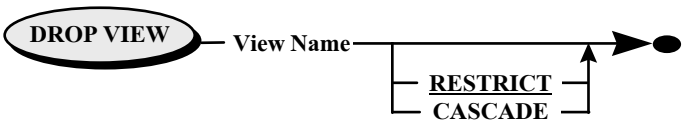
```
1. UPDATE GRADE_DIFF
2.        SET       TERM = 'B'
3.        WHERE  STUDENT_ID = '105' AND
4.                 COURSE_ID = 'M-100' AND
5.                 SEMESTER = 'AUT1998'
```

עקרונית, אין כל בעיה לבצע עדכון זה, מאחר וכל העמודות הדרושות לביצוע העדכון קיימות וניתן להגיע אל השורה בטבלה הבסיסית. העמודה המכילה את הביטוי אינה משתתפת בעדכון ולכן אין כאן בעיה. מספר מערכות מסחריות מאפשרות ביצוע של עדכוני טבלאות מדומות מעבר למגבלות המוגדרות בתקן. לדוגמה, מערכת DB2 כן מאפשרת עדכון מסוג זה.

ביטול טבלה מדומה (Drop View)

פקודה זו מסלקת הגדרה של טבלה מדומה מתוך קטלוג המערכת.

תרשים תחביר:



תרשים 10.4: תרשים תחביר לפקודה DROP VIEW.

הפרמטר CASCADE גורם לביטול כל האובייקטים המתייחסים לטבלה מדומה זו. למשל, אם יש הגדרות אחרות של טבלאות מדומות המתייחסות לטבלה המדומה אותה מבקשים לבטל, גם הגדרות אלו תבוטלנה. האופציה RESTRICT תמנע ביטול של טבלה מדומה אם היא מופיעה בהגדרות של טבלאות מדומות אחרות. זו גם ברירת המחדל.

דוגמה: בטל את הגדרת הטבלה המדומה ששמה HAIFA_STUDENTS. אם קיימות טבלאות מדומות נוספות המבוססות על טבלה זו, יש לבטל גם אותן.

```
1. DROP VIEW HAIFA_STUDENTS  CASCADE
```

השוואה בין טבלה מדומה לבין תת-סכימה

השימוש הבסיסי בטבלאות המדומות מזכיר ביסודו את השימוש בתת-סכימה, כמקובל במערכות היררכיות ורשתיות. הטבלה המדומה מאפשרת יצירת נקודת מבט מותאמת למשתמש או לתוכנית היישום, מבלי שהמשתמש יכיר את המבנה האמיתי של בסיס הנתונים ויקבל תחושה שקיימת טבלה המתאימה בדיוק לצרכיו. באמצעות מנגנון הטבלאות המדומות ניתן לאכוף בקרת גישה ובטיחות, כך שמשתמשים מסוימים יוגבלו

לראות רק חלקים מסוימים מבסיס הנתונים. עם זאת קיימים מספר הבדלים עקרוניים בין המושג תת-סכימה כפי שהוגדר על ידי CODASYL וממומש במערכות לניהול בסיסי נתונים המבוססים על המודל הרשתי לבין הרעיון של טבלה המדומה.

- תוכנית יישום אינה מוגבלת לשימוש בטבלה מדומה אחת בלבד, אלא יכולה לפנות למספר בלתי מוגבל של טבלאות מדומות, בדיוק כפי שהיא פונה למספר בלתי מוגבל של טבלאות בסיס. במערכות המבוססות על תת-סכימה, כל תוכנית יישום מוגבלת לתת-סכימה אחת בלבד.
- טבלה מדומה מאפשרת הגבלת גישה עד לרמת הערך בעמודה ולא רק לרמת השדות או הקבצים. מכיון שבבסיס הטבלה המדומה עומדת הפקודה SELECT, על כל עצמתה, ניתן לבנות טבלאות מדומות המבוססות על תנאי שליפה מתוחכמים מאוד. תת-סכימה לעומתה, אינה מתייחסת לתוכן השדות אלא רק לשדות עצמם ולסוגי הרשומות.
- רק חלק מפעולות העדכון נתמכות על ידי טבלה מדומה. טבלה מדומה יכולה להגזר ממספר רב של טבלאות בסיסיות ועל כן, לא תמיד ניתן לעדכן את בסיס הנתונים דרך הטבלה המדומה. לעומת זאת, תת-סכימה מאפשרת לעדכן את כל הנתונים המופיעים בה.

סיכום

לסיכום, לאחר שהבנו מהי הטבלה המדומה וכיצד היא נוצרת, נסקור בקצרה את היתרונות העיקריים של הטבלאות המדומות.

- **נוחות ופשטות** – בסיס נתונים מורכב ממספר רב של טבלאות שונות הקשורות ביניהן במיגוון רחב של קשרים לוגיים. השימוש בטבלאות מדומות יכול לפשט באופן משמעותי את הגישה אל בסיס הנתונים. אפשר לחייב את המשתמש או את מהנדס התוכנה לבצע אוסף מורכב של פעולות צירוף וגזירה של נתונים מטבלאות מקור כדי לענות על כל דרישת מידע שהיא. עם זאת, ניתן להגדיר באופן חד-פעמי את השאילתה לבניית הטבלה המדומה, לאחסן אותה בקטלוג המערכת, ולאפשר לכל המשתמשים או לתוכניות היישום לגשת אל הנתונים דרך הטבלה המדומה. הדרך השנייה נוחה יותר ואינה מחייבת לחזור בכל פעם מחדש על בניית השאילתה המורכבת.
- **בטחון מידע** – הטבלה המדומה מהווה באופן טבעי מנגנון לאבטחת מידע, מאחר וניתן להגדירה בצורה מותאמת לכל משתמש. אם למשל משתמש מסוים אינו מורשה לראות את כל העמודות או את כל השורות של טבלה כלשהי, ניתן לבנות עבורו טבלה מדומה שתאפשר לו גישה אך ורק לפירטי המידע שהוא מורשה לגשת אליהם. בדוגמה שלנו, ניתן לבנות טבלה מדומה המכילה רק את הציונים של הסטודנטים למדעי המחשב ולאפשר למשתמש מסוים לגשת אל טבלה זו בלבד. מנקודת המבט שלו, אלה כל הנתונים הקיימים בטבלה ואין לו כל דרך לראות ציונים של מחלקות אקדמיות אחרות.

- **שיפור אי-תלות בנתונים** – מאחר ולעיתים יש צורך בשינויים בטבלאות בסיס הנתונים, מתן אפשרות גישה אל הטבלאות דרך הטבלאות המדומות משפרת את רמת אי התלות בנתונים. ניתן לבצע שינויים מסוימים בטבלאות המקור מבלי ששינויים אלה יגרמו לשינוי בטבלה המדומה מנקודת המבט של המשתמש. מאחר והמשתמש ניגש אל בסיס הנתונים דרך הטבלה המדומה, הוא יכול להמשיך ולבנות את השאילתות שלו ללא כל שינוי, למרות שטבלאות המקור השתנו.
- **יעילות** – הטבלאות המדומות אינן תופסות מקום פיסי על הדיסקים, כי רק ההגדרה נשמרת בקטלוג המערכת. תהליך ההידור יכול להיות ארוך במקצת בגלל הצורך לבצע שילוב שאילתות (View Resolution), אולם בדרך כלל זמן זה זניח.

יחד עם יתרונות אלה יש לציין גם מספר חסרונות של השימוש בטבלאות מדומות:

- **רמת עדכניות מוגבלת** – לא כל טבלה מדומה היא בת עדכון. בגלל החשיבות הרבה של השימוש בטבלאות מדומות והיתרונות הרבים שלה, ממשיכה להתבצע עבודה מחקרית רבה סביב נושא עדכון הטבלאות המדומות. יש לשער שעם הזמן נהיה עדים לפחות ופחות אילוצים בנושא העדכון, אם כי ברור שיש מספר טבלאות מדומות שבשום אופן לא יוכלו להיות בנות עדכון.
- **ביצועים** – למרות שציינו קודם שה"קנס" על שימוש בטבלה מדומה אינו גבוה, יש מצבים בהם בבסיס הטבלה המדומה עומדת הגדרה מורכבת מאוד ואם עובדים בסביבה אינטראקטיבית, שבה כל גישה מחייבת בנייה מחדש של השאילתה לצורך הידור ומיטוב, בהחלט ייתכן שנהיה עדים לביצועים ירודים יותר של העבודה דרך טבלאות מדומות מאשר דרך טבלאות מקור.

שאלות חזרה ותרגילים

שאלות חזרה

1. מהי טבלה מדומה וכיצד מגדירים אותה? מה ההבדל בין טבלה מדומה לטבלה בסיסית?
2. הסבר את היתרונות והחסרונות של טבלאות מדומות.
3. הסבר איזה בעיות עלולות להתעורר אם נעדכן את בסיס הנתונים דרך טבלה מדומה? מתי ניתן לבצע עדכון ומתי לא? איזה אילוצים חייבים להתקיים כדי להבטיח שניתן לעדכן את בסיס הנתונים דרך טבלה מדומה?
4. הסבר את השלבים של ביצוע שאילתה המשתמשת בטבלה מדומה.
5. האם מותר לפנות באותה שאילתה לטבלאות מדומות ובסיסיות? אם כן, הסבר את תהליך ביצוע השאילתה.
6. מתי מומלץ להשתמש ב- With Check Option בהגדרת הטבלה המדומה? מה המשמעות של Local ושל Cascade בהקשר זה?

תרגילים

1. נתונה הסכימה הטבלאית הבאה:

א. **יצרן** (מספר יצרן, שם יצרן, עיר, כתובת, ארץ)

ב. **מחשב נייח** (שם מודל, מספר יצרן, מהירות מעבד, גודל דיסק, גודל זיכרון, גודל מסך, מחיר בדולרים)

ג. **מחשב נייד** (שם מודל, מספר יצרן, מהירות מעבד, גודל דיסק, סוג מסך, גדול זיכרון, מחיר בדולרים)

ד. **מדפסות** (שם מודל, מספר יצרן, סוג מדפסת, מהירות הדפסה, צורת חיבור, מחיר בדולרים)

כתוב את פקודות SQL להגדרת הטבלאות המדומות הבאות:

א. טבלה מדומה המכילה את כל נתוני המחשבים הניידים, כולל נתוני היצרן.

ב. טבלה מדומה המכילה את נתוני המדפסות והיצרנים שלהם עבור כל המדפסות מסוג הזרקת דיו ובעלי מחיר של פחות מ- 1000 דולר.

ג. טבלה מדומה המכילה את נתוני המודל, היצרן, מהירות המעבד וגדול הדיסק עבור המחשבים הנייחים והניידים של יצרנים יפניים.

2. כתוב שאילתה המציגה את הנתונים של כל סוגי המחשבים, נייחים וניידים, בעלי גודל זיכרון של 64 MB מתוך הטבלה המדומה המופיעה בסעיף ג' של השאלה הקודמת. הצג את תהליך שילוב השאילתות (View Resolution) וכיצד נראית השאילתה שתבוצע בסופו של דבר.

פרק 11

אבטחת נתונים בסביבת SQL (Data Security)

1. מבוא
2. תפישת ההרשאות בסביבת SQL
3. הענקת זכויות גישה (Grant)
4. ביטול זכויות גישה (Revoke)
5. טבלאות מדומות ואבטחת נתונים
6. סיכום
7. שאלות חזרה ותרגילים

מבוא

בסיס הנתונים מיועד לשרת מספר רב של משתמשים שונים ולכן עליו לספק מנגנונים להגדרת הרשאות. דבר זה יבטיח שכל משתמש יוכל לפעול בבסיס הנתונים רק במסגרת הפעולות המותרות לו ובמסגרת הטבלאות שמותר לו לגשת אליהן. שפת SQL מספקת שתי פקודות מיוחדות, GRANT – להענקת זכויות גישה לבסיס הנתונים, ו-REVOKE – לסילוק זכויות הגישה. באמצעותן יכול מנהל בסיס הנתונים לשלוט על ההרשאות.

בנוסף למודל ההרשאות של סביבת SQL, ניתן להשתמש במנגנון הטבלאות המדומות כדי לתת עוצמה נוספת למודל ההרשאות. שימוש בטבלאות מדומות מאפשר להרחיב את מודל ההרשאות גם להגנה על שורות מסוימות בתוך טבלה ולא רק על טבלאות ועמודות וגם לבנות הרשאות על צירופים שונים של טבלאות.

בפרק זה

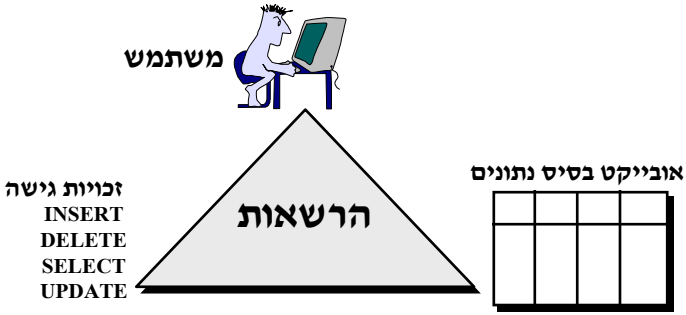
- נציג את תפישת ההרשאות של סביבת SQL, הבנויה משלושה מרכיבים: משתמשים, אובייקטים בבסיס הנתונים וזכויות גישה.
- נסביר את מבנה הפקודה GRANT להענקת זכויות גישה ואת מבנה הפקודה REVOKE לביטול זכויות גישה.
- נציג כיצד לבנות תרשים זכויות גישה.
- נציג את מנגנון הטבלה המדומה ותמיכתו הרבה באבטחת הנתונים.

תפישת ההרשאות בסביבת SQL

תפישת ההרשאות בסביבת SQL מבוססת על שלושה מרכיבים:

- **משתמשים (Users)** – אלה **מבצעים** פעולות שונות בבסיס הנתונים. מערכת RDBMS מבצעת את כל הפעולות בבסיס הנתונים עבור משתמש מסוים. מכאן נובע שהרמה הבסיסית ביותר בתפישת אבטחת הנתונים היא **זיהוי המשתמש**.
- **אובייקטים (Database Objects)** – אובייקטים שמערכת RDBMS צריכה להגן עליהם. האובייקט הנפוץ ביותר הוא **הטבלה**, הן טבלת בסיס והן טבלה מדומה. במציאות מכיל בסיס הנתונים מיגוון רחב של אובייקטים נוספים כגון אינדקסים, תוכניות יישום וכד', ולכן מנגנון ההרשאות צריך לאפשר להגדיר מהו האובייקט המוגן.
- **זכויות (Privileges)** – הפעולות **שמותר** למשתמש לבצע על אובייקט. לדוגמה, משתמש אחד יכול להיות מורשה לבצע את כל הפעולות על טבלה מסוימת (קריאה, עדכון, הוספה, ביטול) בעוד שמשתמש אחר מורשה רק לשלוף נתונים מאותה טבלה, אבל אינו מורשה לעדכן אותה.

כל הרשאה צריכה לשלב את שלושת המרכיבים: מי המשתמש, מה האובייקט המוגן ומה הפעולות שמותר למשתמש לבצע על האובייקט.



תרשים 11.1: שלושת המרכיבים בתפישת ההרשאות בסביבת SQL.

זיהוי משתמשים (User Authentication)

לכל משתמש שרשאי לגשת אל בסיס הנתונים מוקצה **שם משתמש** (User-Id) המזהה אותו באופן חד-ערכי. שם המשתמש מוגדר על ידי מנהל בסיס הנתונים. לפי התקן, שם משתמש הינו SQL Identifier רגיל, כלומר מחרוזת של תווים שמתחילה באות. מערכות מסחריות שונות מגדירות תנאים שונים על שם המשתמש, למשל ב-DB2 שם המשתמש יכול להיות באורך של עד שמונה תווים, ב- SQL Server הוא יכול להיות באורך של עד 30 תווים. בנוסף לשם המשתמש מקובל להוסיף לכל משתמש גם סיסמה (Password).

שם המשתמש מסופק למערכת RDBMS בצורות שונות. אם השימוש הוא אינטראקטיבי, מייד עם תחילת **שיח** (Session), המערכת מבקשת מהמשתמש להזדהות על ידי הזנת שם המשתמש וסיסמתו. אם השימוש הוא על ידי שיבוץ פקודות SQL בשפה מארחת, שם המשתמש והסיסמה מסופקים על ידי תוכנית היישום עם הפנייה הראשונה אל בסיס הנתונים. חלק מהמערכות המסחריות לוקחות את שם המשתמש ואת סיסמתו ישירות ממערכת ההפעלה. כאשר משתמש מבצע תהליך **התחברות** (Login) למערכת ההפעלה, ממילא הוא חייב להזדהות ולכן המערכת יכולה לקבל את שם המשתמש ממערכת ההפעלה ללא צורך בהזדהות כפולה.

בנוסף לשם משתמש המזהה באופן חד-ערכי משתמש כלשהו, ניתן להגדיר גם **שם קבוצה** (Group-Id) באותם מקרים בהם קבוצת משתמשים מקבלת את אותן זכויות גישה. למשל, כל העובדים במחלקה מסוימת מקבלים את אותן זכויות. התקן של SQL אינו מתייחס לסוגיה זו, אולם רוב המערכות המסחריות נותנות פתרונות כלשהם לזכויות גישה של קבוצה בנוסף לזכויות הגישה של המשתמש הבודד. לדוגמה ב- SQL Server קיימת אפשרות לשייך משתמש לקבוצה ולהעניק זכויות גישה לקבוצה כך שכל המשתמשים המשתייכים אליה מקבלים את אותן זכויות. בנוסף, ניתן לבצע שינויים ברמה של המשתמש הבודד למרות השתייכותו לקבוצה. מערכת DB2 פותרת בעיה זו בצורה שונה על ידי מתן אפשרות להגדיר **הרשאות עיקריות** (Primary Authorization Id) **והרשאות משניות** (Secondary Authorization Id).

אובייקטים מוגנים

תפישת ההרשאות מתייחסת להגנה על האובייקטים בבסיס הנתונים. האובייקטים הם בדרך כלל טבלאות מקור וטבלאות מדומות. תקן SQL2 מרחיב את המושג אובייקטים גם **למרחב ערכים** (Domain) ולנושאים נוספים. רוב המערכות המסחריות הרחיבו את המושג לאובייקטים נוספים המנוהלים בבסיס הנתונים. לדוגמה במערכת Oracle ניתן להגן גם על פרוצדורות (Stored Procedures) וגם על אינדקסים. במערכת DB2 ניתן להגן גם על שטחים פיסיים (Tablespaces), שבהן מאוחסנות הטבלאות ולאפשר למנהל בסיס הנתונים להגדיר לאיזה משתמשים מותר להגדיר טבלאות חדשות ובאיזה שטח פיסי.

בעת ההגדרה הראשונית של כל אובייקט בסיס נתונים, מוצמד לו שמו של המשתמש שהגדיר את האובייקט. שם זה מוגדר כשם הבעלים של האובייקט (Object Owner) ורק למשתמש זה יש זכויות גישה ועדכון לאובייקט. כפי שנראה מייד, הבעלים של אובייקט יכול להעניק זכויות גישה לאובייקט שלו למשתמשים נוספים תוך שימוש בפקודה GRANT, או לסלק זכות שהוענקה באמצעות הפקודה REVOKE.

זכויות (Privileges)

הזכויות מגדירות את הפעולות שמותר למשתמש מורשה לבצע על אובייקט מוגן. הפעולות הנפוצות ביותר הן:

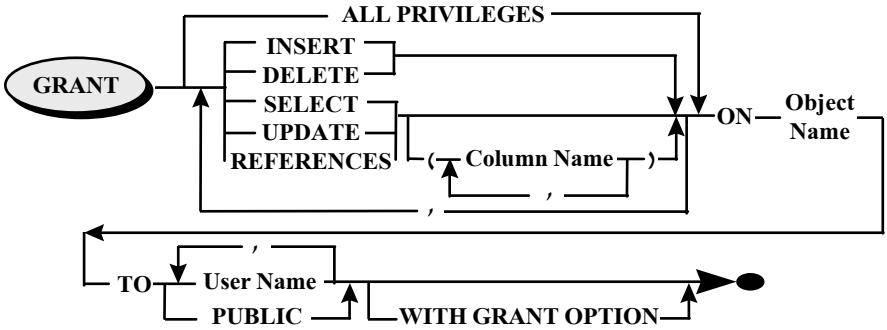
- **בחירה** (Select): זכות לבחור נתונים מתוך טבלת בסיס או טבלה מדומה, כדי לאחזר אותם. התקן גם מאפשר להגדיר את העמודות בטבלה שמותר למשתמש לשלוף מהן.
- **הוספה** (Insert): זכות להוסיף שורות חדשות בטבלת בסיס או בטבלה מדומה.
- **עדכון** (Update): זכות לעדכן שורות של טבלת בסיס או טבלה מדומה. התקן מאפשר להגדיר גם את העמודות בטבלה שמותר למשתמש לעדכן.
- **ביטול** (Delete): זכות לבטל שורות מתוך טבלת בסיס או טבלה מדומה.
- **ייחוס** (Reference): זכות להתייחס לעמודות של טבלה בהגדרת כללי האמינות והשלמות. זכות זו מאפשרת למשתמש להתייחס לעמודה בטבלה אחרת המשתתפת בבדיקת תקינות (בפקודה CHECK), או להתייחס לעמודה בטבלה אחרת באילוץ **שלמות קשרים** (Referential Integrity) למרות שהוא אינו מורשה לגשת לטבלה זו.

כפי שהוסבר, לכל אובייקט חדש שנוצר מוצמד שם הבעלים, כלומר שם המשתמש שהגדיר את האובייקט. לבעלים מוענקים מיידית כל זכויות הגישה לאובייקט שהוא הגדיר. למשתמשים אחרים אין זכויות גישה לאובייקט שלו אלא אם הוא העניק להם אחת או יותר מזכויות הגישה הזו. אם משתמש מגדיר טבלה מדומה, עליו להיות בעל זכויות גישה לכל טבלאות הבסיס שהטבלה המדומה מושתתת עליהם.

הענקת זכויות גישה (Grant)

פקודה זו משמשת להענקת זכות גישה, ומגדירה את שלושת מרכיבי ההרשאה: **מה האובייקט**, **מי המשתמש** ו**מה זכויות הגישה** המוענקות לו.

תחביר הפקודה:



תרשים 11.2: תרשים תחביר לפקודה GRANT.

האובייקט המוגן מוגדר על ידי שם האובייקט, שבדרך כלל הוא שם טבלת בסיס או שם של טבלה מדומה. הפרמטר ALL PRIVILEGES מעניק את כל זכויות הגישה לטבלה. אם מבקשים להעניק רק חלק מהזכויות, יש לציין במפורש את הזכות המוענקת. נשים לב שעבור זכות שליפה ועדכון ניתן גם להתייחס לעמודות בטבלה.

הפרמטר PUBLIC מגדיר לאובייקט זה זכות גישה לכל המשתמשים. הפרמטר WITH GRANT OPTION קובע שהמשתמש שהוענקה לו זכות גישה, יכול בעצמו להעניק זכויות גישה למשתמשים נוספים.

דוגמה א' : הענקת כל הזכויות **לגישה** לטבלת סטודנטים למשתמש ששמו "דן".

```
1.  GRANT ALL PRIVILEGES ON STUDENTS
2.              TO DAN
```

דן קיבל את כל הזכויות לטבלת סטודנטים מבעל הטבלה. דן לא יכול להעניק זכויות גישה למשתמשים אחרים. אם נרצה לאפשר לו להעניק זכויות, הפקודה תהיה:

```
1.  GRANT ALL PRIVILEGES ON STUDENTS
2.              TO DAN WITH GRANT OPTION
```

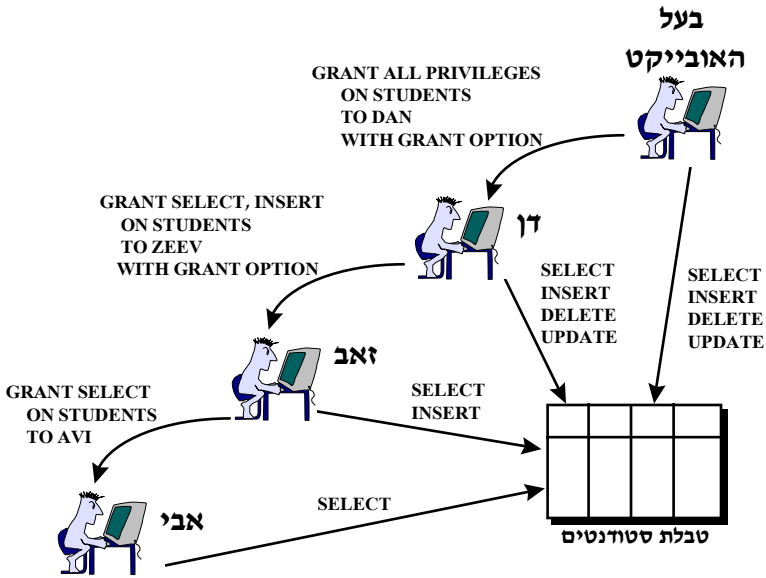
במצב זה, דן יכול להעניק לזאב זכות לשליפה והוספה בלבד לטבלת סטודנטים, וגם מעניק לזאב זכות להעניק זכויות גישה למשתמשים נוספים.

```
1.  GRANT SELECT, INSERT ON STUDENTS
2.              TO ZEEV  WITH GRANT OPTION
```

זאב עצמו יכול להעניק זכות לשליפה בלבד למשתמש ששמו אבי.

```
1.  GRANT SELECT ON STUDENTS TO AVI
```

נוצרת כאן היררכיית זכויות שמתחילה בבעל האובייקט ועוברת דרך דן, דרך זאב ומסתיימת במשתמש ששמו אבי. נשים לב שלבעל האובייקט ולדן יש את כל זכויות הגישה, לזאב יש זכות לשליפה והוספה בלבד ולאבי יש זכות לשליפה בלבד.



תרשים 11.3: היררכיה של זכויות גישה לאובייקט

דוגמה ב' : הענקת זכות **שליפה ועדכון** לטבלת מחלקות למשתמשים דן, זאב ואבי.

```
1.  GRANT SELECT, UPDATE ON DEPARTMENTS
2..             TO DAN, ZEEV, AVI
```

דוגמה ג' : הענקת זכויות **שליפה** לטבלת המחלקות לכל המשתמשים.

```
1.  GRANT SELECT ON DEPARTMENTS TO PUBLIC
```

דוגמה ד' : הענקת זכויות **שליפה ועדכון** על עמודות מסוימות בטבלת קורסים לדן ולזאב.

```
1.  GRANT SELECT (COURSE_ID, COURSE_NAME, TYPE),
2.              UPDATE (COURSE_ID, COURSE_NAME, TYPE)
3.              ON  COURSES
4.              TO   DAN, ZEEV
```

דוגמה ה' : הענקת זכות **שליפה ועדכון** למספר משתמשים למספר טבלאות עם אפשרות שהם יעניקו זכות גישה למשתמשים נוספים.

```
1.  GRANT SELECT ON  DEPARTMENTS, STUDENTS, COURSES
2.              TO DAN, ZEEV, AVI
3.               WITH GRANT OPTION
```

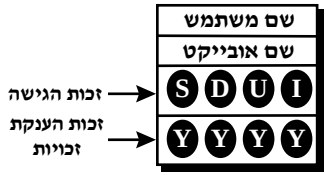
דוגמה ו' : הענקת זכות **שליפה** בלבד לטבלה המדומה ששמה HAIFA_STUDENTS למשתמש ששמו דן.

1. GRANT SELECT ON HAIFA_STUDENTS TO DAN

דוגמה ז' : הענקת זכות **לבנות אינדקס** על הטבלה מחלקות.

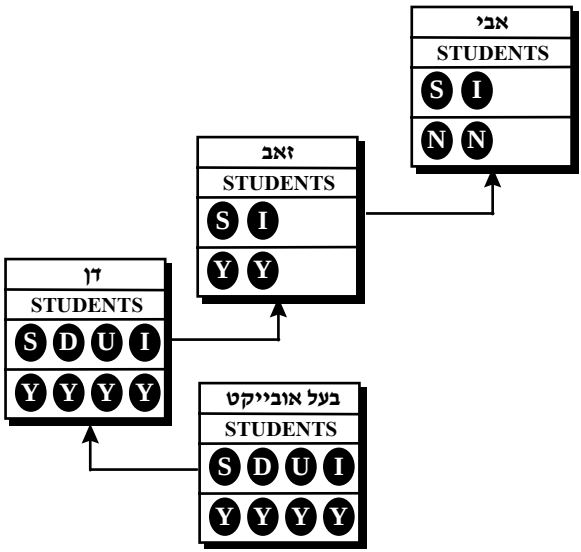
1. GRANT INDEX ON DEPARTMENTS TO DAN

כפי שניתן לראות מדוגמאות אלו, יכולה להתקבל רשת מורכבת של זכויות המועברות ממשתמש למשתמש. המצב אפילו מסתבך, מאחר ומשתמשים מסוימים יכולים לקבל זכויות שונות ממשתמשים אחרים. כדי להבין את מצב ההרשאות ומי העניק למי איזה זכויות, מקובל להשתמש בתרשים הנקרא **תרשים זכויות גישה** (Grant Diagram). בכל צומת של התרשים מופיע מלבן המכיל את הנתונים הבאים: שם המשתמש, שם האובייקט, האם יש לו זכויות להעניק הלאה זכויות ומהן זכויות הגישה לאובייקט: S מסמן SELECT, D מסמן DELETE, U מסמן UPDATE ו- I מסמן INSERT. היתר הענקת זכויות מסומן ב-Y, ואיסור הענקת זכויות מסומן ב-N.



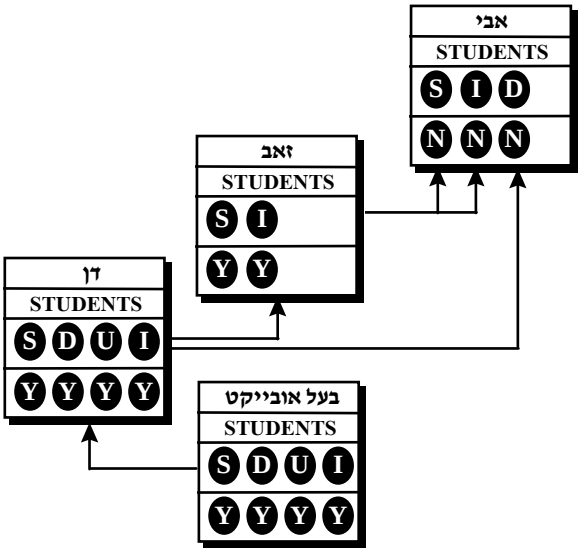
תרשים 11.4: מבנה צומת בתרשים זכויות גישה.

נציג עכשיו את תרשים זכויות הגישה כפי שהתקבל מהמצב המתואר לעיל:



תרשים 11.5: תרשים זכויות גישה.

נניח עכשיו שדן מעניק זכויות **מחיקה** לאבי. אזי עכשיו אבי הוא בעל זכויות לשליפה והוספה אותן קיבל מזאב, וזכות למחיקה אותה קיבל מדן. התרשים הבא מציג מצב זה.

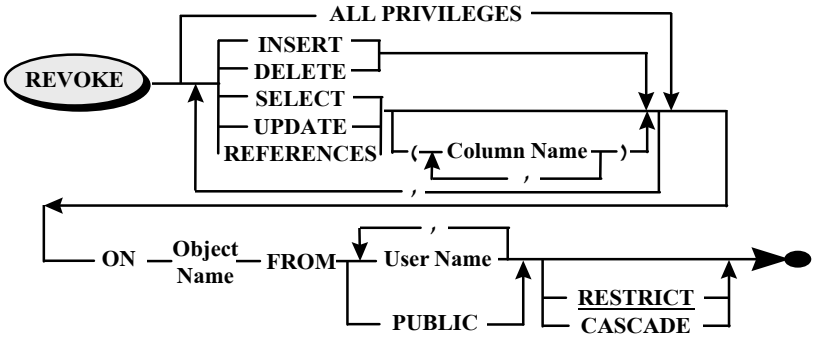


תרשים 11.6: תרשים זכויות גישה מעודכן.

ביטול זכויות גישה (Revoke)

פקודה זו הפוכה לפקודה GRANT כי היא מבטלת את זכויות הגישה שהוענקו למשתמשים מסוימים.

תחביר הפקודה :



תרשים 11.7: תרשים תחביר של הפקודה REVOKE.

מבנה פקודה זו דומה למבנה הפקודה GRANT. הפרמטר RESTRICT מונע את ביטול זכות הגישה אם קיימת היררכיה של זכויות. הפרמטר CASCADE מבטל את כל היררכיית הזכויות שהוענקו על ידי המשתמש.

דוגמה א' : ביטול כל זכויות הגישה לטבלת סטודנטים למשתמש ששמו דן. אם דן העניק זכויות גישה למשתמשים נוספים, יש לבטל גם אותן.

```
1.  REVOKE ALL PRIVILEGES ON STUDENTS
2.                 FROM DAN CASCADE
```

אם אין אנו רוצים לבטל את זכויות הגישה של דן במקרה שהעניק זכויות גישה למשתמשים אחרים, נוכל לרשום :

```
1.  REVOKE ALL PRIVILEGES ON STUDENTS
2.                 FROM DAN RESTRICT
```

גם אם לא היינו רושמים את הפרמטר RESTRICT הביטול היה נמנע, כי זו ברירת המחדל.

דוגמה ב' : ביטול זכות מסוימת מהמשתמש זאב, למשל עדכון הטבלה קורסים.

```
1.  REVOKE  UPDATE  ON  COURSES FROM ZEEV
```

נשים לב, שאם למשתמש זאב יש זכויות אחרות, כגון שליפה, הן לא תבוטלנה.

דוגמה ג' : ניתן גם לבטל את זכויות הגישה לעמודה מסוימת. למשל אם למשתמש זאב יש זכויות עדכון על מספר עמודות בטבלת הקורסים, ניתן לבטל את זכות העדכון של עמודה מסוימת ולהשאיר לו את זכויות העדכון של יתר העמודות.

```
1.  REVOKE  UPDATE ( TYPE) ON  COURSES
2.                 FROM ZEEV
```

דוגמה ד' : ביטול זכויות השליפה מטבלת מחלקות לכל המשתמשים.

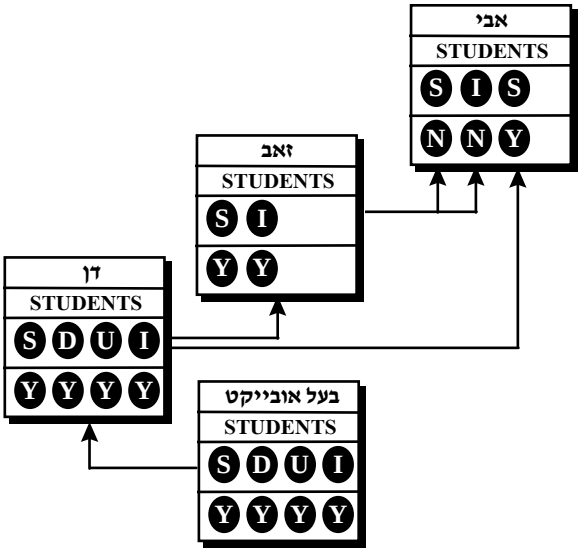
```
1.  REVOKE  SELECT ON DEPARTMENTS FROM PUBLIC
```

דוגמה ה' : נניח ששני משתמשים שונים, למשל דן וזאב, העניקו זכות גישה למשתמש שלישי, אבי. תרשים 11.8 מציג מצב זה. נשים לב שזאב העניק זכות שליפה לאבי ללא יכולת להעניק הלאה זכויות, ודן העניק זכות שליפה לאבי עם יכולת להעניק הלאה זכויות גישה.

עכשיו נבטל את זכות הגישה של זאב על ידי הפקודה :

```
1.  REVOKE  SELECT, INSERT ON  STUDENTS FROM ZEEV
```

המשתמש אבי עדיין יהיה בעל זכויות שליפה כי קיבל אותן גם מדן. נציין שמערכות מסחריות שונות מטפלות בצורה שונה בביטול זכויות בהיררכיית זכויות. חלקן מבטלות את כל ההיררכיה וחלקן – לא.



תרשים 11.8: תרשים זכויות גישה.

טבלאות מדומות ואבטחת נתונים

עד כאן התייחסנו למודל ההרשאות בסביבת SQL והיכולות של הפקודה GRANT בהגדרת והענקת זכויות גישה לאובייקטים של בסיס הנתונים. בפרק הקודם התייחסנו לטבלאות מדומות ואמרנו שגם הן מהוות חלק ממנגנון ההגנה על בסיס הנתונים.

שילוב זה של טבלה מדומה עם מודל ההרשאות של SQL מעניק בסופו של דבר מודל הרשאות מתוחכם מאוד בעל יכולות של תפירת הרשאות ייחודיות למשתמשים שונים. חשוב עם זאת לזכור שטבלאות מדומות סובלות ממספר מגבלות בעדכון הטבלאות ולכן בכל הקשור להגדרת ההרשאות לעדכון, קיימות מגבלות בשימוש בטבלאות מדומות. נרחיב במקצת את הדיון בשילוב של הטבלאות המדומות בתפישת אבטחת הנתונים.

הגנה על שורות בתוך טבלה

כפי שראינו, הפקודה GRANT מסוגלת לקבוע זכויות גישה לטבלה או לעמודות בתוך טבלה. אם רוצים להעניק זכויות גישה ברמת ערכים בתוך עמודה, לא נוכל לעשות זאת עם הפקודה GRANT, אבל נוכל לעשות זאת על ידי שילוב של טבלה מדומה עם פקודה זו.

נניח שיש צורך לאפשר למשתמש כלשהו לגשת בטבלה קורסים לשורות של קורסים מסוג מעבדה בלבד. הדרך לממש הגנה זו היא על ידי הגדרת הטבלה המדומה הבאה:

```
1.  CREATE VIEW LAB_COURSES AS
2.        SELECT    *
3.        FROM      COURSES
4.        WHERE     TYPE = 'LAB'
```

עכשיו נעניק למשתמש זאב זכות גישה לטבלה המדומה שהגדרנו.

```
1.  GRANT SELECT ON LAB_COURSES TO ZEEV
```

נזכיר רק, שמשתמש שמעניק זכות גישה זו חייב להיות בעצמו בעל זכות גישה לטבלת קורסים. אם אין לו זכות גישה לטבלת זו, הוא אינו יכול להעניק זכויות גישה לטבלה מדומה המבוססת על טבלת זו.

הגנה על שילוב כלשהו של עמודות ושורות

בעת הצגת נושא הטבלאות המדומות הסברנו את העוצמה של כלי חשוב זה ואת התחכום הרב העומד בבסיס ההגדרה של הטבלה המדומה. כמעט כל מה שניתן להגדיר בשאילתת SQL רגילה ניתן להגדרה כטבלה מדומה. מכיון שטבלה מדומה יכולה להיות מבוססת גם על צירוף בין טבלאות שונות, ניתן לתת הרשאות גישה למשתמשים דרך טבלה מדומה שמאחוריה מסתתרות מספר טבלאות בסיס.

לדוגמה, נציג את הטבלה המדומה של ציוני הסטודנטים למדעי המחשב.

```
1.  CREATE VIEW CS_GRADES AS
2.          SELECT  S.NAME, S.STUDENT_ID, C.COURSE_NAME,
3.                  C.COURSE_ID, G.SEMESTER, G.TERM,
4.                  G.GRADE, G.GRADE_SEM
5.          FROM    STUDENTS S, COURSES C, GRADES G
6.          WHERE   G.STUDENT_ID = S.STUDENT_ID AND
7.                  G.COURSE_ID = C.COURSE_ID AND
8.                  C.DEPARTMENT_ID = 'CS'
```

עכשיו נוכל להעניק זכות גישה לטבלה מדומה זו למשתמשים דן, אבי וזאב עם יכולת להעניק זכויות גישה למשתמשים נוספים.

```
1.  GRANT SELECT ON CS_GRADES
        TO ZEEV, DAN, AVI WITH GRANT OPTION
```

סיכום

בפרק זה הצגנו את מרכיבי מודל ההרשאות בסביבת SQL. כפי שראינו המודל מתייחס לשלושה מרכיבים שונים: מי המשתמש, מה האובייקט המוגן ומהן זכויות הגישה המוענקות למשתמש. שילוב כלשהו בין שלושת המרכיבים האלה מגדיר **הרשאה**. את ההרשאות האלו מעניקים באמצעות הפקודה GRANT ומבטלים באמצעות הפקודה REVOKE. הפקודה GRANT מאפשרת האצלת סמכות להענקת זכויות גישה למשתמשים אחרים ולכן יכולה להבנות היררכיה מורכבת למדי של זכויות גישה.

בנוסף למודל ההרשאות של סביבת SQL ניתן להשתמש בשילוב של הטבלאות המדומות עם מודל ההרשאות לקבלת מודל הרשאות מתוחכם מאוד.

שאלות חזרה ותרגילים

שאלות חזרה

1. מהן רמות ההרשאה השונות שבהן שפת SQL תומכת?
2. הסבר מהו הקשר בין פקודת Grant לפקודת Revoke.
3. הסבר כיצד טבלה מדומה יכולה לשמש לאבטחת גישה לבסיס נתונים.
4. מהם כל הזכויות (Privileges) שניתן להגדיר? הסבר כל אחת מהזכויות האלו.
5. הסבר מדוע יש צורך בפרמטר With Grant Option בפקודה Grant.

תרגילים

1. נתונה הטבלה הבאה:

עובדים (מספר עובד, שם עובד, עיר מגורים, כתובת, טלפון)

כתוב את פקודות SQL האלו:

א. הענקת זכות Select בלבד למשתמש "רונן".

ב. הענקת זכויות Select, Insert ו-Delete למשתמש "אייל". לאייל מותר להעניק זכויות גישה נוספות.

ג. הענקת זכות Select לכל העמודות וזכות Update לעמודות עיר מגורים וכתובת למשתמשים "רותי" ו"תמי".

ד. המשתמש "אייל" מעניק זכויות Select ו-Delete למשתמש "יורם". ליורם מותר להעניק זכויות גישה נוספות.

ה. המשתמש "יורם" מעניק זכות Select למשתמש "אריה". לאריה אסור להעניק זכויות גישה למשתמשים אחרים.

ו. בטל את כל זכויות הגישה של המשתמש "אייל".

2. צייר את תרשים זכויות הגישה המתקבל המתקבל כתוצאה מסעיפים א' עד ה' בשאלה הקודמת.
3. כתוב את פקודות SQL המתאימות להענקת הזכויות Select ו-Update למשתמש "אריה" ולעובדים שעיר מגוריהם "הרצליה" בלבד.

פרק 12

תכנות בסביבת SQL
(Programming with SQL)

1. מבוא
2. הפעלת קדם-מהדר בשיטה הסטטית
3. תכנות עם פקודות SQL דינמיות (Dynamic SQL)
4. תכנות עם ממשק תכנות (Application Program Interface)
5. סיכום
6. שאלות חזרה ותרגילים

מבוא

עד כאן הצגנו את פקודות SQL והנחנו שהן מופעלות באופן אינטראקטיבי מתחנת עבודה כלשהי. שיטת עבודה זו מתאימה בעיקר עבור משתמשים מזדמנים המבקשים להפעיל שאילתות שונות או לבצע עדכונים מוגבלים בבסיס הנתונים. ברור שזהו מיעוט המקרים בהם משתמשים בבסיסי נתונים טבלאיים. הרוב המוחלט של השימוש בבסיסי נתונים טבלאיים הוא על ידי תוכניות יישום המבצעות פעולות בעלות משמעות עסקית. פרק זה הינו מורכב יחסית ודורש רקע בתכנות, רצוי בשפת C ומיועד בעיקר למהנדסי תוכנה המתכוונים לפתח תוכניות יישום הפועלות בסביבת SQL.

מערכות לניהול משאבי הארגון, הידועות בשם מערכות ERM (Enterprise Resource Management), מערכות לניהול לקוחות, המכונות מערכות CRM (Customer Relationship Management), מערכות לחיוב וגביה (Billing and Customer Care) או מערכות מידע פנימיות שכל ארגון מפתח, כל אלו הן דוגמאות למערכות מידע גדולות, המורכבות מאלפי תוכניות יישום והמבצעות את התנועות העסקיות (Business Transactions) שהארגון צריך לבצע. דוגמאות לתנועות עסקיות שמערכות כאלו מבצעות הן עדכון נתוני לקוח, עדכון יתרות המלאי במחסנים, רישום של תנועות פיננסיות להפקת דוח רווח והפסד ומאזן, הפקת חשבונות ללקוח, גביית תשלומים מלקוחות, פתיחת הזמנות רכש, ניהול נתוני עובדים, קידום עובד בדרגה וכד'.

היישומים העסקיים השונים פועלים על תשתית של בסיס נתונים ותוכניות יישום מורכבות המכילות את כל הלוגיקה העסקית לביצוע התנועות. תוכניות היישום כתובות בשפת תכנות כלשהי, שפות כגון COBOL, C, Pascal, PL/1, Visual Basic, PowerBuilder, Java, Magic, Developer 2000, ABAP (שפת התכנות הקניינית של מערכת SAP שהינה אחת ממערכות ERP הנפוצות) וכד'. יישומים אלה פועלים בעיקר מול בסיסי נתונים טבלאיים ומשתמשים בשפת SQL כשהיא משובצת בתוך שפת תכנות כלשהי. מקובל לקרוא לשפת תכנות כזו **שפה מארחת** (Host Language), מאחר והיא מארחת בתוכה פקודות בשפה אחרת, שפת SQL שמיועדות לגישה ולטיפול בנתונים המנוהלים בבסיסי נתונים טבלאיים.

בפרק זה

- ❖ נסקור את הצורה בה ניתן להפעיל פקודות SQL מתוך שפה מארחת כלשהי.
- ❖ נראה ששיבוץ פקודות SQL בתוך השפה המארחת דורש מספר מנגנונים מיוחדים והגדרת מספר שטחי עבודה שיאפשרו למערכת RDBMS לתקשר עם תוכנית היישום.
- ❖ נציג ונדון בשיטות הגישה לבסיסי נתונים.

אפשרויות הגישה לבסיסי נתונים טבלאיים מתוך תוכנית יישום

קיימות שלוש שיטות עיקריות לגישה לבסיסי נתונים טבלאיים מתוך שפת תכנות מארחת:

- **שימוש בקדם-מהדר ובפקודות SQL סטטיות** (Embedded Static SQL): בשיטה זו פקודות SQL משובצות באופן ישיר בתוך השפה המארחת. כל פקודות SQL מסומנות בדרך מיוחדת. תוכנית היישום עוברת תהליך **קדם-הידור** (Pre Compilation) שהופך פקודות אלו לפקודות CALL מיוחדות הפונות לקבלת השירותים של מערכת RDBMS. מערכת זו מספקת את קדם-המהדר עבור שפות התכנות המארחות הנתמכות. בזמן ריצת התוכנית, פקודות Call מפעילות את שגרות זמן הריצה של מערכת RDBMS ומעבירות את פרמטרים המתאימים, מקבלות את הנתונים מבסיס הנתונים ומעבירות אותם למשתנים של השפה המארחת. פקודות SQL מוגדרות באופן מלא בזמן כתיבת תוכנית היישום, למעט פרמטרים שניתן להעביר אל הפקודה או לקבל מהפקודה. מסיבה זו מקובל לקרוא לשיטה זו בשם SQL סטטי.
- **שימוש בקדם-מהדר** ובפקודות **SQL דינמיות** (Embedded Dynamic SQL): בשיטה הסטטית לא ניתן לאפשר למשתמש להגדיר תוך כדי העבודה את הטבלה שהוא מבקש לפעול עליה, איזה עמודות ברצונו לראות או את תנאי השליפה של השאילתה. כל אלה חייבים להיות מוגדרים מראש כדי שניתן יהיה לתכנת אותם בתוך פקודות SQL המשובצות. לעיתים יש קושי להגדיר מראש את כל מרכיבי פקודות SQL. לדוגמה, פיתוח מחולל שאילתות שיאפשר למשתמש להחליט במהלך הביצוע (On the fly) איזה טבלאות הוא רוצה לראות, איזה עמודות מתוך השפה הוא רוצה להציג ואיזה תנאי שליפה הוא רוצה להגדיר – אינו בר ביצוע בשיטה הסטטית. לשם כך פותחה שיטה לשיבוץ פקודות SQL דינמיות, המאפשרת לבנות את משפט SQL תוך כדי ריצת התוכנית, ולא מראש.
- **שימוש בממשק תכנות יישומים** (SQL Application Programming Interface): בשיטה זו הפנייה אל מערכת RDBMS מתבצעת האמצעות ממשק תכנות (API) מיוחד. הממשק מאפשר למסור למערכת את מהות השירות המבוקש ואת הפרמטרים השונים. שיטה זו אינה דורשת קדם-מהדר, כי תוכנית היישום בונה את כל הפניות על ידי שימוש בפקודה CALL. יצרן בסיס הנתונים מפרסם את כל השירותים שהמערכת מסוגלת לספק וכיצד יש להפעיל אותם.

יש מערכות מסחריות שאינן תומכות בכל השיטות האלו. להלן מספר דוגמאות:

- **Oracle** תומכת בכל השיטות. בעבודה עם קדם-מהדר קיימת תמיכה בשפות התכנות Cobol, Pascal, C, PL/1, Ada ובממשק התכנות OCL (Oracle Call Level).
- **Sybase** ו- **Microsoft SQL Server** תומכות בממשק תכנות בלבד.
- **Informix** תומכת בעבודה עם קדם-מהדר בשיטה הסטטית והדינמית בלבד ובשפות התכנות Cobol, C, Ada ועוד.

- **DB2** תומכת בעבודה עם קדם-מהדר בשיטה הסטטית והדינמית ובשפות תכנות Cobol, Assembler,C , PL/1, Fortran. בנוסף יש תמיכה בממשק התכנות DB2 CLI (Call Level Interface).
- **CA-Ingres** תומכת בעבודה עם קדם-מהדר בשיטה הסטטית והדינמית ובשפות התכנות Cobol, C, Pascal, Ada, PL/1 ועוד.

הפעלת קדם-מהדר בשיטה הסטטית

השיטה הסטטית היא אחת משלוש השיטות לכתיבת יישומים משובצי פקודות SQL הפועלים במערכות RDBMS. בסעיף זה נסקור את עקרונות העבודה עם קדם-המהדר ונציג שימוש בדוגמאות הכתובות בשפת C כשפה מארחת. העקרונות של שיבוץ פקודות SQL בתוך שפה מארחת אחרת דומים מאוד, ומחייבים בעיקר את ההתאמות לצורת הגדרת המשתנים של השפה ולכללי השפה.

עקרונות העבודה עם קדם-מהדר

ללא תלות בשפת התכנות המארחת, עקרונות העבודה עם קדם-מהדר זהים וכוללים:

- **שילוב פקודות רגילות ופקודות SQL:** פקודות SQL משובצות במקומות המתאימים בתוך השפה המארחת. משמעות הדבר היא שמפתח תוכנית היישום כותב את היישום ובונה את הלוגיקה שלו באמצעות הפקודות הרגילות של שפת התכנות. בכל מקום בתוכנית שבו צריך לשלוף או לעדכן נתונים בבסיס הנתונים הוא כותב פקודות SQL. בצורה זו, כל מורכבות הפניות לשגרות של מערכת RDBMS והפרמטרים השונים, נסתרים מעיני מפתח היישום. המפתח כותב פקודות SQL רגילות וקדם-המהדר מתרגם אותן לפקודות המיוחדות להפעלת מערכת RDBMS.
- **סימון פקודות SQL**: פקודות SQL אינן פקודות מוכרות וחוקיות בשפת התכנות המארחת ולכן דרוש לסמן אותן באופן מיוחד כדי שקדם-המהדר יוכל לזהותן. כל פקודת SQL מתחילה עם מחרוזת קבועה – EXEC SQL – ומסתיימת בתו הסיום הרגיל של השפה, או במחרוזת END EXEC. לדוגמה, בשפת C, כל פקודת SQL תסתיים בתו נקודה-פסיק (;) ; בשפת COBOL כל פקודה תסתיים במחרוזת END EXEC ; ב- Java כל פקודה תתחיל ב- sql# ותסתיים בנקודה-פסיק.
- **פנייה למשתנים של השפה המארחת:** פקודות SQL יכולות להכיל פנייה למשתנים (Variables) של השפה המארחת. שיטה זו מאפשרת קשר דו-סיטרי בין השפה המארחת לבין פקודות SQL המשובצת בה. ניתן להעביר פרמטרים לצורך ביצוע פקודות SQL ולאחר הביצוע ניתן לקבל את התוצאות של פקודת SQL חזרה בתוך משתנים של השפה המארחת להמשך עיבוד. כל המשתנים של השפה המארחת המופיעים בפקודת SQL כלשהי חייבים להיות מוגדרים בתוך קטע מיוחד המתחיל בפקודה BEGIN DECLARE SECTION ומסתיים בפקודה END DECLARE SECTION. כל משתנה כזה יופיע בתוך פקודת SQL עם קידומת מיוחדת, נקודתיים.

- **מבנה פקודות SQL**: פקודות SQL המשובצות בתוך שפה מארחת הן במבנה רגיל ומוכר, למעט מספר הבדלים שרובם נובעים מהצורך לאפשר פנייה למשתנים של השפה המארחת.
- **עבודה עם טבלאות תוצאתיות**: הפקודה SELECT מציבה אתגר מיוחד בשיבוצה בתוך שפה מארחת, כי היא יכולה להחזיר טבלת תוצאה מרובת שורות. כדי לטפל במצב זה פותח מנגנון מיוחד הקרוי Cursor (סַמָן), המאפשר לגשר על ההבדלים בין שיטת העבודה של השפה המארחת המבצעת כל פעם פקודה אחת, לבין תפישת העבודה של סביבת SQL שיכולה להחזיר לאחר ביצוע SELECT טבלת תוצאה עם מספר בלתי ידוע מראש של שורות.
- **פקודות ייחודיות לעבודה עם קדם-מהדר:** בנוסף לפקודות SQL הרגילות, קיימות מספר פקודות ייחודיות שמטרתן לאפשר עבודה בתוך שפה מארחת. בין הפקודות האלו נמצא פקודות המאפשרות עבודה עם מנגנון הסמן, כגון FETCH, OPEN CURSOR, פקודות מיוחדות לעבודה עם SQL דינמי או פקודות מיוחדות להגדרת המשתנים וכד'. פקודות אלו אינן נדרשות ואינן חוקיות אם שיטת הפעלת הפקודות היא אינטראקטיבית.
- **שטח תקשורת:** התקשורת בזמן ריצה בין מערכת RDBMS לבין השפה המארחת מתבצעת באמצעות שטח תקשורת (Communication Area) בזיכרון, המאפשר העברת מידע בין מערכת RDBMS לבין שפת התכנות המארחת. שטח תקשורת זה מכיל אוסף משתנים שמערכת RDBMS אחראית לתוכנם. פקודות השפה המארחת יכולות לפנות למשתנים אלה כדי לקבל מידע על ביצוע הפקודה כגון האם הפקודה בוצעה בהצלחה או נכשלה וכד'. שטח תקשורת זה הינו בעל מבנה קבוע ומוכנס אל תוך השפה המארחת על ידי הפקודה INCLUDE.

תהליך קדם-הידור (Pre-compilation Process)

פקודות SQL המשובצות בשפה מארחת אינן "חוקיות" למעשה ואינן מוכרות על ידי המהדר של השפה המארחת. כדי לאפשר את השיבוץ, משתמשים בקדם-מהדר ובתהליך עבודה מיוחד. מטרת קדם-המהדר לאתר את כל פקודות SQL המשובצות ולהחליף אותן בפקודות Call אשר נתמכות על ידי כל שפות התכנות ומכילות את כל הפרמטרים הדרושים לפנייה למערכת RDBMS. יצרן מערכת ה-RDBMS מספק קדם-מהדר מתאים עבור כל שפה מארחת. לדוגמה, בסביבת Oracle קיים מהדר PRO*C עבור שפת C ומהדר PRO*COBOL עבור שפת COBOL ומהדרים נוספים. בסביבת DB2 קיים מהדר SQLJ עבור Java, ומהדרים אחרים עבור שפות תכנות נוספות.

נתאר את השלבים השונים בתהליך ההידור של תוכנית יישום בשפה מארחת הכוללת פקודות SQL ונשתמש בדוגמה של תוכנית הכתובה בשפת C. התפקיד של כל פקודת SQL יוסבר בהמשך. כרגע נתמקד בתהליך בניית התוכנית, משלב הכתיבה ועד שלב הריצה.

שלב א': כתיבת תוכנית יישום

בשלב הזה מפתח היישום כותב תוכנית בשפה המארחת ומשבץ בה פקודות SQL כפי הצורך.

```
1.  # include <stdio>
2.  # include <ctype.h>
3.  EXEC SQL BEGIN DECLARE SECTION;
4.    varchar  userid(20);
5.    varchar  password (20);
6.    varchar  student_number (5);
7.    varchar  student_name (20);
8.
9.  EXEC SQL END DECLARE SECTION;
10. EXEC SQL INCLUDE SQLCA;
11. main()
12. {
13.   printf (“Enter user name: “;
14.   scanf (“%s”, userid);
15.   printf (“\n Enter password: “);
16.   scanf (“%s”, password);
17.   EXEC SQL WHENEVER SQLERROR GOTO sql_error;
18.   EXEC SQL CONNECT :userid IDENTIFIED BY :password;
19.   printf (“\n Enter Student Number: “);
20.   scanf (“%s”, student_number);
21.   EXEC SQL WHENEVER SQLERROR GOTO sql_error;
22.   EXEC SQL WHENEVER NOT FOUND GOTO sql_not_found;
23.   EXEC SQL SELECT NAME INTO :student_name
24.                FROM    STUDENTS
25.                WHERE STUDENT_ID = :student_number;
26.    printf (“Student Name is:  %s\n“, student_name);
27.    exit()
28. sql_error:
29.    printf (“SQL error: %ld\n”, sqlca.sqlcode);
30.    exit();
31. sql_not_found:
32.    printf (“Student not found. \n”);
33.    exit();
34.  }
```

תרשים 12.1: תוכנית בשפת C שבה משובצות פקודות SQL.

התוכנית המוצגת מבצעת שאילתה להצגת שם הסטודנט. נסביר אותה בקצרה:

- שורה 1 גורמת להכללת אוסף של משתנים כלליים הנשמרים בקובץ STDIO.H, כדי שלא נצטרך בכל תוכנית לחזור ולהגדיר משתנים אלה.
- שורות 13 עד 16 בתוכנית מבקשות את המשתמש להזין את שם המשתמש ואת הסיסמה שלו.
- שורה 18 מבצעת את הקישור עם בסיס הנתונים.

- שורות 19 ו-20 מבקשות מהמשתמש להזין מספר סטודנט.
- שורות 23 עד 25 מכילות את השאילתה המתייחסת לטבלה סטודנטים.
- שורה 26 מציגה את שם הסטודנט על המסך.
- שורות 31 ו-32 מציגות הודעת שגיאה אם המשתמש מזין מספר סטודנט שאינו קיים.

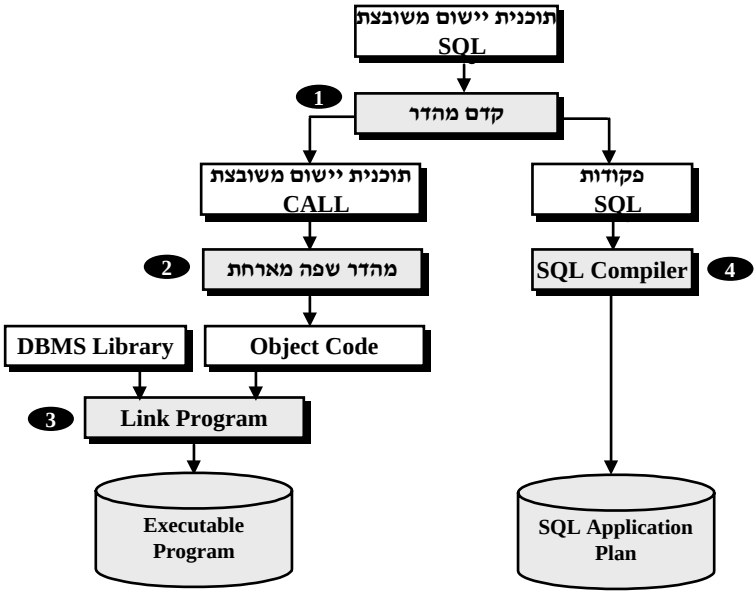
הפונקציות main(), printf(), scanf() הן של שפת C. כל תוכנית C חייבת להתחיל בפונקציה main() המציינת היכן התוכנית מתחילה. כל התוכנית חייבת להיות עטופה בסוגריים מסולסלים, { }. הפונקציה printf(), מדפיסה על המסך את המחרוזת המופיעה בין הגרשיים ומאפשרת דילוג לשורה הבאה על ידי ההוראה \n. הפונקציה scanf() קוראת קלט מהמסך. בשלב זה לא נסביר את מהות הפקודות השונות של SQL, אלא רק נצביע על כך שלפנינו תוכנית רגילה הכתובה בשפת C הכוללת בתוכה פקודות SQL.

שלב ב': הידור התוכנית

בשלב זה מתחיל תהליך ההידור של התוכנית. להבדיל מהידור רגיל של תוכנית בשפת C, מתחלק כאן תהליך ההידור למספר שלבי ביניים.

- **צעד 1 – קדם-הידור:** תחילה מופעל קדם-מהדר המסופק על ידי יצרן מערכת RDBMS. קדם-מהדר זה מאתר את כל פקודות SQL על פי המחרוזת EXEC SQL שבראש כל אחת. הוא מחליף כל פקודה כזאת בפקודה CALL מתאימה, עם כל הפרמטרים הדרושים. פקודות CALL פונות לשגרות מיוחדות של מערכת RDBMS, שהן שגרות זמן ריצה (Run Time Routines) המספקות את כל השירותים לכל תוכניות היישום הפונות לבסיס הנתונים. כל פקודות SQL השייכות לתוכנית היישום עוברות עיבוד ומיטוב (ראה שלב 4) ומוכנסות לקובץ מיוחד המכיל את כל הבקשות לגישה לבסיס הנתונים של תוכנית היישום. קובץ זה נקרא בשם DataBase Request Module, ובקיצור – DBRM. קובץ כזה נוצר עבור כל תוכנית יישום.
- **צעד 2 – הידור:** תוכנית היישום המכילה פקודות C רגילות ופקודות Call מועברת למהדר הרגיל של שפת C. המהדר בודק את תקינות התוכנית ומתרגם אותה לשפת מכונה ומייצר קובץ המכיל את פקודות המכונה, Object Code. נדגיש כאן שאם לא היינו מפעילים את קדם-המהדר ומתרגמים את פקודות SQL לפקודות Call, המהדר של שפת C היה דוחה את כל פקודות SQL כפקודות לא תקינות.
- **צעד 3 – קישור לשגרות בסיס הנתונים:** פקודות המכונה עוברות תהליך מיוחד של קישור (Link) עם שגרות זמן הריצה של בסיס הנתונים. שגרות אלו מופעלות על ידי פקודות Call המופיעות בתוכנית. התוצאה היא שמתקבל קובץ המכיל **תוכנית מוכנה לביצוע** (Executable Program).
- **צעד 4 – הידור פקודות SQL**: במקביל לתהליך הידור תוכנית היישום, מתבצע הידור נוסף המתייחס לבקשות הגישה לבסיס הנתונים. יצרן מערכת RDBMS מספק את מעבד פקודות SQL. תהליך ההידור הוסבר בפרק שהציג את השפה לטיפול בנתונים

ולכן לא נחזור עליו כאן. נזכיר כאן בקצרה שכל פקודת SQL עוברת סריקה ופיענוח (Parsing), בדיקת תקינות (Validation), אופטימיזציה והכנת תוכנית הגישה. התוצאות של תהליך ההידור עבור כל פקודות SQL שנמצאו בקובץ DBRM נשמרות בקובץ מיוחד הקרוי Application Plan. פקודות SQL יכולות להכיל הפניות למשתנים של השפה המארחת. שיטה זו מאפשרת להעביר פרמטרים אל השפה המארחת וממנה.



תרשים 12.2: תהליך הידור של תוכנית שבה משובצות פקודות SQL.

שלב ג': הרצת התוכנית

בשלב ההידור הוכנו שני קבצים: קובץ הריצה של התוכנית וקובץ המכיל את תוכניות הגישה של כל פקודות SQL כפי שהוכנו על ידי מעבד SQL. נסקור עכשיו את תהליך הריצה של תוכנית היישום שבדוגמה. השלבים השונים מתוארים בתרשים 12.3.

- **צעד 1 – טעינה:** תוכנית היישום נטענת אל הזיכרון ומתחילה להתבצע.
- **צעד 2 – הזנת שם משתמש וסיסמה:** המשתמש מזין את שם המשתמש ואת הסיסמה. נתונים אלה מועברים כפרמטרים לפקודה CONNECT. נשים לב שהפקודה CONNECT הוחלפה על ידי קדם-המהדר בפקודה CALL עם הפרמטרים המתאימים.
- **צעד 3 – בדיקת הרשאה:** שגרת זמן הריצה של RDBMS מופעלת ובודקת בקטלוג המערכת אם המשתמש מוכר ואם סיסמתו מתאימה. אם הוא אינו מוכר למערכת או שהסיסמה אינה מתאימה, בקשת הגישה של תוכנית היישום לבסיס הנתונים נדחית עם הודעת שגיאה מתאימה.

- **צעד 4 – קישור לבסיס הנתונים:** אם תהליך ההזדהות הסתיים בהצלחה, מתבצע קישור של תוכנית היישום עם בסיס הנתונים. מנקודה זו ואילך בסיס הנתונים מנהל קישור (Connection) פתוח עם תוכנית היישום ועוקב אחר הפעולות שהיא מבצעת על פי ההרשאות שהוגדרו למשתמש.

- **צעד 5 – הזנת מספר סטודנט:** המשתמש מתבקש להזין את מספר הסטודנט שלו. המספר המוזן מועבר מהמסך אל המשתנה student_number שנמצא בשטח העבודה של תוכנית היישום.

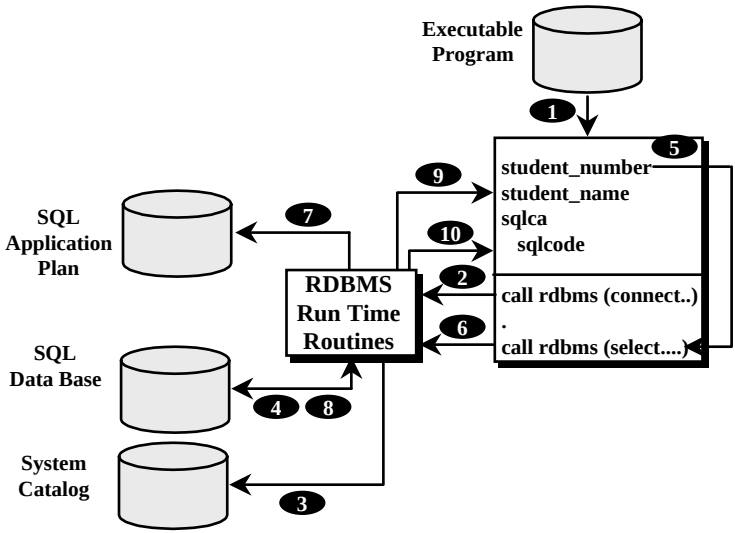
- **צעד 6 – הפעלת בקשת השליפה:** תוכנית היישום מגיעה לביצוע הפקודה Call שצריכה לבצע את שליפת הנתונים מבסיס הנתונים. נשים לב שוב, שהפקודה SELECT הוחלפה על ידי קדם-המהדר לפקודה Call עם פרמטרים מתאימים.

- **צעד 7 – שליפת תוכנית הגישה:** שגרות זמן הריצה של מערכת RDBMS מקבלות את הבקשה, הכוללת בין השאר את השם של תוכנית הגישה (Access Plan) ושולפות אותה. כזכור, תוכנית הגישה הוכנה על ידי מהדר SQL ואוחסנה בקובץ SQL Access Plan. בין שלב הכנת תוכנית הגישה לבין שלב הרצת תוכנית הגישה ייתכן ובוצעו שינויים בבסיס הנתונים שיש להם השלכה על תוכנית הגישה. למשל בוטל אינדקס שנלקח בחשבון בעת הכנת תוכנית הגישה או היה שינוי כלשהו באחת הטבלאות המשתתפות בתוכנית הגישה. שינויים אלה יכולים לגרום לכך שתוכנית הגישה אינה בת ביצוע.

כדי למנוע מצבים אלה, מערכת RDBMS עוקבת אחר כל שינוי בהגדרת בסיס הנתונים ומסמנת כל תוכנית גישה המשתמשת באובייקט כלשהו שהשתנה. עם שליפת תוכנית הגישה לצורך הפעלתה, מתבצעת בדיקה אם היה שינוי באובייקט רלוונטי. אם כן, מופעל תהליך ההידור מחדש באופן אוטומטי. מקובל לקרוא לתהליך זה Automatic SQL Recompilation. עם גמר ההידור, תוכנית הגישה המעודכנת מאוחסנת מחדש בקטלוג ולאחר מכן מופעלת. כל התהליך הזה קורה מאחורי הקלעים וללא התערבות מפתחי היישום. נדגיש שמערכת RDBMS אינה יכולה לזהות מצבים שבהם כתוצאה משינוי בהגדרת בסיס הנתונים, ניתן היה לבנות תוכנית גישה טובה יותר. למשל, הוספת אינדקס חדש יכולה לגרום לשיפור משמעותי בזמני העיבוד, אולם המערכת אינה יודעת לזהות באופן אוטומטי את תוכניות הגישה שיש להדר מחדש. כדי להנות מהשיפורים האפשריים, יש לבצע מהלך יזום של הידור מחדש של תוכניות יישום רלוונטיות.

- **צעד 8 – שלב שליפת הנתונים:** תוכנית הגישה מופעלת לאחר שמועברים אליה כל הפרמטרים המתאימים, שאחד מהם הוא מספר הסטודנט שאת נתוניו יש להציג. תוכנית הגישה פונה קודם אל קטלוג המערכת כדי לראות אם למשתמש יש הרשאה לשלוף נתונים מטבלת הסטודנטים. אם למשתמש יש הרשאה, תוכנית הגישה מבצעת את הגישה אל טבלת הסטודנטים ושולפת משם את שם הסטודנט. אם למשתמש אין הרשאה, שגרות הריצה מחזירות חזרה קוד שגיאה מתאים לתוך המשתנה sqlcode בשטח התקשורת SQLCA.

- **צעד 9 – העברת הנתונים אל תוכנית היישום:** שגרות זמן הריצה מקבלות את שם הסטודנט מתוכנית הגישה ומעבירות אותו חזרה אל המשתנה student_name הנמצא בשטח העבודה של תוכנית היישום.
- **צעד 10 – עדכון סטטוס ביצוע הפקודה:** שגרות זמן הריצה של מערכת RDBMS מעדכנות את שטח התקשורת המשותף SQLCA, עם נתונים שונים הקשורים לביצוע הפקודה האחרונה. למשל למשתנה sqlcode השיגרה מכניסה ערך מסוים המסמן שהפקודה האחרונה בוצעה בהצלחה. אם הסטודנט המבוקש לא היה נמצא בטבלה, שלב 9 לא היה מתבצע ולמשתנה sqlcode היה מוכנס ערך המצביע על כשלון ביצוע הפקודה.
- **צעד 11 – המשך ביצוע:** תוכנית היישום מקבלת שוב את הפיקוח וממשיכה לבצע את הפקודות. במקרה שלנו, שם הסטודנט כפי שהוא מופיע במשתנה student_name, היה מוצג על המסך או במקרה של כישלון היתה מופיעה על המסך הודעת שגיאה מתאימה.



תרשים 12.3: תיאור שלבי הריצה.

שטח התקשורת – SQLCA (SQL Communication Area)

מערכת RDBMS משתמשת בשטח מיוחד בזיכרון המחשב, הנקרא בשם SQLCA כדי לתקשר ולדווח לתוכנית היישום על אירועים שונים שמתרחשים תוך כדי פעולתה. לדוגמה, אם בקשת ההתקשרות לבסיס הנתונים נכשלה בגלל בעיית הרשאה, או אם בקשה להוספת שורה חדשה בטבלה נכשלה מכיון שכבר קיימת שורה עם מפתח זהה, או אם תוכנית היישום מנסה לבטל שורה מטבלה אבל הביטול יכול לגרום לבעיית שלמות הקשרים (Referential Integrity). תקן SQL1 לא הגדיר את מבנה שטח התקשורת ולכן קיימים הבדלים מסוימים בין צורת המימוש של שטח תקשורת זה במערכות מסחריות שונות.

המבנה המקובל של שטח התקשורת SQLCA מוצג בתרשים 12.4.

```
1.  struct sqlca
2.  {
3.   char   sqlcaid[8];          /* contains fixed text "SQLCA    "*/
4.   long   sqlabc;              /* the length of the SQLCA structure */
5.   long   sqlcode;             /* SQL Return Code.  */
6.          struct
7.          {
8.           short sqlerrml;     /* length of error message */
9.           long  sqlerrmc [70]; /* text of error message */
10.          } sqlerrm;
11.  char  sqlerrp[8];           /* unused */
12.  long  sqlerrd[6];           /* number of rows processed by a DML statement */
13.  char  sqlwarn[8];           /* warning flag array */
14.  char  sqlext[8];            /* extension to sqlwarn array */
15.  };
```

תרשים 12.4: מבנה שטח התקשורת SQLCA.

שילוב ההגדרה של שטח התקשורת בתוכנית מתבצע על ידי הפקודה –

EXEC SQL INCLUDE SQLCA;

פקודה זו מכניסה את ההגדרות האלו אל תוכנית היישום ללא צורך בהגדרת המשתנים שוב ושוב עבור כל תוכנית יישום.

מבין כל המשתנים המופיעים בשטח התקשורת, המשתנה sqlcode הינו השימושי ביותר. הוא מכיל את **קוד ההחזר** (Return Code) וערכו נקבע על ידי רוטינות זמן הריצה של מערכת RDBMS.

- ערך אפס מצביע על כך שפקודת SQL הסתיימה בהצלחה.
- ערך חיובי מצביע על כך שפקודת SQL הסתיימה בהצלחה, אבל התקבלה הערה. לדוגמה, אם במהלך ביצוע הפקודה היה צורך לקצץ באורך שדה כלשהו, או שבוצעה פעולת עיגול או כדומה.
- ערך שלילי מצביע על כך שפקודת SQL לא בוצעה כתוצאה משגיאה כלשהי, כמו נסיון להוסיף שורה עם מפתח כפול, נסיון לגשת לטבלה שתוכנית היישום אינה מורשית וכד'.

בנוסף לקוד ההחזר, המשתנה sqlerrmc מכיל את המלל המלא של ההודעה, כך שניתן להדפיס מלל זה כדי לקבל מושג ברור יותר של מהות ההודעה.

אי הבהירות שהיתה קיימת עד להופעת תקן SQL בכל הנוגע לטיפול במצבים חריגים גרמה לקביעת עובדות שונות על ידי יצרני מערכות RDBMS. כל יצרן קבע את המשמעות של קודי ההחזר השונים. מאחר ונושא זה הוא מהותי מאוד בעבודה עם שפת SQL מתוך שפה מארחת, נוצר קושי להביא את היצרנים השונים להסכים ביניהם על קודים אחידים. במקום לנסות ולכפות על היצרנים השונים קודים זהים, דבר שהיה עלול להביא לצורך בשכתוב תוכניות יישום, הוחלט להגדיר משתנה חדש שערכיו יוגדרו בתקן. תקן SQL2

מגדיר משתנה חדש, sqlstate, שתפקידו לקלוט את קודי המצב של בסיס הנתונים וקבע את הערכים הסטנדרטיים שלו. הערכים השונים חולקו למספר קטגוריות. המשתנה מורכב משני חלקים:

- **SQL Error Class** – מחרוזת של שני תווים המציינת את הקטגוריה של הודעת השגיאה. לדוגמה, הודעות הקשורות לבעיות של שלמות קשרים שייכות לקטגוריה 23.
- **SQL Error Subclass** – מחרוזת של שלושה תווים המגדירה את ההודעה בתוך הקטגוריה.

רוב המערכות המסחריות מנהלות כיום בתוך שטח התקשורת SQLCA את שני המשתנים sqlcode ו-sqlstate ומחזירות את ההודעות לשניהם, לראשון לפי מה שהיה מקובל עד כה ולשני – על פי התקן. יישומים חדשים יכולים להשתמש במשתנה החדש בעוד שיישומים ישנים יכולים להמשיך ולהשתמש בקודי ההודעה הישנים.

טיפול במצבי שגיאה (Error Handling)

כל פקודת SQL יכולה להסתיים בהצלחה או בכישלון ולכן תוכנית היישום צריכה לבדוק את קוד ההחזר לאחר כל פקודה. למעשה, לאחר כל פקודת SQL על מפתח היישום לכתוב אוסף פקודות לבדיקת קוד ההחזר.

```
        .
        .
1.      EXEC SQL SELECT NAME INTO :student_name
2 .                FROM    STUDENTS
3.                 WHERE STUDENT_ID = :student_number;
4.      if (sqlca.sqlcode < 0)
5.         goto sql_error;
6.      .
7.      .
8.   sql_error:
9.      printf ("SQL error: %ld\n", sqlca.sqlcode);
10.      exit();
11.     .
12.     .
```

תרשים 12.5: בדיקת קוד החזר.

הסבר:

- שורות 1 עד 3 מבצעות את הפקודה SELECT.
- שורה 4 מבצעת בדיקה לגבי ערך קוד ההחזר. אם הקוד הוא שלילי, הפיקוח מועבר אל רוטינת הטיפול בשגיאות המתחילה בשורה 8.
- שורות 8 ועד 10 מכילות שיגרה המציגה את קוד ההחזר.

בדיקת קוד ההחזר לאחר כל פקודת SQL יכולה להיות מייגעת מאוד מנקודת המבט של מפתח היישום והמתכנת. שפת SQL מספקת את הפקודה EXEC SQL WHENEVER שמטרתה לפשט את הטיפול במצבי השגיאה. זוהי פקודה ייחודית לעבודה בשיטה של שיבוץ פקודות SQL בשפה מארחת. היא מורה לקדם-המהדר לשתול מייד לאחר כל פקודת SQL את הפקודות לבדיקת קוד ההחזר ולהעביר את הפיקוח למקרה שקוד ההחזר שונה מאפס.

תחביר הפקודה:

תרשים 12.6: תרשים תחביר של הפקודה WHENEVER.

ניתן לכתוב את הפקודה בכל מקום בתוך תוכנית היישום. אם הפקודה מופיעה יותר מאשר פעם אחת, התוקף של כל פקודה הוא עד להופעת פקודה נוספת. הפרמטרים של הפקודה הם:

- NOT FOUND – מנחה את קדם-המהדר לשתול לאחר כל פקודת SQL בדיקה וטיפול במקרה שלא נמצאו שורות בפקודה SELECT.
- SQLERROR – מנחה את קדם-המהדר לשתול לאחר כל פקודת SQL בדיקה וטיפול במקרה שערך קוד ההחזר היה שלילי.
- SQLWARNING – מנחה את קדם-המהדר לשתול לאחר כל פקודת SQL בדיקה וטיפול במקרה שערך קוד ההחזר היה חיובי.

בהמשך, לאחד משלושת הפרמטרים ניתן לבקש העברת פיקוח לשיגרה ששמה מופיע בתווית (label), או להעביר את הפיקוח לפקודה הבאה בתוכנית.

דוגמה : התוכנית שבתרשים 12.7 מדגימה שימוש בפקודה WHENEVER וכיצד ניתן לכתוב אותה מספר פעמים בתוך התוכנית. מרגע שכותבים פקודה זו, הפרמטרים שלה בתוקף עד למופע הבא שלה.

הסבר:

- בשורה 1 מופיעה הפקודה WHENEVER שמנחה את קדם-המהדר להכניס פקודות לבדיקת קוד החזר שלילי והעברת הפיקוח לשיגרה sql_update_error. קדם-המהדר יכניס את הבדיקה לאחר פקודות SQL שמתחילות בשורות 2 ו-5.
- בשורה 9 מופיעה הפקודה WHENEVER חדשה ולכן המופע הקודם בטל ועכשיו קדם-המהדר יכניס פקודות להעברת הפיקוח לשיגרה אחרת.
- לאחר פקודת SQL שבשורה 10 תוכנס בדיקה שתעביר את הפיקוח לשיגרה sql_delete_error במקרה של קוד החזר שלילי.

```
.
EXEC SQL WHENEVER SQLERROR GOTO sql_update_error;
EXEC SQL UPDATE STUDENTS
            SET CITY = :city
            WHERE STUDENT_ID = :student_number;
EXEC SQL UPDATE GRADES
            SET GRADE = :grade + 5
            WHERE STUDENT_ID = :student_number AND
                    COURSE_ID = :course_id;
EXEC SQL WHENEVER SQLERROR GOTO sql_delete_error;
EXEC SQL DELETE STUDENTS
            WHERE STUDENT_ID = :student_number;
.
.
sql_update_error:
  printf ("SQL update error: %ld\n", sqlca.sqlcode);
  exit();
sql_delete_error:
  printf ("SQL delete error: %ld\n", sqlca.sqlcode);
  exit();
.
```

תרשים 12.7: הפקודה WHENEVER.

למעשה, לאחר קדם-ההידור התוכנית תיראה כך:

```
EXEC SQL UPDATE STUDENTS
            SET CITY = :city
            WHERE STUDENT_ID = :student_number;
 if (sqlca.sqlcode < 0)
    goto sql_update_error;
EXEC SQL UPDATE GRADES
            SET GRADE = :grade + 5
            WHERE STUDENT_ID = :student_number
                    AND COURSE_ID = :course_id;
 if (sqlca.sqlcode < 0)
    goto sql_update_error;
EXEC SQL DELETE STUDENTS
            WHERE STUDENT_ID = :student_number;
 if (sqlca.sqlcode < 0)
    goto sql_delete_error;
.
.
sql_update_error:
  printf ("SQL update error: %ld\n", sqlca.sqlcode);
  exit();
sql_delete_error:
  printf ("SQL delete error: %ld\n", sqlca.sqlcode);
  exit();
```

תרשים 12.8: תוכנית עם WHENEVER לאחר הפעלת קדם-המהדר.

נשים לב שפקודות WHENEVER נעלמו ובמקומן התווספו פקודות חדשות בשורות 5,6 בשורות 11,12 ובשורות 15,16. בצורה זו, יכול מפתח היישום לפזר פקודות WHENEVER בתוכנית ולחסוך לעצמו כתיבה של שורות רבות וכך להפוך את התוכנית לקריאה יותר.

עבודה עם משתני השפה המארחת (Host Language Variables)

ניתן להשתמש במשתנים של השפה המארחת בתוך פקודות SQL. שילוב משתנים בתוך פקודות SQL הוא הדרך להעביר נתונים מתוך השפה המארחת אל פקודות SQL ומפקודות SQL חזרה אל השפה המארחת. משתנים אלה יכולים להופיע בתוך פקודות SQL בכל מקום שבו מותר לקבוע (Constant) להופיע. מאחר והשיטה היא שיטה סטטית, אסור להשתמש במשתנים כדי לקבוע את שמות הטבלאות והעמודות. כפי שנראה בהמשך, אם יש צורך בבניית פקודות SQL תוך כדי ריצת התוכנית, ניתן להשתמש בשיטה הדינמית.

כל משתנה שיכול להשתתף בתוך פקודת SQL חייב להיות מוצהר מראש בתוך קטע המתחיל בפקודה BEGIN DECLARE ומסתיים בפקודה END DECLARE. משתנה של השפה המארחת המופיע בתוך פקודת SQL יופיע עם קידומת נקודותיים (:) לפני שם המשתנה.

בדוגמה הבאה אנו רואים כיצד המשתנים student_number ו- student_name משולבים בתוך הפקודה SELECT. נוכל לראות שמשתנים אלה מוגדרים בתחילת התוכנית על ידי הפקודה DECLARE.

```
1.    EXEC SQL BEGIN DECLARE SECTION;
2.    varchar  userid(20);
3.    varchar  password (20);
4.    varchar  student_number (5);
5.    varchar  student_name (20);
6.    EXEC SQL END DECLARE SECTION;
7.
8.  main()
9.  {
10.     printf ("\n Enter Student Number: ");
11.     scanf ("%s", student_number);
12.
13.     EXEC SQL SELECT NAME INTO :student_name
14.                  FROM    STUDENTS
15.                  WHERE STUDENT_ID = :student_number;
16.
17.   }
```

תרשים 12.9: משתני תוכנית משולבים בפקודה SELECT.

נשים לב שתוצאת השאילתה מוכנסת אל משתנה של התוכנית המארחת באמצעות הפקודה INTO במשפט SELECT. מאחר ושפת SQL תומכת במספר טיפוסי נתונים שאינם בהכרח טיפוסי הנתונים הנתמכים על ידי השפה המארחת, יש לתת תשומת לב מיוחדת לתאימות זו. לדוגמה, טיפוס SMALLINT בשפת SQL מקביל לטיפוס SHORT בשפת C בעוד טיפוס INTEGER בשפת SQL מקביל ל-INT או ל-LONG בשפת C.

טיפול בערכים חסרים בתוך שפה מארחת

תשומת לב מיוחדת ניתנת לטיפול ב**ערכים חסרים** (Null Values), מאחר ורוב שפות התכנות אינן תומכות במושג זה. כדי להתמודד עם בעיה זו מספקת שפת SQL משתנה מיוחד, משתנה שנקרא Indicator Variable. לכל משתנה שמוצהר על ידי הפקודה DECLARE ניתן להצמייד משתנה נוסף בעל טיפוס נתונים שלם קצר (short בשפת C). המשתנה הזה יכול לקבל ערך אפס, ערך חיובי או ערך שלילי והמשמעות היא:

- **ערך אפס** – המשמעות היא שהמשתנה מכיל ערך תקין כלשהו.
- **ערך שלילי** – המשמעות היא שהמשתנה מכיל ערך חסר (Null).
- **ערך חיובי** – המשמעות היא שהמשתנה מכיל ערך תקין שעבר קיצוץ או עיגול משום שלא היה מספיק מקום לאחסן את הערך ש-SQL העביר אליו.

דוגמה: יש לכתוב תוכנית שתאפשר להכניס ערך Null לעיר המגורים של הסטודנט.

```
1.    EXEC SQL BEGIN DECLARE SECTION;
2.    varchar  userid(20);
3.    varchar  password (20);
4.    varchar  student_number (5);
5.    varchar  student_city (20);
6.    short    student_city_ind;
7.    EXEC SQL END DECLARE SECTION;
8.
9.  main()
10. {
11.    printf ("\n Enter Student Number: ");
12.    scanf ("%s", student_number);
13.    student_city_ind = -1;
14.    EXEC SQL UPDATE STUDENTS
15.                SET CITY = :student_city :student_city_ind
16.                WHERE STUDENT_ID = :student_number;
17.
18. }
```

תרשים 12.10: תוכנית הקולטת ערכים חסרים.

הסבר:

- בשורה 6 מוגדר המשתנה הנומרי student_city_ind מייד לאחר הצהרת המשתנה student_name. אגב, לא חובה להצמייד אותם אחד ליד השני בשלב ההגדרה. זו פשוט דרך נוחה יותר והתוכנית הופכת לקריאה יותר אם מקפידים על שיטה זו.
- בשורה 13 מוכנס ערך שלילי כלשהו לתוך המשתנה student_city_ind, כלומר המשמעות היא שבעת ביצוע פקודת העדכון, עיר המגורים תקבל ערך Null.
- בשורה 15 מופיע ליד המשתנה student_city גם המשתנה student_city_ind ללא פסיק ביניהם. אי הופעת הפסיק בין שני המשתנים היא הגורמת לכך שהמהדר יתייחס אל המשתנה השני כאל Indicator Variable. מאחר והמשתנה student_city_ind מכיל ערך

שלילי, מערכת RDBMS תכניס לעיר המגורים של הסטודנט את הערך Null. למעשה הפקודה שהמערכת תבצע היא:

```
14.   EXEC SQL UPDATE STUDENTS
15.                SET CITY =  NULL
16.                WHERE STUDENT_ID = :student_number;
```

אם המשתנה student_city_ind היה מכיל ערך חיובי, הפקודה שהיתה מתבצעת:

```
14.   EXEC SQL UPDATE STUDENTS
15.                SET CITY =  :student_city
16.                WHERE STUDENT_ID = :student_number;
```

גם בעת שליפת שורות מבסיס הנתונים ניתן להשתמש במשתנים מיוחדים אלה כדי לבדוק אם עמודה מסוימת מכילה ערך חסר.

דוגמה: יש לכתוב תוכנית המאפשרת שליפת עיר המגורים של סטודנט כלשהו מטבלת סטודנטים. אם עיר המגורים של הסטודנט המבוקש מכילה Null, התוכנית צריכה להדפיס הודעה מיוחדת; אחרת, התוכנית תציג את עיר המגורים. ראה תרשים 12.11.

```
1.    EXEC SQL BEGIN DECLARE SECTION;
2.    varchar  userid(20);
3.    varchar password (20);
4.    varchar  student_number (5);
5.    varchar  student_name (20;
6.    varchar  student_city (20);
7.    short      student_city_ind;
8.    EXEC SQL END DECLARE SECTION;
9.
10.  main()
11.  {
12.     EXEC SQL WHENEVER NOTFOUND GOTO sql_not_found;
13.     printf ("\n Enter Student Number: ");
14.     scanf ("%s", student_number);
15.     EXEC SQL SELECT NAME INTO
16.                        :student_name, :student_city :student_city_ind
17.                    FROM     STUDENTS
18.                    WHERE STUDENT_ID = :student_number;
19.    if (student_city_ind < 0)
20.       printf ("City missing for the student %f\n", student_number);
21.       else
22.       printf ("Name: ", student_name, "City: %f\n", student_city);
23.   sql_not_found:
24.      printf ("Student not found: %ld\n", student_number);
25.     exit();
26.   }
```

תרשים 12.11: תוכנית המגיבה לערכים חסרים.

הסבר:

- בשורה 16 ניתן לראות שליד המשתנה student_city הוספנו את המשתנה המיוחד student_city_ind, שוב ללא פסיק ביניהם.
- בשורה 19 מתבצעת בדיקת ערך המשתנה student_city_ind. אם ערכו שלילי, כלומר העמודה student_city מכילה Null, נדפיס הודעה מיוחדת כפי שמופיעה בשורה 20 אחרת נדפיס את שם הסטודנט ואת עיר המגורים כפי שמופיע בשורה 22 .
- אם לא נמצאה שורה עם מספר סטודנט כמבוקש, השיגרה המופיעה בשורות 23 עד 25 תופעל בגלל השימוש בפקודה WHENEVER המופיעה בשורה 12.

נראה עכשיו דוגמה לשימוש במשתנים מיוחדים אלה לציון ערכים חסרים, בתהליך של הוספת שורה חדשה לטבלת סטודנטים.

דוגמה : יש לבנות תוכנית להוספת שורה חדשה לטבלת סטודנטים. שם הסטודנט ועיר המגורים יכולים לקבל ערך Null. למען הפשטות, נציג רק את הגדרות המשתנים ואת פקודת ההוספה.

```
1 .    EXEC SQL BEGIN DECLARE SECTION;
2.     varchar  student_number (5);
3.     varchar  student_name (20);
4.     short      student_name_ind;
5 .    varchar  student_city (20);
6.     short      student_city_ind ;
7.     EXEC SQL END DECLARE SECTION;
8 .
9 .  main()
10.  {
11.      EXEC SQL INSERT INTO STUDENTS (STUDENT_ID, NAME, CITY)
12.          VALUES (:student_id, :student_name :student_name_ind,
13.                      :student_city : student_city_ind);
14 .  }
```

תרשים 12.12: תוכנית להוספת שורה עם ערכים חסרים

הסבר:

- בשורות 4 ו-6 הגדרנו את המשתנים המיוחדים שתפקידם לטפל ב-Null.
- בשורות 11 עד 13 מופיעה פקודת ההוספה של שורה חדשה לבסיס הנתונים. ליד כל משתנה רגיל מופיע המשתנה המיוחד ללא הפרדה של פסיק ביניהם.

כמובן שתוכנית היישום צריכה לקבוע את הערכים של משתנים אלה לפני ביצוע פקודת ההוספה. אם יש להכניס Null לשם הסטודנט ולעיר המגורים, יש לקבוע ערך שלילי כלשהו למשתנים student_name_ind ו- student_city_ind. במצב זה, מערכת RDBMS תתעלם מתוכן המשתנים הרגילים ותכניס Null אל השורות בבסיס הנתונים. אם רוצים להכניס ערכים רגילים, יש לדאוג ששני המשתנים המיוחדים יכילו ערך חיובי כלשהו.

שאילתות מרובות שורות (Multi-row Queries)

עד כאן השתמשנו בדוגמאות של פקודות SELECT השולפות שורה בודדת בלבד. שאילתות אלו, Single Row Queries, אינן מהוות בעיה מיוחדת. כל העמודות המופיעות במשפט SELECT מוכנסות אל תוך משתנים של השפה המאחרת תוך שימוש במשפט INTO. בעיה מיוחדת מתעוררת כאשר השאילתה עשויה להחזיר מספר בלתי ידוע מראש של שורות. כאן נוצרת בעיה: שפת התכנות לא יודעת להכין מראש שטח למספר בלתי ידוע של שורות. גם אם מספר השורות היה ידוע,עדיין נשאלת השאלה כיצד תוכנית היישום תעבד בבת אחת מספר רב של שורות. זו למעשה הדוגמה הקלאסית להבדל בין תוכנית יישום הפועלת בשיטת One-Record-at-a-Time לבין מערכת RDBMS הפועלת בשיטת One-Set-at-a-Time.

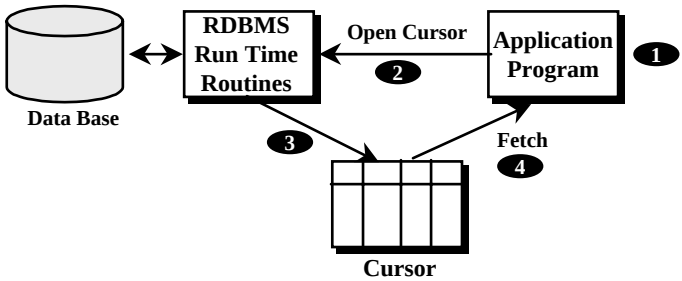
הפתרון לבעיה זו דורש הגדרת מנגנון **סמן** מיוחד, **Cursor**, שמאפשר תקשורת בין שפות פרוצדורליות כמו C, COBOL ואחרות לבין שפת SQL. ניתן להתייחס אל הסמן, ה-Cursor, כאל טבלה מדומה המכילה את טבלת התוצאה. טבלה זו מנוהלת על ידי מערכת RDBMS ותוכנית היישום יכולה לשלוף ממנה בכל פעם שורה אחת באמצעות הפקודה FETCH.

התנהגות הסמן מזכירה מאוד עבודה עם קובץ רגיל: פותחים אותו באמצעות Open, קוראים ממנו באמצעות Read וסוגרים אותו עם Close. בדיוק כמו שכל פקודת Read זוכרת את הרשומה האחרונה שקראה, כלומר יודעת לנהל את מקומה היחסי בקובץ (Current Position), כך גם הפקודה FETCH זוכרת את מקומה היחסי בתוך הסמן.

עבודה עם הסמן – Cursor

סמן Cursor	***סמן* ב-SQL מוגדר כטבלת התוצאה של שאילתה מרובת שורות, וגם כמצביע על שורת הפלט התורנית בטבלת פלט זו.**

תרשים 12.13 מציג את עקרונות העבודה עם שאילתות מרובות שורות.



תרשים 12.13: עיקרון העבודה עם Cursor.

- **שלב 1 – הגדרת Cursor**: תוכנית היישום משתמשת בפקודה DECLARE CURSOR כדי להכריז שהיא מתכוונת להשתמש בשאילתה מרובת שורות. הפקודה מכילה את השאילתה, כלומר את הפקודה SELECT. תוכנית היישום יכולה להצהיר על מספר בלתי מוגבל של סמנים.
- **שלב 2 – הפעלת השאילתה:** להפעלת השאילתה, תוכנית היישום צריכה לפתוח את הסמן באמצעות הפקודה OPEN CURSOR. ביצוע הפקודה OPEN CURSOR גורמת למערכת RDBMS לבצע את השאילתה. תוכנית היישום יכולה לפתוח מספר סמנים, זה אחר זה לפני שהיא מתחילה לשלוף מהם שורות.
- **שלב 3 – בניית Cursor**: את השורות שמערכת RDBMS שולפת היא מעבירה אל הטבלה המיוחדת של **הסמן**, ה-**Cursor**. צורת מימוש העבודה עם הסמן שונה ממערכת למערכת ואינה מחייבת דווקא ניהול של טבלה מיוחדת, כפי שיכול להתקבל הרושם מתיאור לוגי זה. עקרונית, מערכת RDBMS יכולה לבצע את השאילתה ולשמור את תוצאותיה בשטחי העבודה שלה וליצור אשליה של טבלת Cursor המכילה את הטבלה התוצאתית.

לדוגמה, אם השאילתה סורקת שורות של טבלאות, מערכת RDBMS יכולה להחזיר לתוכנית היישום שורה אחר שורה ללא צורך בבניית טבלת ביניים. כמובן ששיטת העבודה עם שאילתות מורכבות הכוללות תת-שאילתות, פונקציות מובנות, הקבצות ומיון הינה מורכבת יותר ודורשת בחלק גדול מהמקרים לייצר את הטבלה המדומה לפני שניתן להתחיל להעביר שורות לתוכנית היישום. אחת הסיבות לשם המיוחד, Cursor, שמשמעותו סמן או מצביע, נובעת מהאשליה שפקודות FETCH למעשה מצביעות כל פעם על שורה חדשה בטבלת התוצאה.

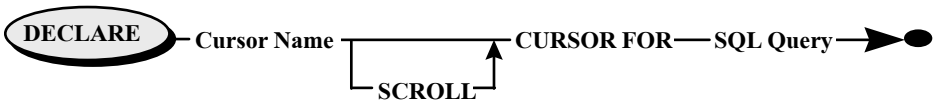
- **שלב 4 – שליפת השורות מה-Cursor:** לאחר שטבלת הסמן כבר מוכנה, תוכנית היישום יכולה לבקש באופן סדרתי שורה אחר שורה, באמצעות הפקודה FETCH. אלו הן השורות שנמצאות בטבלה Cursor. מכיון שפקודה זו מתייחסת לסמן (Cursor) מסוים, התוכנית יכולה לבצע פקודות כאלו על מספר סמנים, ולא על אחד בלבד.
- **שלב 5 – סגירת Cursor**: בגמר קריאת השורות מתוך טבלת הסמן (Cursor), תוכנית היישום סוגרת את הטבלה. נדגיש שמכיון שטבלה זו יכולה לתפוס משאבי מחשוב רבים, מומלץ מאוד לשחרר אותם מייד בגמר השימוש, כדי לנצל באופן יעיל את סביבת המחשוב.

נעבור עכשיו באופן שיטתי על הפקודות השונות לעבודה עם שאילתות מרובות שורות תוך שימוש במנגנון הסמן – Cursor.

הפקודה DECLARE להגדרת Cursor

פקודה זו מגדירה למעשה את השאילתה. לכל סמן (Cursor) יש לתת שם לוגי חד-ערכי, כך שתוכנית היישום תוכל לפנות לכל אחד מהם בנפרד.

תחביר הפקודה:



תרשים 12.14: תרשים תחביר לפקודה DECLARE CURSOR.

תקן SQL1 התייחס רק למצב שבו ניתן לסרוק את טבלת הסמן בכיוון אחד, קדימה. לעומת זאת, תקן SQL2 מאפשר לנוע בתוך טבלת הסמן קדימה, אחורה, לקפוץ ישר לסוף, להתקדם כמה שורות בבת אחד ועוד. אם מבקשים לאפשר תנועה בתוך טבלת הסמן יש להכריז על כך מראש על ידי שימוש בפרמטר SCROLL (גלגול). נדגיש שלא כל המערכות המסחריות תומכות בתכונה זו. מימוש התכונה Scroll Cursor מורכב יותר ודורש משאבי מחשוב גדולים יותר, ולכן לא כל המערכות המסחריות תומכות בתכונה זו. גם במערכות התומכות בתכונה זו, יש להשתמש בפרמטר זה רק במידת הצורך, ולא כברירת מחדל.

שאילתת SQL המופיעה כאן היא שאילתה רגילה שיכולה להכיל את כל מרכיבי פקודת SELECT רגילה. השאילתה יכולה להכיל פנייה למשתנים של השפה המארחת. חובה שהפקודה DECLARE תבוצע על ידי תוכנית היישום לפני שמגיעים לפתיחת סמן. מסיבה זו מומלץ מאוד לכתוב את הפקודה צמוד ולפני הפקודה OPEN CURSOR, כדי להבטיח שתחילה יוגדר הסמן ואחר כך הוא ייפתח.

דוגמה : יש לכתוב תוכנית לשליפת כל הציונים של הסטודנטים הגרים בעיר מסוימת ושיש להם ציונים בין 60 ל-80.

```
1.  EXEC SQL DECLARE student_grades CURSOR FOR
2.      SELECT G.STUDENT_ID, S.NAME,
3.              COURSE_ID, SEMESTER, TERM, GRADE
4.      FROM   GRADES G, STUDENTS S
5.      WHERE G.STUDENT_ID = G.STUDENT_ID AND
6.              GRADE BETWEEN 60 AND 80 AND
7.              CITY = :city_name;
```

תרשים 12.15: שימוש בפקודה DECLARE.

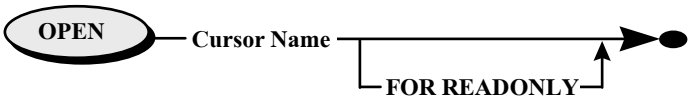
הסבר:

- שורות 1 עד 7 מכילות את הגדרת ה-Cursor. נשים לב, שהגדרת השאילתה נעשית בצורה רגילה. השם הלוגי של הסמן הוא student_grade.
- בשורה 7 יש פנייה למשתנה של השפה המארחת, משתנה שיקבל את ערכו תוך כדי ריצת התוכנית.

הפקודה OPEN לפתיחת Cursor

למרות ששם הפקודה מרמז שהיא פותחת את הסמן בלבד, היא גם מבצעת את השאילתה המתאימה. במובן הזה, פקודה זו היא המקבילה האנטראקטיבית של שיגור פקודת SQL מהמסך אל מערכת RDBMS. מכיון שפקודה זו מפעילה את השאילתה, הביצוע שלה יקח זמן רב יותר מאשר זה של הפקודה DECLARE שרק מגדירה את השאילתה וגורמת למהדר SQL להכין את תוכנית הגישה.

תחביר הפקודה:



תרשים 12.16: תרשים תחביר לפקודה OPEN

ברירת המחדל של מערכת RDBMS היא שמותר לעדכן באמצעות הסמן את בסיס הנתונים. מכיון שמצב זה דורש נעילה של שורות בתוך הטבלה (בנושא זה נדון בהמשך) השיטה דורשת משאבי מחשוב רבים יותר ויכולה לגרום להקטנת מספר המשתמשים שיכולים לעבוד בו-זמנית עם הטבלאות. הפרמטר FOR READ ONLY מאפשר להכריז מראש שה-Cursor מיועד לקריאה בלבד ולכן אין צורך בנעילות תוך כדי סריקת השורות.

דוגמה : פתיחת Cursor שהוגדר בדוגמה הקודמת לצורך קריאה בלבד.

```
1.  EXEC SQL DECLARE student_grades CURSOR FOR
2.      SELECT G.STUDENT_ID, S.NAME,
3.             COURSE_ID, SEMESTER, TERM, GRADE
4.      FROM   GRADES G, STUDENTS S
5.      WHERE G.STUDENT_ID = G.STUDENT_ID AND
6.            GRADE BETWEEN 60 AND 80 AND
7.            CITY = :city_name;
8.  EXEC SQL OPEN student_grades FOR READONLY
```

תרשים 12.17: פתיחת Cursor לקריאה בלבד.

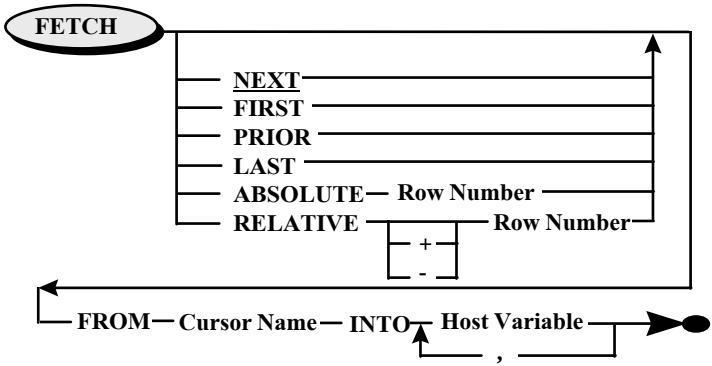
הסבר:

- שורות 1 עד 7 מגדירות את ה-Cursor.
- שורה 8 פותחת את ה-Cursor ולמעשה – מבצעת את השאילתה. נשים לב ש-Cursor נפתח לקריאה בלבד.

הפקודה FETCH לשליפת שורות מתוך Cursor

פקודה זו שולפת שורה בודדת מתוך Cursor. כל העמודות המופיעות במשפט SELECT מוכנסות אל תוך המשתנים של השפה המארחת.

תחביר הפקודה:



תרשים 12.18: תרשים תחביר לפקודה FETCH עם תמיכה ב- Scroll Cursor.

פקודה זו מבוססת על תקן SQL2 התומך ב- Scroll Cursor, כלומר קריאה קדימה או אחורה בתוך טבלת הסמן. ברירת המחדל היא NEXT שפירושה שליפת השורה הבאה שלאחר השורה האחרונה שנשלפה. מייד לאחר פתיחת Cursor, הסמן המסמן איזו שורה נשלפה עומד לפני השורה הראשונה. כדי לקבל את השורה הראשונה, יש לבצע את הפקודה FETCH פעם אחת. ביצוע הפקודה FETCH לאחר שהשורה האחרונה כבר נשלפה, מפעילה את המצב NOT FOUND.

הפרמטר FIRST מאפשר גישה מהירה לשורה הראשונה, ואילו הפרמטר LAST מאפשר גישה מהירה לשורה האחרונה. הפרמטר PRIOR מאפשר שליפה של השורה הקודמת יחסית לשורה האחרונה שנשלפה. הפרמטר ABSOLUTE מאפשר להגיע לשורה מסוימת, למשל לשורה העשירית בתוך ה-Cursor. הפרמטר RELATIVE מאפשר גלגול לפנים או לאחור של מספר שורות באופן יחסי לשורה האחרונה שנשלפה.

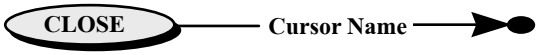
מערכות מסחריות שאינן תומכות בתכונה זו, תומכות בגירסה מצומצמת של פקודה זו.

תרשים 12.19: תרשים תחביר לפקודה FETCH ללא תמיכה ב- Scroll Cursor.

הפקודה CLOSE לסגירת Cursor

פקודה זו סוגרת את הסמן, ה-Cursor, ומשחררת את כל המשאבים שהיו תפוסים לביצוע השאילתה. יש לבצע פקודה זו רק לאחר פקודת OPEN. עם ביצוע פקודת הסגירה, פתיחה מחדש של ה-Cursor על ידי הפקודה OPEN תביא לביצוע מחדש של השאילתה. לכן יש לסגור Cursor רק לאחר שתוכנית היישום גמרה לטפל בשאילתה. אם תוכנית היישום אינה מבצעת בצורה מפורשת הפקודה CLOSE, פקודה זו תופעל באופן אוטומטי עם סיום התוכנית.

תחביר הפקודה:



תרשים 12.20: תרשים תחביר לפקודה CLOSE.

דוגמה להפעלת שאילתה מרובת שורות

דוגמה : יש לכתוב תוכנית השולפת את השורות של ציוני הסטודנטים שגרים בעיר מסוימת והציונים שלהם בין 60 ל-80. תוכנית זו תתבסס על הפקודות שהוצגו עד כה לטיפול בשאילתות מרובות שורות. למען הנוחות, נפריד בין הקטע העוסק בהגדרות לבין הקטע המבצע את הלוגיקה.

הקטע שבתרשים 12.21 מציג את ההגדרות של המשתנים השונים בהם נשתמש בתוכנית. תרשים 12.22 מציג את התוכנית עצמה. מספור השורות בתוכנית מתחיל ב-20 למען הסדר בלבד.

```
1 .   #include <stdio>
2.    # include <ctype.h>
3.    EXEC SQL BEGIN DECLARE SECTION;
4.      varchar  userid(20);
5.      varchar  password (20 );
6 .     varchar  student_number (5);
7.      varchar  student_name (20);
8.      varchar  course_number(10);
9.      varchar semester(7);
10 .    varchar term(1 );
11      short     grade;
12.     varchar student_city (10);
13.   EXEC SQL END DECLARE SECTION;
14 .  EXEC SQL INCLUDE SQLCA;
```

תרשים 12.21: דוגמה לקטע תוכנית הכולל את ההגדרות.

הסבר לתוכנית שבתרשים 12.22:

- שורות 22 עד 27 מטפלות בקבלת שם המשתמש והסיסמה ובהתקשרות אל בסיס הנתונים.
- שורות 28 ו-29 קולטות מהמשתמש את שם העיר המבוקשת.
- שורות 32 עד 38 מגדירות את השאילתה עצמה, כלומר את הסמן.
- שורה 39 מבצעת את השאילתה. נשים לב שבשורה 31 הגדרנו מצב שגיאה שיגרום לכך שאם ביצוע השאילתה נכשל בגלל NOT FOUND, התוכנית תעבור לשיגרה המדפיסה הודעת שגיאה מתאימה על המסך ועוצרת את ביצוע התוכנית.
- שורה 40 משנה את התנאי של NOT FOUND כך שבמהלך סריקת השורות הדלקת תנאי זה תביא לקפיצה לשגרת סיום הסריקה, שבה מתבצעת סגירת Cursor.
- שורות 42 ועד 44 מבצעות את שליפת השורה מתוך השאילתה ומעבירה את התוצאה אל משתני התוכנית.
- שורה 45 מדפיסה את התוצאה.

לא פירטנו כאן את הדפסת כל המשתנים ואת עריכת התוצאה. נשים לב שפקודות השליפה והעריכה של התוצאה מתבצעות בתוך לולאה, שתמשיך להתבצע עד שנגיע להפעלת התנאי NOT FOUND, לאחר ניסיון לשליפת שורה לאחר השורה האחרונה.

```
20 .  main()
21.   {
22.      printf (“ Enter user name:“;
23.      scanf (“%s”, userid);
24.      printf (“ \n Enter password:“);
25 .     scanf (“%s”, password);
26.      EXEC SQL WHENEVER SQLERROR GOTO sql_error;
27.      EXEC SQL CONNECT :userid IDENTIFIED BY :password;
28 .     printf (“ \n Enter City Name:“);
29 .     scanf (“%s”, student_city);
30.      EXEC SQL WHENEVER SQLERROR GOTO sql_error;
31 .     EXEC SQL WHENEVER NOT FOUND GOTO city_not_found;
32.      EXEC SQL DECLARE student_grades CURSOR FOR
33 .       SELECT G.STUDENT_ID, S.NAME, COURSE_ID,
34 .                  SEMESTER, TERM, GRADE
35 .      FROM GRADES G, STUDENTS S
36.       WHERE G.STUDENT_ID = G.STUDENT_ID AND
37 .                    GRADE BETWEEN60  AND80 AND
38 .                     CITY = :city_name;
39.      EXEC SQL OPEN student_grades FOR READONLY;
40.      EXEC SQL WHENEVER NOT FOUND GOTO end_of_query;
41 .     for (; ; ) {
42.        EXEC SQL FETCH student_grades
43 .                      INTO :student_number, :student_name,
44 .                              :course_number,    :semester,    :term,   :grade;
45 .       printf (“Student Data::  %s\n“, student_name, .........);
46.                }
47.      exit()
48 .   sql_error:
49 .      printf (“SQL error: %ld\n”, sqlca.sqlcode);
50.       exit();
51.    city_not_found:
52 .      printf (“City not found. \n”);
53 .      exit();
54.    end_of_query:
55.       EXEC SQL CLOSE student_grades;
56.       EXEC SQL DISCONNECT;
57.       exit ()
58.    }
```

תרשים 12.22: דוגמה לתהליך קליטת הדרישה וביצוע שאילתה מתוך תוכנית.

שימוש ב-Cursor לעדכון בסיס הנתונים

עדכון בסיס הנתונים יכול להתבצע על ידי שיבוץ פקודות עדכון רגילות בתוך תוכנית יישום. לפעמים יש צורך לבצע עדכון תוך כדי סריקת Cursor. עד כאן התייחסנו ל-Cursor לקריאה בלבד. במידת הצורך ניתן גם לבצע עדכון שורות או ביטול שורות באמצעותו, תוך הכרזה מראש שהוא בר-עדכון ותוך התחשבות במספר מגבלות. אין משמעות להוספת שורות דרך Cursor, מאחר וניתן להוסיף אותן באופן ישיר אל טבלאות הבסיס.

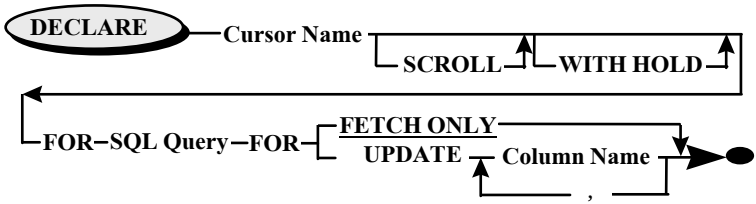
פקודת הפתיחה של ה-Cursor קובעת אם הוא ישמש לקריאה בלבד או שניתן יהיה גם לעדכן באמצעותו את בסיס הנתונים. מקובל לקרוא ל-Cursor שהוא בר-עדכון, גם בשם

Updatable Cursor. קיימות מגבלות לעדכון דרך Cursor, המזכירות את המגבלות לעדכון בסיס הנתונים דרך טבלאות מדומות. המגבלות הן:

- מתייחס לטבלה בודדת.
- אינו מכיל מיון.
- אינו מכיל הקבצות (HAVING ו- GROUP BY).
- אינו מבקש ביטול שורות כפולות, כלומר אין שימוש בפרמטר DISTINCT.

מגבלות אלו הוגדרו על ידי תקן SQL1. מערכות מסחריות מסוימות וביניהן DB2 מחייבות שבנוסף למגבלות אלו, על התוכנית להצהיר בעת פתיחת Cursor שהוא בר-עדכון, כולל התייחסות לעמודות שמותר לעדכן. תקן SQL2 מניח לעומת זאת שכל Cursor שנפתח הוא בר-עדכון אלא אם הוא הוגדר לקריאה בלבד. כדי לאפשר עדכון דרך Cursor יש לבצע מספר שינויים בפקודות הרגילות שהוצגו עד כה. נציג את השינויים העיקריים בפקודות אלו:

- **הפקודה DECLARE** – בפקודה המגדירה את השאילתה יש להוסיף את שמות העמודות שמותר לעדכן דרך ה-Cursor. חובה שעמודות אלו תופענה ברשימת העמודות הנשלפות והמוגדרות במשפט SELECT של השאילתה.



תרשים 12.23: תרשים תחביר לפקודה DECLARE עם אפשרות לעדכון.

את הסבר הפרמטר WITH HOLD נדחה לפרק העוסק בתנועות. הדוגמה הבאה מציגה הגדרה של Cursor בשם student_grades שמציג את הנתונים של סטודנט מסוים מתוך טבלת ציונים. ההגדרה מאפשרת לעדכן את הציון במהלך סריקת הטבלה.

```
1.  EXEC SQL DECLARE student_grades CURSOR FOR
2.          SELECT STUDENT_ID, COURSE_ID, SEMESTER, TERM, GRADE
3.          FROM    GRADES
4.          WHERE  STUDENT_ID = :student_number
5.          FOR UPDATE OF GRADE;
```

- **הפקודה UPDATE** – בפקודה זו, המשמשת לעדכון שורות בבסיס הנתונים, יש להוסיף במשפט WHERE את הפרמטר CURRENT OF המצביעה על העובדה שיש לעדכן את השורה האחרונה שנשלפה מתוך ה-Cursor על ידי הפקודה FETCH האחרונה. נשים לב שלעומת פקודת עדכון רגילה בשפת SQL שיכולה לעדכן בבת אחת מספר רב של שורות, הגירסה של הפקודה UPDATE המשמשת לעדכון Cursor,

מאפשרת עדכון של שורה בודדת בלבד. המערכת תאפשר עדכון רק של העמודות שמותר לעדכן על פי ההגדרה בפקודה DECLARE.

```
1.  EXEC SQL UPDATE GRADES
2.          SET GRADE = :grade
3.          WHERE  CURRENT OF student_grades;
```

- **הפקודה DELETE** – בדומה לפקודת העדכון, גם בפקודה זו יש להוסיף את פרמטר CURRENT OF בתנאי המופיע במשפט WHERE. פקודה זו תבטל את השורה הנוכחית שנשלפה על ידי הפקודה FETCH האחרונה.

```
1.  EXEC SQL DELETE FROM GRADES
2.          WHERE  CURRENT OF student_grades;
```

התוכנית המוצגת בתרשים 12.24 הינה קטע מתוך תוכנית מקיפה יותר, המאפשרת למשתמש לעדכן ציון של סטודנט תוך כדי סריקת השורות של הטבלה סטודנטים.

הסבר:

- בשורות 5 ו-6 מבקשים מהמשתמש להזין את מספר הסטודנט.
- בשורות 9 עד 13 מוגדרת השאילתה להצגת השורות מתוך טבלת הציונים. נשים לב שבשורה 13 מופיעה העובדה שדרך Cursor זה ניתן יהיה לעדכן רק את הציון ולא כל נתון אחר שנשלף ומוצג.
- בשורה 14 פותחים את ה-Cursor. מכיון שהוא נפתח ללא הפרמטר FOR READ ONLY, משמעות הדבר היא שאנו פותחים Updatable Cursor.
- שורות 16 עד 28 מכילות את הלולאה ששולפת את השורות של השאילתה.
- בשורה 17 מבצעים את שליפת השורה ובשורה 19 מציגים אותה (לא כל פרטי פקודת ההדפסה מופיעים). לאחר הצגת נתוני הסטודנט, מוצגת הודעה על המסך שבה המשתמש מתבקש לקבוע אם ברצונו לעדכן את הציון או להמשיך לציון הבא.
- בשורה 21 קולטים את התשובה של המשתמש ומכניסים אותה למשתנה action_flag, שכמובן צריך להגדירו בתחילת התוכנית אבל, מאחר ולא הצגנו את כל יתר ההגדרות, גם הגדרה זו אינה מופיעה.
- אם המשתמש מבקש לעדכן את הציון, מופיעה הודעה המבקשת ממנו להזין את הציון החדש ובשורה 25 עד 27 מופיעה פקודת העדכון המתייחסת לעדכון הציון לשורה האחרונה שנשלפה מה-Cursor.

```
1.  main()
2.  {
3.     .
4.     .
5.     printf ("\n Enter Student Number: ");
6.     scanf ("%s", student_number);
7.     EXEC SQL WHENEVER SQLERROR GOTO sql_error;
8.     EXEC SQL WHENEVER NOT FOUND GOTO student_not_found;
9.     EXEC SQL DECLARE student_grades CURSOR FOR
10.        SELECT STUDENT_ID, COURSE_ID, SEMESTER, TERM, GRADE
11.        FROM     GRADES
12.        WHERE  STUDENT_ID = :student_number
13.        FOR UPDATE OF GRADE;
14.  EXEC SQL OPEN student_grades ;
15.  EXEC SQL WHENEVER NOT FOUND GOTO end_of_query;
16.    for (; ; ) {
17.      EXEC SQL FETCH student_grades INTO :student_number,
18.                                  :student_name, :course_number,  :term, :grade;
19.      printf ("Student Data::  %s\n", student_name, .........);
20.      printf ("Update Row (U) or Next Row (N)? :");
21.      scanf ("%s", action_flag);
22.      if (action_flag = 'U') {
23.         printf ("Enter new grade: ");
24.         scanf ("%s", grade);
25.         EXEC SQL UPDATE GRADES
26.         SET GRADE = :grade
27.         WHERE CURRENT OF student_grades ; }
28.     }
29.    exit()
30.  sql_error:
31.     printf ("SQL error: %ld\n", sqlca.sqlcode);
32.     exit();
33.  student_not_found:
34.     printf ("Student not found. \n");
35.     exit();
36.  end_of_query:
37.     EXEC SQL CLOSE student_grades;
38.     EXEC SQL DISCONNECT;
39.     exit ()
40.   }
```

תרשים 12.24: דוגמה לקטע תוכנית הכולל את ההגדרות.

פקודות SQL דינמיות (Dynamic SQL)

אחת המגבלות העיקריות של השיטה שהוצגה עד כה היא שפקודות SQL המופיעות בתוך שפה מארחת, בין באופן ישיר בין כחלק מהגדרת Cursor, יכולות לפנות למשתנים של השפה המארחת רק למטרות מסוימות: העברת פרמטרים המופיעים בעיקר בפקודות SELECT, UPDATE, DELETE ו-INSERT. הדוגמה הבאה מציגה פקודה לעדכון טבלת הציונים, פקודה הפונה למשתנה של השפה המארחת כדי לקבוע איזה שורה תעודכן:

```
1.  EXEC SQL UPDATE GRADES
2.              SET GRADE = :grade
3.              WHERE STUDENT_ID = :student_number AND
4.                      COURSE_ID = :course_id;
```

המשתנים student_number ו- course_id הם של השפה המארחת ומופיעים במשפט WHERE בדוגמה זו.

נניח שהיינו רוצים לכתוב תוכנית שתאפשר למשתמש לקבוע בזמן ריצה אם ברצונו לעדכן את הציון או את ציון הסמסטר. עם זאת, צריך לעשות זאת מבלי לכתוב שתי פקודות עדכון שונות, אחת לכל מצב. נניח שהיינו יכולים לכתוב את התוכנית הזו:

```
1.  printf ("\n Enter Student Number: ");
2.  scanf ("%s", student_number);
3.  EXEC SQL WHENEVER SQLERROR NOTFOUND GOTO student_not_found;
4.  EXEC SQL SELECT STUDENT_ID, COURSE_ID, SEMESTER,
5.                  TERM, GRADE, GRADE_SEM
6.          FROM GRADES
7.          WHERE STUDENT_ID = :student_number
8.          INTO :student_number, :course_number, :semester,
9.                  :term, :final_grade, semester_grade;
10.   printf ("\n Student data .........);
11.   printf ("\n Enter F to update final grade or enter S to update the semester grade: ");
12. scanf ("%s", action_flag);
13. if (action_flag = 'F')
14.     grade_to_update = 'GRADE' ;
15.     else
16.     grade_to_update = 'GRADE_SEM';
17. printf ("\n Enter new grade: ");
18. scanf ("%s", grade);
19. EXEC SQL WHENEVER SQLERROR goto sql_error;
20. EXEC SQL UPDATE GRADES
21.         SET :grade_to_update = :grade
22.         WHERE STUDENT_ID = :student_number AND
23.                 COURSE_ID = :course_id;
```

תרשים 12.25: דוגמה לקטע תוכנית לעדכון ציונים.

הסבר:

- שורות 1 ו-2 מאפשרות למשתמש להזין את מספר הסטודנט.
- שורות 4 עד 9 שולפות את נתוני הסטודנט מתוך טבלת ציונים.
- שורה 10 מציגה את נתוני הסטודנט. הפקודה אינה מציגה את כל הפירוט משיקולי נוחות וקריאות.
- שורות 10 ו-11 מאפשרות למשתמש להחליט אם הוא רוצה לעדכן את הציון הסופי או את ציון הסמסטר.
- שורות 13 עד 16 מכניסות למשתנה חדש, grade_to_update את שם העמודה שאותה יש לעדכן, בהתאם לבקשת המשתמש.
- שורות 17 ו-18 קולטות את הציון החדש שצריך להחליף את הציון הקודם.
- שורות 20 עד 23 מבצעות את פקודת העדכון. נשים לב שבשורה 21, שם העמודה שאותה יש לעדכן מופיע כמשתנה.

כמובן שתוכנית זו אינה חוקית בשיטת העבודה שהצגנו עד כה – **השיטה הסטטית** לשיבוץ פקודות SQL בשפה מארחת. בשיטה זו לא ניתן לקבוע שמות של עמודות או טבלאות כמשתנים המקבלים את תוכנם תוך כדי ריצת התוכנית. בשיטה הסטטית, כל מרכיבי פקודת SQL ידועים מראש וחייבים להכתב על ידי מפתח היישום בשלב כתיבת היישום כדי לבצע הידור לכל פקודות SQL ולהכין עבורן תוכנית גישה לבסיס הנתונים. מאחר ובדוגמה שלנו שם העמודה לא ידוע מראש לא ניתן לבצע הידור מראש של הפקודה UPDATE. בדוגמה הפשוטה שלנו כמובן שניתן היה לכתוב את התוכנית עם שתי פקודות UPDATE, אחת לכל מקרה, אולם מה ניתן לעשות במצבים מורכבים יותר?

כאן נכנסת לתמונה שיטת עבודה שונה לשיבוץ פקודות SQL בשפה מארחת – **השיטה הדינמית** (Dynamic Embedded SQL). שיטה זו מתאימה במיוחד למצבים שבהם לא ניתן להגדיר מראש מהן הטבלאות, העמודות ואפילו תנאי השליפה. הדוגמה הברורה ביותר היא אם נרצה לבנות מחולל שאילתות שיאפשר למשתמש לא מנוסה לבנות שאילתות מזדמנות. מחולל השאילתות ישאל את המשתמש לאיזה טבלה הוא רוצה לגשת, יציג לו את כל העמודות המופיעות בטבלה וישאל אותו איזה עמודות הוא מבקש לראות. המשתמש יצביע על העמודות המבוקשות ויגדיר את תנאי השליפה ויפעיל את השאילתה. במצב זה, אין כל אפשרות לבנות מראש את פקודות SQL בתוך מחולל השאילתות, מאחר והן לא ידועות. עבור מצבים מיוחדים אלה פותחה השיטה הדינמית.

לפני שנתאר את השיטה הדינמית, נדגיש שקיים שוני בצורת היישום של השיטה הדינמית במערכות RDBMS מסחריות שונות. להבדיל מהשיטה הסטטית, הדומה כמעט בכל המערכות המסחריות, השיטה הדינמית חשופה יותר למבנה הפנימי של מערכת RDBMS ומכאן השוני בצורת היישום.

בנייה דינמית של פקודות SQL

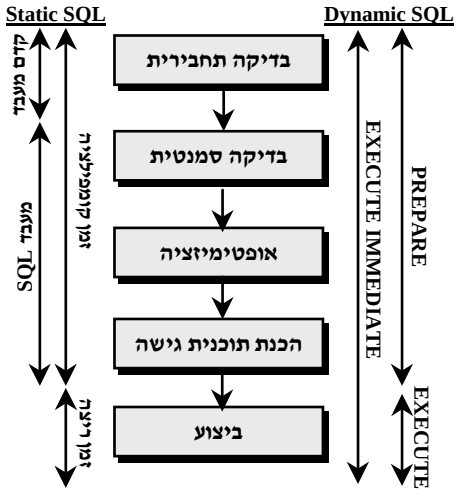
בצורה פשטנית במקצת, השיטה הדינמית מאפשרת לבנות פקודות SQL בתוך משתנה של השפה המארחת. משתנה זה יקבל את התוכן שלו תוך כדי ריצת התוכנית. רק לאחר שכל פקודת SQL נבנתה בתוך המשתנה, ניתן למסור את תוכנו למעבד SQL לביצוע הידור. משמעות הדבר היא שההידור של פקודות SQL בשיטה הדינמית מתבצעת במהלך ריצת התוכנית ולא מראש, כמו בשיטה הסטטית. כמובן ששיטה זו צורכת משאבי מחשוב רבים יותר בגלל הצורך לבצע הידור ואופטימיזציה של פקודות SQL תוך כדי ריצת התוכנית ולכן זמני הביצוע של תוכניות העובדות בשיטה הדינמית יהיו גרועים יותר מהשיטה הסטטית. מצד שני קיבלנו כאן גמישות עצומה, המאפשרת לנו לבנות פקודות SQL מורכבות תוך כדי ריצה.

תרשים 12.26 מציג את ההבדל המהותי בין השיטה הסטטית לבין השיטה הדינמית.

- **השיטה הסטטית:** כפי שניתן לראות בצד שמאל, בשיטה זו שלבי הבדיקה התחבירית, הבדיקה הסמנטית, האופטימיזציה והכנת תוכנית הגישה מבוצעים בזמן ההידור על ידי שני רכיבים שונים: (א) קדם-המהדר השולף את פקודות SQL מתוך תוכנית היישום, מחליף אותן בפקודות CALL ומבצע את הבדיקה התחבירית הבסיסית; ו-(ב) מעבד SQL שמבצע את כל שאר השלבים עד שתוכנית הגישה מוכנה. בזמן הריצה מופעלת תוכנית הגישה ומוסרת פרמטרים ממשתני השפה המארחת.
- **השיטה הדינמית:** פקודת SQL אינה ידועה בשלב ההידור של תוכנית היישום, ולכן כל השלבים מתבצעים בזמן ריצה. קיימות שתי שיטות לביצוע:
  - **ביצוע מיידי (EXECUTE IMMEDIATE)** בשיטה זו הפקודה עוברת את כל השלבים עד להכנת תוכנית הגישה לבסיס הנתונים ומתבצעת מיידית.
  - **ביצוע דו-שלבי (PREPARE and EXECUTE):** בשיטה זו קיימת הפרדה בין שלב הכנת תוכנית הגישה לבסיס הנתונים לבין שלב ביצוע תוכנית הגישה. בשלב ההכנה הראשון מתבצעת הכנת תוכנית הגישה על ידי הפעלת פקודה מיוחדת, PREPARE, המפעילה את מעבד SQL ומבצעת את כל השלבים עד להכנת תוכנית הגישה. בשלב השני מבוצעת הפקודה EXECUTE המפעילה את תוכנית הגישה לבסיס הנתונים. נחזור ונדגיש שגם שלב PREPARE מתבצע בזמן ריצה.

  ההבדל הוא שאם פקודת SQL מתבצעת פעם אחר פעם מכיון שהיא משתתפת בלולאה, למשל, יהיה זה בזבזני מאוד לבצע כל פעם מחדש את כל השלבים כפי שהפקודה EXECUTE IMMEDIATE עושה. במקרה זה ניתן לבצע את ההכנה מחוץ ללולאה, ובתוך הלולאה – להפעיל את הפקודה EXECUTE בלבד. בנוסף לכך, אם יש צורך להעביר פרמטרים בזמן ריצה מהשפה המארחת אל פקודות SQL, ניתן לעשות זאת רק על ידי שימוש בשיטה דו-שלבית.

למרות שהשיטה הדינמית מורכבת יותר לתכנות, היא הפכה לנפוצה מאוד בעיקר בסביבות שרת/לקוח, המפעילות כלי שאילתות מתוחכמים לבסיסי נתונים תפעוליים. משתמשים בה גם לכתיבת שאילתות מורכבות לבסיסי הנתונים של מחסני הנתונים.



תרשים 12.26: הבדלים בין השיטה הסטטית והדינמית.

ביצוע מיידי של פקודת SQL

לביצוע מיידי של פקודת SQL מופעלת הפקודה EXECUTE IMMEDIATE. פקודה זו מאפשרת שיגור של פקודת SQL, המשובצת בתוך משתנה של השפה המארחת, אל מעבד SQL, כדי להפעילה מיידית בגמר הכנת תוכנית הגישה לבסיס הנתונים.

תחביר הפקודה:

תרשים 12.27: תרשים תחביר של הפקודה EXECUTE IMMEDIATE.

פקודת SQL שבתוך המשתנה אינה יכולה להכיל הפניות למשתנים של השפה המארחת, כלומר המשתנה חייב להכיל פקודת SQL מושלמת ללא העברת פרמטרים. אם יש צורך בהעברת פרמטרים, יש להשתמש בשיטה של הכנה וביצוע, שתוסבר מייד בהמשך. לא ניתן לבצע באמצעות הפקודה EXECUTE IMMEDIATE והפקודה SELECT המחזירות מספר שורות. לשם כך, יש צורך להפעיל מנגנון דומה למנגנון ה-Cursor, תוך שימוש בפקודות מיוחדות שעברו התאמות מסוימות לעבודה בסביבה דינמית.

דוגמה : בתרשים 12.28 נציג את תוכנית היישום הקודמת, אך הפעם ניישם אותה בשיטה הדינמית.

הסבר:

- שורה 2 מגדירה את המשתנה sql_statement שישמש לבנייה דינמית של פקודת SQL.
- שורה 3 מגדירה את המשתנה sql_condition שיכיל את התנאי שיופיע בפקודת העדכון.
- שורה 7 מכינה את תחילת פקודת SQL תוך שימוש בפקודה של שפת C המאפשרת העתקה של מחרוזת כלשהי אל תוך משתנה.
- שורות 26 ו-27 מאפשרות למשתמש להזין את התנאי לעדכון. על המשתמש להזין מחרוזת במבנה כגון: WHERE STUDENT_ID = '105' AND COURSE_ID = 'C-55'. כמובן שאם הוא רוצה לעדכן שורה אחרת הוא יזין מספר סטודנט ומספר קורס אחר. הסיבה שיש צורך להזין את המספרים האלה כקבועים ולא כמשתנים של השפה המארחת נובעת מהעובדה שהפקודה EXECUTE IMMEDIATE אינה מרשה שפקודת SQL המועברת למעבד SQL תכיל משתנים. היא חייבת להיות פקודת SQL מושלמת.
- שורה 29 מכניסה אל תוך המשתנה sql_statement את התוכן של המשתנה grade_to_update. משתנה זה מכיל את המחרוזת GRADE או GRADE_SEM בהתאם לציון שהמשתמש ביקש לעדכן. אנו משתמשים כאן בפקודת C המאפשרת לשרשר שתי מחרוזות תווים.
- שורה 30 משלימה את יתרת פקודת SQL. נשים לב שההמשך נכתב בדיוק כמו שהיינו כותבים בפקודת SQL רגילה.
- שורה 31 מבצעת את פקודת EXECUTE IMMEDIATE. פקודה זו תפעיל את מעבד SQL שיבדוק את תקינות הפקודה, יבצע אופטימיזציה, יכין את תוכנית הגישה ויפעיל אותה. אם תוך כדי ביצוע תהליך ההידור ימצא המעבד שפקודת SQL אינה תקינה, הוא ירשום בתוך שטח ה-SQLCA את קודי ההחזר המתאימים.

```
1.  EXEC SQL BEGIN DECLARE SECTION;
2.      char  sql_statement[100];
3.      char sql_condition[200];
4.      short grade;
5.  EXEC SQL END DECLARE SECTION;
6.  char action_flag[1];
7.  strcpy (sql_statement, “UPDATE GRADES SET “);
8.  printf (“\n Enter Student Number: “);
9.  scanf (“%s”, student_number);
10. EXEC SQL WHENEVER SQLERROR NOTFOUND GOTO student_not_found;
11. EXEC SQL SELECT STUDENT_ID, COURSE_ID,
12.                     SEMESTER, TERM, GRADE, GRADE_SEM
13.             INTO :student_number, :course_number,
14.                 :semester, :term, :final_grade, semester_grade
15.             FROM GRADES
16.             WHERE STUDENT_ID = :student_number;
17. printf (“\n Student data .........);
18. printf (“\n Enter F to update final grade or enter S to update the semester grade: ”);
19. scanf (“%s”, action_flag);
20. if (action_flag = ‘F’)
21.     grade_to_update = ‘GRADE’ ;
22.     else
23.     grade_to_update = ‘GRADE_SEM’;
24. printf (“\n Enter new grade: “);
25. scanf (“%s”, grade);
26. printf (“\n Enter WHERE condition”);
27. scanf (“%s”, sql_condition);
28. EXEC SQL WHENEVER SQLERROR goto sql_error;
29. strcat (sql_statement, grade_to_update);
30. strcat (sql_statement , sql_condition);
31. EXEC SQL EXECUTE IMMEDIATE :sql_statement;
32. exit();
```

תרשים 12.28: דוגמה לתוכנית לפי השיטה הדינמית.

ביצוע דו-שלבי של פקודת SQL

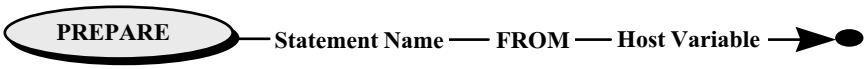
כל פעם שהפקודה EXECUTE IMMEDIATE מתבצעת, מערכת RDBMS מבצעת את כל השלבים עד להכנת תוכנית הגישה ומפעילה אותה מיידית. שיטה זו טובה אם יש לבצע כל פקודת SQL פעם אחת בלבד. אם פקודת SQL מתבצעת פעמים רבות, למשל בתוך לולאה, יש בזבוז רב בשיטה זו, מאחר ותוכנית הגישה לבסיס הנתונים נבנית כל פעם מחדש. חסרון נוסף של שיטת הביצוע המיידי נובע מהעובדה שלא ניתן להשתמש בתוך פקודת SQL במשתנים של השפה לצורך העברת פרמטרים. פקודת SQL הנבנית בתוך משתנה, חייבת להיות פקודה שלמה ולא יכולה להכיל הפניות למשתנים נוספים של השפה המארחת.

כדי לפתור הן את בעיית בזבוז המשאבים בגלל הצורך לבנות כל פעם מחדש את תוכנית הגישה והן את בעיית העברת הפרמטרים אל פקודת SQL, פותחה השיטה הדו-שלבית לבנייה דינמית של פקודות SQL הקרויה Two Step Dynamic SQL.

בשיטה זו יש אבחנה בין שני שלבים:

- **שלב הכנת תוכנית הגישה (Prepare):** בשלב זה בונים את פקודת SQL בצורה רגילה בתוך משתנה של השפה המארחת, כמו בשיטה הקודמת. לאחר בניית פקודת SQL בתוך המשתנה, מופעל מעבד SQL תוך שימוש בפקודה PREPARE. פקודה זו בונה את תוכנית הגישה לבסיס הנתונים אולם אינה מבצעת אותה. קיים שוני בין מערכות RDBMS מסחריות וצורת הטיפול שלהן בהעברת פרמטרים. לדוגמה, כאשר צריך ב-DB2 להעביר פרמטרים לפקודת SQL, ניתן לבצע זאת תוך שימוש בתו סימן השאלה (?) כדי לציין שזהו פרמטר. לעומת זאת, ב-Oracle מותר לשים שם של משתנה בתוך פקודת SQL הדינמית ואין צורך להשתמש בסימני שאלה.
- **שלב הביצוע (Execute):** בשלב זה מבצעים על ידי הפקודה EXECUTE את תוכנית הגישה שהוכנה בשלב הקודם על ידי. פקודת הביצוע יכולה להעביר פרמטרים אל פקודת SQL. ניתן לבצע שלב זה פעם אחר פעם, למשל בתוך לולאה, ולמסור בכל פעם פרמטרים שונים וללא צורך לעבור כל פעם מחדש דרך כל שלבי הכנת תוכנית הגישה.

תחביר הפקודה PREPARE:



תרשים 12.29: תרשים תחביר לפקודה PREPARE

לכל פקודה שמכינים באמצעות PREPARE יש לתת שם Statement Name. שם זה ישמש לזיהוי הפקודה שיש לבצע בשלב השני. פקודת SQL שעבורה יש להכין תוכנית גישה צריכה להמצא בתוך משתנה של השפה המארחת. אם במהלך הכנת תוכנית הגישה ימצא מעבד SQL שגיאות, הוא ידווח עליהם בתוך המשתנים SQLCODE ו- SQLSTATE בהתאם. כאמור קיים שוני בין צורת הטיפול של מערכות מסחריות בהעברת פרמטרים. ב-DB2 העברת פרמטרים משפת התכנות המארחת אל פקודת SQL מתבצעת על ידי רישום תו מיוחד, סימן השאלה. נציג לדוגמה פקודת עדכון המכילה שני פרמטרים.

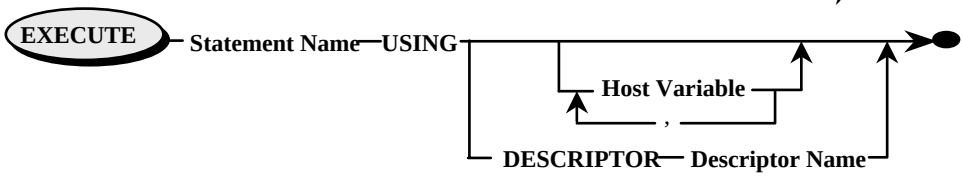
```
1.  UPDATE STUDENTS
2.      SET CITY = ?
3.      WHERE STUDENT_ID = ?
```

הפרמטר הראשון יכיל את שם עיר המגורים החדשה של הסטודנט והפרמטר השני יכיל את מספר הסטודנט עבורו יש לבצע את העדכון. פרמטרים אלה יועברו אל הפקודה בזמן הביצוע. לעומת זאת, במערכת Oracle נוכל לרשום את הפקודה הנ"ל בצורה הבאה:

```
1.  UPDATE STUDENTS
2.      SET CITY = :city_name
3.      WHERE STUDENT_ID = :student_number
```

כפי שניתן לראות, ב-Oracle רושמים את שמות המשתנים כאילו היו פרמטרים לכל דבר. בכל מקרה, אין להעביר כפרמטר שמות של טבלאות ושל עמודות. ניתן להכין בשיטה זו פקודות עדכון או פקודות SELECT המחזירות שורה אחת בלבד. הפקודה SELECT היכולה להחזיר מספר שורות מחייבת עבודה עם מנגנון ה-Cursor ולכן יש להכינה בדרך מיוחדת.

תחביר הפקודה EXECUTE:



תרשים 12.30: תרשים תחביר לפקודה EXECUTE

פקודה זו מבצעת את תוכנית הגישה שהוכנה בשלב הקודם. קיימות שתי שיטות שונות להעברת פרמטרים אל פקודת SQL:

- **שימוש במשתנים בתוך פקודת SQL:** בשיטה זו רושמים זה אחר זה את שמות המשתנים המופיעים בתוך פקודת SQL.
- **שימוש בשטח עבודה מיועד:** בשיטה זו אין רושמים את הפרמטרים בתוך EXECUTE עצמה, אלא משתמשים בהפנייה לשטח עבודה מיוחד בשם SQLDA (SQL Data Area), שבו מופיעים הפרמטרים שיש להעביר. שיטה זו מתאימה למצב שבו לא ידוע מראש מספר הפרמטרים שיש להעביר ומה טיפוס הנתונים שלהם.

מספר הפרמטרים המופיעים בפקודה EXECUTE או המופיעים בשטח העבודה, חייב להיות זהה למספר סימני השאלה שבפקודת SQL במערכת DB2, או למספר המשתנים המופיעים בתוך פקודת SQL במערכת Oracle. הפרמטרים יועברו לפי הסדר, כלומר המשתנה הראשון יחליף את סימן השאלה הראשון, המשתנה השני יחליף את סימן השאלה השני וכך הלאה. בשתי השיטות, ניתן להעביר ערכים חסרים אל הפרמטרים תוך שימוש במשתנה Indicator Variable, המציין אם המשתנה מכיל ערך כלשהו או שהוא מכיל Null.

אם משתמשים בשיטת העברת הפרמטרים באמצעות שטח העבודה המיוחד, יש להשתמש בפקודת SQL זו: EXEC SQL INCLUDE sqlda; המכניסה אל תוכנית היישום את כל ההגדרות שטח העבודה. כל מערכת RDBMS מסחרית יוצרת שטח זה במבנה שונה. נציג בהמשך את המבנה של שטח SQLDA במערכת DB2.

הפקודה INCLUDE תשלב בתוכנית היישום את ההגדרות האלו:

```
1.  struct sqlda {
2.      char   sqldaid[8];
3.      long   sqldabc;
4.      short sqln;
5.      short sqld;
6.      struct {
7.          short sqltype;
8.          short sqllen;
9.          char *sqldata;
10.         short *sqlind;
11.         struct {
12.             short sqlname1;
13.             char  sqlresult[30];
14.         } sqlname;
15.     } sqlvar[1];
16. }
```

תרשים 12.31: תוצאה של הפקודה INCLUDE.

כפי שניתן לראות, שטח SQLDA מורכב משני סוגי חלקים:

חלק בעל אורך קבוע (Fixed Part): חלק זה נמצא בתחילת שטח SQLDA ומגדיר את גודלו.

- שורה 2 מגדירה משתנה המכיל את שם השטח.
- שורה 3 מגדירה את המשתנה sqldabc המציין את מספר הבתים שתופס כל שטח SQLDA.
- שורה 4 מגדירה את המשתנה sqln המכיל מונה של מספר החלקים בעלי האורך המשתנה שהוקצו.
- שורה 5 מגדירה את המשתנה sqld המכיל את המונה של מספר העמודות הקשורות לפקודה זו.

חלק בעל אורך משתנה (Variable Part): חלק זה מכיל את התוכן של כל אחד מהפרמטרים המועברים או המוחזרים כתוצאה מביצוע הפקודה.

- שורה 7 מגדירה את המשתנה sqltype המכיל קוד המציין את טיפוס הנתונים של הפרמטר המועבר. לדוגמה, קוד 20 יכול לציין שהמשתנה הוא מסוג CHAR, קוד 21 מציין שהמשתנה הוא מסוג VARCHAR, קוד 30 מציין שהמשתנה הוא מסוג INTEGER וכד'. קודים אלה שונים ממערכת RDBMS אחת לשנייה.
- שורה 8 מגדירה את המשתנה sqllen המציין את האורך בבתים של הפרמטר המועבר.
- שורה 9 מגדירה את המשתנה sqldata* מסוג מצביע. משתנה זה מכיל הצבעה לשטח בתוכנית המכיל את הערך של הפרמטר.

- שורה 10 מגדירה את המשתנה sqlind* שהוא מסוג מצביע. משתנה זה מכיל הצבעה לשטח בתוכנית המכיל את הערך של משתנה מסוג Indicator Variable, המשמש לטיפול בערכים חסרים. אם משתנה זה מכיל ערך שלילי, יועבר ערך Null אחרת יועבר הערך האמיתי של המשתנה.

בגלל רמת הסיבוכיות הגבוהה של עבודה עם שטח SQLDA ועם פקודות SQL דינמיות ובגלל השונות הרבה בין מערכות RDBMS מסחריות שונות, נגביל את הדיון רק לדוגמאות של עבודה עם העברת פרמטרים בתוך הפקודה EXECUTE.

דוגמה : יש לבנות תוכנית לעדכון ציוני הסטודנטים. לאחר עדכון הציון, על התוכנית לשאול את המשתמש אם ברצונו לעדכן ציון נוסף. אם התשובה היא חיובית, יש לאפשר עדכון של ציון נוסף. העברת הפרמטרים נעשית על ידי הרישום שלהם בפקודה EXECUTE.

```
1.  EXEC SQL BEGIN DECLARE SECTION;
2.        char       sql_statement[100];
3.        short      new_grade;
4.        varchar student_number(5);
5.  EXEC SQL END DECLARE SECTION;
6.  char action_flag[1];
7.  strcpy (sql_statement,
8.  "UPDATE GRADES SET GRADE = ? WHERE STUDENT_ID = ? ");
9.  EXEC SQL PREPARE update_student_grade FROM sql_statement;
10. do {
11.   printf ("\n Enter Student Number: ");
12.   scanf ("%s", student_number);
13.   printf ("\n Enter new grade: ");
14.   scanf ("%s", new_grade);
15.   EXEC SQL WHENEVER SQLERROR NOTFOUND
16.   GOTO student_not_found;
17.   EXEC SQL EXECUTE update_student_grade
18.   USING :new_grade, :student_number;
19.   printf ("\n Do you want to update another student (Y/N):");
20.   scanf ("%c", action_flag);
21. }
22. until (action_flag ! = 'Y');
23. exit();
24. student_not_found:
25. printf ("Student not found. \n");
26. exit();
```

תרשים 12.32: דוגמה לתוכנית עדכון הכוללת העברת פרמטרים.

הסבר:

- בשורות 7 ו-8 מכניסים לתוך המשתנה sql_statement את פקודת SQL לעדכון ציוני סטודנט. נשים לב שהציון ומספר הסטודנט מופיעים כפרמטרים ולכן במקומם מופיע סימן השאלה.
- שורה 9 מכינה את תוכנית הגישה על ידי ביצוע הפקודה PREPARE. תוכנית הגישה תקרא בשם update_student_grade.

- שורות 10 עד 21 מכילות את הלולאה שתתבצע כל עוד המשתמש יבקש לעדכן ציונים נוספים.
- שורות 11 ו-12 קולטות את מספר הסטודנט שעבורו יש לעדכן את הציון.
- שורות 13 ו-14 קולטות את הציון החדש שיחליף את הציון הקודם.
- שורות 17 ו-18 מבצעות את הפקודה לאחר ששני הפרמטרים מועברים אליה. נשים לב לסדר הופעת הפרמטרים.
- שורות 19 ו-20 מבקשות מהמשתמש להזין Y אם הוא מבקש לעדכן ציון נוסף או N אם הוא רוצה להפסיק את עדכון הציונים.

פקודות SELECT המחזירות מספר בלתי ידוע של שורות

פקודות SELECT המחזירות מספר בלתי ידוע מראש של שורות קרויות **פקודות SELECT דינמיות** או **שאילתות דינמיות** (Dynamic Queries). בניית פקודות מסוג זה, המחזירות מספר בלתי ידוע מראש של שורות, הוא תהליך מורכב יחסית. המורכבות נובעת כמובן מהשילוב בין העובדה שלא ניתן לדעת מראש מהן העמודות שתוחזרנה כתוצאה מהשאילתה ומהו טיפוס הנתונים של כל עמודה כזאת. עם לא ניתן לדעת מראש כמה שורות תכיל טבלת התוצאה. כתיבת תוכניות יישום העוסקות בהקצאה דינמית של שטחי עבודה הינה פעולה מורכבת, בגלל הצורך לעבוד עם משתנים מטיפוס מצביע. אנו לא נציג קטעים אלה. ביצוע של שאילתות מרובות שורות דינמיות מחייב שימוש במנגנון Cursor כדי לאפשר עיבוד סדרתי של השורות המוחזרות על ידי השאילתה.

נציג כאן רק את העקרונות והשלבים של עבודה עם שאילתות דינמיות:

- **הגדרת Cursor:** יש להגדיר Cursor באמצעות פקודה DECLARE CURSOR דינמית, המצביעה על המשתנה שבתוכו נמצאת השאילתה. בגירסה הסטטית יש לכתוב את השאילתה המגדירה Cursor בתוך הפקודה DECLARE CURSOR, בעוד שבגירסה הדינמית הפקודה אינה מכילה את השאילתה, אלא את שם המשתנה שבתוכה השאילתה תִבָּנֶה במהלך ריצת התוכנית. בדוגמה הבאה מגדירים Cursor בשם query1_statement שיוגדר במהלך ריצת התוכנית בתוך המשתנה query1_buffer.

```
1. EXEC SQL DECLARE query1_cursor CURSOR FOR query1_buffer
```

- **בניית השאילתה:** תוכנית היישום בונה את השאילתה בתוך המשתנה המופיע בפקודת DECLARE CURSOR. בדוגמה שלנו, התוכנית תבנה את הפקודה SELECT המתאימה בתוך משתנה ששמו query1_buffer. פקודה זו יכולה להכיל פרמטרים. נניח שבמהלך ריצת התוכנית המשתנה query1_buffer מקבל את הערך הזה:

```
1. strcpy (query1_buffer, "SELECT STUDENT_ID, NAME
2.                 FROM STUDENTS WHERE CITY = 'Haifa'");
```

נזכיר ששאילתה זו נבנתה באופן דינמי ולא על ידי כתיבתה מראש. בזמן ריצת התוכנית המשתמש קובע עם איזה טבלה הוא רוצה לפעול – טבלת סטודנטים במקרה שלנו. עכשיו הוא קובע איזה עמודות הוא רוצה לראות – העמודות מספר הסטודנט ועיר המגורים. לבסוף הוא קובע את תנאי השליפה – כל הסטודנטים הגרים בחיפה.

- **הכנת תוכנית הגישה:** לאחר שהפקודה SELECT הוכנה בתוך המשתנה המתאים, יש לבצע את הכנת תוכנית הגישה. הכנה זו תבוצע באמצעות הפקודה PREPARE.

1. EXEC SQL PREPARE query_statement FROM query1_buffer

- **קבלת מבנה טבלת התוצאה:** בשלב כתיבת התוכנית לא היה ידוע כמה עמודות תכיל הטבלה התוצאתית ומה טיפוס הנתונים של כל עמודה, ולכן צריך לבצע פקודה מיוחדת וחדשה המיועדת רק עבור שאילתות מרובות שורות דינמיות: הפקודה DESCRIBE. פקודה זו גורמת למערכת RDBMS להעביר אל תוך השטח SQLDA את מספר העמודות שהשאילתה החזירה, את האורך של כל עמודה ואת טיפוס הנתונים של כל עמודה.

1. EXEC SQL DESCRIBE query_statement INTO query1_sqlda

השימוש בפקודה זו מתבצע לאחר שתוכנית הגישה כבר הוכנה אבל לפני שהיא מבוצעת. בדוגמה שלנו, מאחר והשאילתה שולפת שתי עמודות, יכילו המשתנים של שטח SQLDA את התיאור של טיפוס הנתונים והאורך של כל אחת מהעמודות.

1. **sqld = 2**
2. **sqlvar[0].sqltype = 20**
3. **sqlvar[0].sqllen = 5**
4. **sqlvar[0].sqlname.sqlname1 = 10**
5. **sqlvar[0].sqlname.sqlnamec = STUDENT_ID**
6. **sqlvar[1].sqltype = 20**
7. **sqlvar[1].sqllen = 20**
8. **sqlvar[1].sqlname.sqlname1 = 4**
9. **sqlvar[1].sqlname.sqlnamec = NAME**

כפי שניתן לראות, המשתנה sqld המופיע בשורה 1, מכיל את הערך 2, כלומר השאילתה מחזירה שתי עמודות. המערך מכיל כניסה אחת עבור כל אחת מהעמודות. המשתנה sqltype שבשורה 2 מציין את טיפוס הנתונים של העמודה, בדוגמה זו הקוד 20 מציין שטיפוס הנתונים הוא CHAR. המשתנה sqllen מכיל את אורך העמודה. המשתנה sqlname.sqlname1 שבשורה 4 מכיל את האורך של שם העמודה, והמשתנה sqlname.sqlnamec המופיע בשורה 5 מכיל את שם העמודה. בדוגמה זו, העמודה הראשונה קרויה STUDENT_ID ואורך שם העמודה הוא 10, והעמודה השנייה קרויה NAME ואורך השם שלה הוא 4. באמצעות מידע זה יכולה תוכנית היישום להחליט כיצד להציג את הנתונים שהתקבלו כתוצאה מהפעלת השאילתה.

- **הקצאת שטחי זיכרון עבור טבלת התוצאה:** בהתבסס על המידע שמערכת RDBMS מעבירה אל שטח העבודה SQLDA כתוצאה מביצוע DESCRIBE, תוכנית היישום מקצה באופן דינמי שטחי זיכרון עבור כל אחת מהעמודות של טבלת התוצאה. בשפת C מבוצעת הקצאה זו באמצעות הפקוד malloc. בעיה עקרונית של עבודה עם הפקודה DESCRIBE ועם SQL דינמי בכלל, היא היעדר הידע המוקדם על גודל שטח הזיכרון שיש להקצות עבור SQLDA.

תוכנית היישום יכולה להניח מספר מקסימלי כלשהו של עמודות ששאילתה תחזיר ולהקצות את השטח עבור מספר מקסימלי זה. התוכנית יכולה לבדוק לאחר כל פקודה אם השטח שהוקצה מספיק על ידי בדיקה של ערך המשתנה sqld. אם השאילתה החזירה מספר גדול יותר של עמודות מהמספר שהוקצה, יש להקצות מחדש את השטח המוגדל ולבצע את הפקודה DESCRIBE מחדש כדי ששטח SQLDA יקבל את הערכים הנכונים עבור כל העמודות. אין מגבלה על מספר הפעמים שמותר לבצע את הפקודה DESCRIBE.

- **פתיחת Cursor**: התוכנית תבצע עכשיו פתיחת Cursor. נזכיר שפקודה זו מבצעת את תוכנית הגישה שהוכנה לפני כן על ידי הפקודה PREPARE. פקודה זו מחליפה את EXECUTE הרגילה, שהיינו מבצעים לולא הצורך לטפל במספר בלתי ידוע של שורות. תוכנית הגישה תקבל בשלב זה את כל הפרמטרים הדרושים מתוך תוכנית היישום.

1. EXEC SQL OPEN query1_cursor

- **שליפת שורות מתוך Cursor**: בשלב זה מתחילה להתבצע הלולאה לשליפת שורות מתוך ה-Cursor באמצעות פקודות FETCH. להבדיל מהגירסה הסטטית של FETCH המגדירה את שמות המשתנים אליהם יש להעביר את העמודות, בגירסה הדינמית מתבצע שימוש ב-SQLDA כדי להנחות את מערכת RDBMS איזה משתנים יקבלו את התוצאות.

1. EXEC SQL FETCH query1_cursor USING DESCRIPTOR query1_sqlda

- **סגירת Cursor**: הפקודה CLOSE סוגרת את ה-Cursor.

1. EXEC SQL CLOSE query1_cursor

כפי שניתן לראות, תהליך בניית תוכניות יישום המשתמשות בשאילתות מרובות שורות דינמיות מורכב יותר, בגלל הצורך בעבודה עם SQLDA והצורך בהקצאה דינמית של שטחי זיכרון.

עבודה עם SQLDA על פי תקן SQL2

העבודה עם SQLDA היא מורכבת יחסית ומחייבת את מפתח היישום לעסוק בנושאים כגון הקצאה דינמית של שטחי זיכרון, שימוש במצביעים וכד'. כדי לפשט את העבודה, תקן SQL2 מציג גישה שונה במקצת, גישה המבוססת על עקרונות התכנות מוכוון אובייקטים (Object Oriented). בגישה זו, שטח SQLDA הינו אובייקט אשר המבנה שלו מוסתר ממפתח היישום והוא יכול לגשת אליו רק באמצעות הפונקציות שלו (Methods). מנקודת מבטו של מפתח היישום, שטח SQLDA מכיל מספר משתנים וביניהם COUNT המכיל את מונה מספר הפרמטרים או העמודות, LENGTH המכיל את אורך הפרמטר, DATA המכיל את התוכן של הפרמטר ו- INDICATOR המציין האם הנתון מכיל Null, NAME, שם הפרמטר. נציג כאן מספר פקודות המבוססות על תקן SQL2 לעבודה עם שטח SQLDA.

- הקצאת השטח SQLDA תתבצע על ידי פקודה הקובעת מהו המספר המקסימלי של המופעים הדרושים לאחסון העמודות של תוצאות השאילתה. לדוגמה כדי לפתוח שטח חדש בעל השם sqlda1 שיכיל חמישה פרמטרים, נכניס למשתנה number_of_parameters את הערך 5 ונכתוב:

```
1. EXEC SQL ALLOCATE DESCRIPTOR sqlda1
2.                 WITH MAX :number_of_parameters;
```

בגמר השימוש בשטח זה, ניתן לשחרר את הזיכרון על ידי הפקודה:

```
1. EXEC SQL DEALLOCATE DESCRIPTOR sqlda1 ;
```

- הגישה אל המשתנים המוגדרים בתוך SQLDA מתבצעת על ידי הפעלת הפונקציות המאפשרות את קריאת תוכן המשתנים או קביעתו. לדוגמה, כדי לקבל את תוכן המונה של מספר העמודות שהוקצה עבורם שטח זיכרון, יש לכתוב את הפקודה:

```
1. EXEC SQL GET DESCRIPTOR :sqlda1 COUNT = :count;
```

לדוגמה, כדי לקבוע את הערכים של הפרמטר הראשון בתוך SQLDA נכתוב את הפקודה:

```
1. EXEC SQL SET DESCRIPTOR :sqlda1 VALUE 1
2.                 LENGTH = :prm_length,  DATA = :prm_data ;
```

ממשק תכנות יישומים – API (Application Program Interface)

בשני הסעיפים הקודמים הצגנו שיטה המאפשרת שיבוץ של פקודות SQL בתוך שפה מארחת. מאחר ופקודות SQL אלו מופיעות באופן ישיר בתוך השפה המארחת והן אינן מוכרות על ידי המהדר של השפה המארחת, נוצר הצורך להשתמש בקדם-מהדר מיוחד. קדם-מהדר זה מתרגם, בין היתר, את פקודות SQL לפקודות CALL, פקודות הנתמכות על ידי כל שפת תכנות מודרנית.

לעומת השיטה המבוססת על קדם-מהדר, פותחה שיטה ולפיה ניתן לגשת אל מערכת RDBMS גם בצורה ישירה תוך שימוש בממשק התכנות (API– Application Program Interface) שהמערכת מעמידה לרשות מפתחי היישום. בשיטה זו, יצרן מערכת RDBMS מפרסם אוסף של ממשקי תכנות, או במילים אחרות – פונקציות, שבאמצעותן ניתן לבקש ממערכת RDBMS לבצע משימות שונות כמו לאחזר נתונים, לעדכן את תוכן בסיס הנתונים וכד'. שיטה זו הפכה לנפוצה במיוחד עם הופעת טכנולוגיית שרת/לקוח וקיבלה דחיפה עם הופעת בסיס הנתונים של חברת Sybase ובשלב מאוחר יותר עם התפוצה הרחבה של מערכת SQL Server של חברת Microsoft.

שתי מערכות אלו תומכות בממשק תכנות בלבד ואינן תומכות בעבודה עם קדם-מהדר. יחד איתן, גם מערכות מסחריות נוספות וביניהן מערכת Oracle, הוסיפו תמיכה בממשק תכנות, בנוסף לעבודה עם קדם-מהדר. תקן SQL2 אינו מטפל בממשק התכנות ולכן השוני בצורת העבודה במערכות מסחריות שונות יכול להיות גדול. כל מערכת מסחרית מספקת אוסף שונה של פונקציות, שמות שונים של הפונקציות ומספר פרמטרים שונה. בגלל השוני הגדול הזה בין ממשקי התכנות של מערכות מסחריות שונות, הוקם גוף הנקרא SQL Access Group, או בקיצור SAG, ששם לו למטרה לנסות ולהביא לגיבוש תקן בנושא זה. חברת Microsoft התבססה על טיוטא של קבוצה זו ויצאה עם תקן משלה, ממשק ODBC, המנסה להביא לסטנדרטיזציה מסוימת בכל נושא ממשקי התכנות.

העבודה עם ממשק תכנות דורשת לעיתים עבודה רבה יותר מאשר עם קדם-מהדר, כי במובן מסוים זהו ממשק ברמה נמוכה יותר בין שפת התכנות לבין מערכת RDBMS. עם זאת, ממשק התכנות מאפשר גמישות רבה יותר, אשר מזכירה את גמישות השיטה הדינמית לשיבוץ פקודות SQL, אבל ללא המורכבות הרבה שלה.

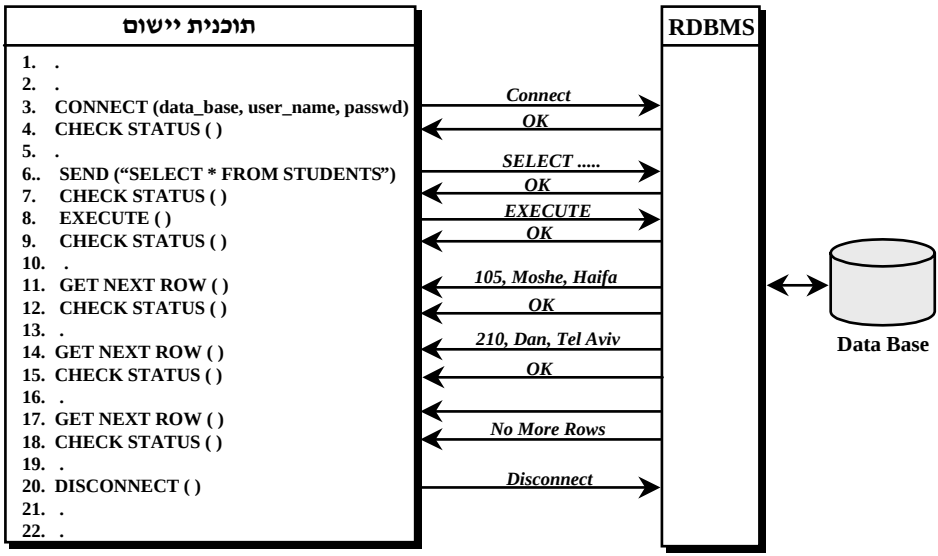
עקרון העבודה עם ממשק תכנות

כל מערכת מסחרית התומכת בממשק תכנות, מספקת אוסף של פקודות מיוחדות המאפשרות לתוכנית היישום לבקש ממערכת RDBMS לבצע מטלות שונות. לכל אחת מהפונקציות האלו יש לספק פרמטרים בהתאם. המערכת מבצעת את הפונקציה המבוקשת ומחזירה את התוצאה או את הסטטוס חזרה אל תוכנית היישום.

תרשים 12.33 מתאר את ההידודיות (אינטראקציה) בין תוכנית יישום לבין מערכת RDBMS. הפונקציות המופעלות כאן על ידי תוכנית היישום אינן של מערכת RDBMS מסוימת, אלא נועדו רק להסביר את הרעיון. נסביר בקצרה את תהליך העבודה:

- בשורה 3 תוכנית היישום מתקשרת אל בסיס הנתונים. היא מעבירה כפרמטר את שם בסיס הנתונים אליו היא מבקשת להתקשר, את שם המשתמש ואת הסיסמה.
- בשורה 4 תוכנית היישום בודקת את הסטטוס של ההתקשרות. אם ההתקשרות לא הצליחה מסיבה כלשהי, כמו למשל טעות בשם בסיס הנתונים, משתמש לא מוכר, משתמש לא מורשה לגשת לבסיס הנתונים וכד', תוכנית היישום תודיע על שגיאה.
- בשורה 6 תוכנית היישום שולחת בקשה להכין תוכנית גישה עבור פקודה SELECT מסוימת. מערכת RDBMS מכינה את תוכנית הגישה, אבל אינה מפעילה אותה.

❖ בשורה 7 תוכנית היישום בודקת את הצלחת הכנת תוכנית הגישה. אם מערכת RDBMS מצאה שגיאה כלשהי בפקודה, כגון טעות תחביר, משתמש לא מורשה לגשת לטבלה וכד', תוכנית היישום תדווח על השגיאה.



תרשים 12.33: תיאור ההידודיות (אינטראקציה) בין תוכנית היישום ומערכת RDBMS.

❖ בשורה 8 תוכנית היישום מבקשת את ביצוע תוכנית הגישה שהוכנה קודם.

❖ בשורה 9 תוכנית היישום בודקת אם בקשת הביצוע הצליחה. למשל אם מערכת RDBMS אינה מוצאת את תוכנית הגישה היא תדווח על הודעת שגיאה מתאימה.

❖ תוכנית היישום נכנסת ללולאה של שליפת שורות. בשורות 11, 14 ו-17 תוכנית היישום שולפת שורה אחר שורה מתוך הטבלה התוצאתית. התוכן של כל שורה מועבר ממערכת RDBMS אל תוכנית היישום לצורך עיבוד.

❖ בשורות 12, 15 ו-18 תוכנית היישום בודקת את סטטוס ביצוע הפקודה. בשורות 12 ו-15 קוד ההחזר הוא תקין. לעומת זאת, בשורה 18 קוד ההחזר מציין שלא נמצאו יותר שורות. זהו הסימן שיש להפסיק את הלולאה.

❖ בשורה 20 תוכנית היישום מבקשת לנתק את הקשר בינה ובין מערכת RDBMS.

ממשק התכנות של מערכת SQL Server

מערכת SQL Server של חברת Sybase היתה אחת המערכות הראשונות שהתבססה על ממשק תכנות ולא על קדם-מהדר. לטענת החברה, שיטה זו התאימה במיוחד ליעוד העיקרי של המערכת: בסיס נתונים לסביבת שרת/לקוח. מיקרוסופט רכשה את הקוד של המערכת ופיתחה עליו את המערכת המסחרית שלה, SQL Server, כיום בגירסה 7. במקביל המשיכה Sybase לפתח מוצר שלה בשם Sybase ASE, אשר נשתמש בו כדי להדגים ממשק תכנות. חשוב להדגיש שגם מוצרים נוספים תומכים בממשק תכנות וביניהם בסיס הנתונים של Oracle, שהממשק שלו קרוי OCI (Oracle Call Interface). להבדיל מממשק התכנות של SQL Server, ממשק התכנות של Oracle מזכיר במידה רבה את העבודה עם קדם-מהדר וקיימת הקבלה רבה בין פקודות משובצות לבין ביצוען דרך ממשק התכנות.

נסקור תחילה מספר פונקציות בסיסיות שניתן להפעילן על ידי ממשק תכנות:

- dblogin() מתקשרת לבסיס הנתונים על ידי קבלת שטח זיכרון מיוחד למטרה זו.
- dbopen() מתקשרת לבסיס הנתונים.
- dbexit() מנתקת קשר קיים עם בסיס הנתונים.
- dbcmd() מעבירה את פקודת SQL אל מעבד הפקודות להכנת תוכנית גישה.
- dbresults() מחזירה את סטטוס ביצוע הפקודה.
- dbnextrow() מבקשת את השורה הבאה בטבלת התוצאה.

ממשק התכנות המלא מכיל מספר גדול יותר של פונקציות. אנו נסתפק במספר פונקציות בסיסיות אלו כדי להציג תוכנית יישום פשוטה המבצעת עדכון של הציון בטבלת הציונים. בתרשים 12.34 נראה דוגמה לקטע תוכנית עדכון עם פונקציות API.

הסבר לתרשים:

- שורה 3 מכינה את השטח עבור הנתונים הדרושים לביצוע Login לבסיס הנתונים.
- שורה 8 מכינה את השטח הדרוש לביצוע ההתקשרות.
- שורות 9 עד 11 מכינות את שם המשתמש, את הסיסמה ומבצעות את ההכנה לקראת ההתקשרות לבסיס הנתונים.
- שורה 12 מבצעת את ההתקשרות על ידי הפעלת הפונקציה dbopen.
- שורות 18 ו-19 מעבירות את פקודת העדכון למעבד SQL. נשים לב שניתן לשבור פקודה אחת למספר פקודות, אם פקודת SQL היא ארוכה ואינה נכנסת לשורה אחת.
- שורה 20 מבקשת את הפעלת תוכנית הגישה ולמעשה מבצעת את פקודת העדכון.
- שורה 21 מכניסה לתוך המשתנה sql_return_code את סטטוס ביצוע הפקודה על ידי הפעלת הפונקציה dbresults.
- שורה 27 מבצעת את ההתנתקות מבסיס הנתונים עם סיום התוכנית.

```
1.   main
2.   {
3.   LOGINREC  *loginrec;
4.   DBPROCESS      *dbproc;
5.    int       sql_return_code;
6.   short       new_grade;
7.   varchar student_number[5];
8.   char action_flag[1];
9.   loginrec = dblogin ( );
10.  DBSETLUSER (loginrec, “RON”);
11.  DBSETLPWD  (loginrec, “121234”);
12.  dbproc = dbopen (loginrec, “server-name “);
13.  do {
14.     printf (“\n Enter Student Number: “);
15.     scanf (“%s”, student_number);
16.     printf (“\n Enter new grade: “);
17.     scanf (“%s”, new_grade);
18.     dbcmd (dbproc, “UPDATE GRADES SET GRADE = new_grade”);
19.     dbfcmd (dbproc, “WHERE STUDENT_ID = student_number”);
20.     dbsqlexec (dbproc);
21.     sql_return_code = dbresults (dbproc);
22.      if (sql_return_code = = FAIL) goto student_not_found;
23.     printf (“\n Do you want to update another student (Y/N):”);
24.     scanf (“%c”, action_flag);
25.  }
26.   until (action_flag ! = ‘Y’);
27.   dbexit (dbproc);
28.  exit();
29.  student_not_found:
30.      printf (“Student not found. \n”);
31.      exit();
```

תרשים 12.34: דוגמה לקטע תוכנית עדכון עם פונקציות API.

כפי שניתן לראות, התוכנית אינה מורכבת מדי ומזכירה במידה רבה את התוכנית שכתבנו בתחילת הפרק תוך שימוש ב- Embedded SQL.

נציג עוד תוכנית אחת המשתמשת בממשק התכנות ומפעילה שאילתה. החידוש בתוכנית זו הוא שימוש בפונקציה dbbind המבצעת קישור בין עמודה בטבלה התוצאתית לבין משתנה של השפה המארחת. עבור כל עמודה המופיעה בטבלה התוצאתית מבוצע קישור נפרד עם משתנה של השפה המארחת.

דוגמה: יש לכתוב תוכנית המציגה את מספרי ושמות הסטודנטים של כל הסטודנטים הגרים בעיר מסוימת. התוצאות תוצגנה במיון לפי שם הסטודנט. התוכנית מוצגת בתרשים 12.35.

הסבר:

- שורות 9 עד 12 מבצעות את ההתקשרות לבסיס הנתונים.
- שורות 13 ו-14 מבקשות מהמשתמש להזין את שם העיר המבוקשת.
- שורות 15 עד 17 מעבירות את פקודת SQL אל המעבד להכנת תוכנית הגישה.
- שורה 18 מבצעת את השאילתה.
- שורה 19 בודקת שפקודת SQL בוצעה בהצלחה.
- שורות 20 ו-21 מקשרות בין עמודה של הטבלה התוצאתית ובין שם של משתנה בשפה המארחת שיקבל את הערכים. הפקודה מכילה את המספר הסידורי, את טיפוס הנתונים ואת האורך של העמודה וכן את שם המשתנה שיקבל את הערך.
- שורות 22 עד 26 מבצעות את הלולאה לשליפת כל אחת מהשורות בטבלת התוצאה. הפונקציה dbnextrow מביאה כל פעם שורה חדשה ומעבירה את תוכן העמודות אל המשתנים שהוגדרו על ידי הפונקציה dbbind.
- שורה 29 מבצעת את ההתנתקות מבסיס הנתונים.

```
1.  main
2.  {
3.  LOGINREC  *loginrec;
4.  DBPROCESS     *dbproc;
5.   int       sql_return_code;
6.  char student_number[5];
7.  char student_name[20];
8.  char student_city[20];
9.  loginrec = dblogin ( );
10. DBSETLUSER (loginrec, “RON”0;
11. DBSETLPWD  (loginrec, “121234”);
12. dbproc = dbopen (loginrec, “ “);
13. printf (“\n Enter Student City: “);
14. scanf (“%s”, student_city);
15. dbcmd (dbproc, “SELECT STUDENT_ID, NAME FROM”);
16. dbfcmd (dbproc, “STUDENTS WHERE CITY = student_city”);
17. dbfcmd (dbproc, “ORDER BY NAME”);
18. dbsqlexec (dbproc);
19. sql_return_code = dbresults (dbproc);
20. dbbind (dbproc, 1, NTBSSTRINGBIND, 20, &student_number);
21. dbbind (dbproc, 2, NTBSTRINGBIND, 20, &student_name);
22. while (sql_return_code = dbnextrow (dbproc) == SUCCEED)
23. {
24.     printf (“Student Number:  %s\n”, student_number);
25.     printf (“Student Name: %s\n”, student_city);
26.  }
27.  if (status = = FAIL)
28.     printf (“SQL Error. \n”);
29.  dbexit (dbproc);
30. exit();
```

תרשים 12.35: דוגמה לקטע תוכנית דיווח עם פונקציות API.

הבדלים עיקריים בין ממשק תכנות לבין פקודות משובצות

נסכם בקצרה את ההבדלים בין שתי השיטות שהצגנו לגישה לבסיס נתונים טבלאי מתוך תוכנית יישום. באופן עקרוני, הפונקציונליות של שתי שיטות עבודה אלו היא זהה, כלומר ניתן לתכנת את אותו יישום הן תוך שימוש בפקודות SQL משובצות בתוכנית היישום והן על ידי שימוש בממשק התכנות כדי להפעיל שירותים של מערכת RDBMS. ההבדל ביניהם הוא בעיקר בנקודת המוצא השונה של שתי שיטות אלו.

נקודת המוצא של Embedded SQL היא שבדרך כלל העבודה היא עם פקודות SQL סטטיות, ואילו נקודת המוצא של SQL API היא שבדרך כלל העבודה היא עם פקודות SQL דינמיות. כמובן שגם שיטת Embedded SQL תומכת בפקודות SQL דינמיות, אולם כפי שהראינו, שיטה זו מורכבת למדי ודורשת רמת מיומנות גבוהה יחסית של מהנדסי התוכנה. השימוש במבנה הדינמי של SQLDA הוא מורכב ומחייב עבודה עם שטחי זיכרון בגודל משתנה ושימוש במצביעים. מאידך, השימוש בממשק התכנות מאפשר שילוב קל יחסית גם של פקודות SQL סטטיות וגם של פקודות SQL דינמיות.

סיכום

פרק זה הציג את צורת העבודה עם שפת SQL מתוך שפת תכנות מארחת כלשהי. כפי שראינו קיימות מספר שיטות שונות לעבודה מתוך שפה מארחת: עם קדם-מהדר ופקודות סטטיות, עם קדם-מהדר ופקודות דינמיות ועם ממשק תכנות. חלק מהמערכות המסחריות תומכות במספר שיטות שונות ולכן מפתח התוכנה יכול לבחור בשיטה הנוחה והמתאימה יותר למשימה.

פרק זה הציג את המושג Cursor, המנגנון המתמודד עם השוני הבסיסי בין סביבת SQL הפועלת בתפישת One-Set-at-a-Time לבין שפות התכנות המארחות הפועלות בשיטת One-Record-at-a-Time. סביבת העבודה של SQL יכולה להחזיר בבת אחת מספר רב ובלתי ידוע מראש של שורות כתוצאה מביצוע שאילתה ולעומתה, שפות התכנות המקובלות יודעות לטפל בכל פעם בשורה אחת בלבד. מנגנון ה-Cursor מגשר בין שתי תפישות אלו ופועל למעשה כמצביע דמיוני אל השורות של השאילתה ומאפשר להתקדם בכל פעם אל השורה הבאה בתוך טבלת התוצאה באמצעות הפקודה FETCH. במידת הצורך, ניתן לבצע במגבלות מסוימות גם עדכון או ביטול של השורות שה-Cursor מצביע עליהם. את השיטה הדינמית ניתן לממש תוך שימוש בקדם-מהדר או על ידי שימוש בממשק תכנות.

השיטה הסטטית והשיטה הדינמית נבדלות זו מזו במספר מישורים:

- **ביצועים:** השיטה הסטטית מאפשרת את הכנת תוכנית הגישה לבסיס הנתונים עוד בשלב ההידור. בזמן ביצוע התוכנית, מועברים אל תוכנית הגישה הפרמטרים ומבצעים אותה. לעומת זאת בשיטה הדינמית, יש להכין את תוכנית הגישה תוך כדי ריצת התוכנית. בין אם משתמשים בגישה החד שלבית לביצוע מיידי של הפקודות ובין אם משתמשים בגישה דו-שלבית, אין ספק שהשיטה הסטטית היא בעלת ביצועים טובים יותר.
- **עדכניות:** בשיטה הסטטית מכינים את תוכנית הגישה בזמן ההידור ולכן יש להיות ערים לצורך לבנות מחדש את תוכנית הגישה מעת לעת כתוצאה משינויים במבנה בסיס הנתונים. לעומת זאת, בשיטה הדינמית תוכנית הגישה הינה עדכנית, מאחר והיא נבנית במהלך ריצת התוכנית. כל הפעלה מחדש של התוכנית תגרום לבנייה מחדש של תוכנית הגישה.
- **פשטות:** השיטה הסטטית המבוססת על קדם-מהדר ושיטת העבודה עם ממשק תכנות הן פשוטות יחסית. השיטה הדינמית המבוססת על קדם-מהדר הינה מורכבת יחסית ומחייבת רמת מיומנות גבוהה יותר של מפתחי היישומים.
- **גמישות:** השיטה הדינמית הינה גמישה יותר, מאחר והיא מאפשרת את בניית פקודות SQL במהלך ריצת התוכנית. לעומתה, השיטה הסטטית מחייבת כתיבת כל פקודות SQL מראש. יחד עם זאת, כדאי להדגיש שברוב הגדול של תוכניות היישום, מפתח היישום יודע בדיוק את הפעולות שעליו לבצע. הצורך בשיטה הדינמית הוא במיעוט המקרים ובעיקר בעת הצורך בכתיבת תוכניות כלליות יותר.

שאלות חזרה ותרגילים

שאלות חזרה

1. הסבר את ההבדל בין SQL אינטראקטיבי, SQL משובץ, SQL סטטי ו-SQL דינמי. האם ניתן להשתמש ב-SQL דינמי באופן אינטראקטיבי?
2. הסבר מה השימוש של שטח הזיכרון המיוחד SQLCA. כיצד הוא מוכנס אל תוכנית היישום?
3. הסבר מהו השימוש של הפקודה SQL WHENEVER.
4. הסבר מהו מנגנון ה- Cursor בשפת SQL.
5. הסבר את ההבדל בין שיבוץ הפקודה SELECT השולפת שורה אחת בלבד לבין הפקודה SELECT השולפת מספר שורות.
6. הסבר כיצד שפת SQL מתמודדת עם ערכי Null כאשר היא פועלת מתוך שפה מארחת.
7. מהו Scroll Cursor ומתי יש צורך להשתמש בו?

תרגילים

1. נתונה הסכימה הטבלאית הבאה:

לקוחות (קוד לקוח, שם לקוח, עיר)

פריטים (קוד פריט, שם פריט, סוג פריט, מחיר פריט)

הזמנות (קוד לקוח, קוד פריט, תאריך הזמנה, כמות מוזמנת)

א. כתוב תוכנית יישום המציגה את רשימת כל הלקוחות שהזמינו פריט מסוים בכמות גדולה מכמות מסוימת. על התוכנית לאפשר קליטת הפרמטרים קוד פריט וכמות מוזמנת.

ב. כתוב תוכנית יישום המגדילה או מקטינה באחוז מסוים את מחירי כל הפריטים מסוג Video. על התוכנית לקלוט את אחוז השינוי (מספר חיובי מציין הגדלת המחיר ומספר שלילי מציין את הוזלת המחיר).

ג. כתוב תוכנית יישום המבטלת את כל ההזמנות של לקוחות שתאריך ההזמנה שלהן הוא מתחת לשנתיים מהיום.

ד. כתוב תוכנית יישום המוסיפה שורה חדשה לטבלת הלקוחות. שים לב לכך שעיר מגורים יכולה לקבל ערך Null.

פרק 13

מזניקים ופרוצדורות בסיס נתונים

(Stored Procedures and Triggers)

1. מבוא
2. מנגנון המזניק (Trigger)
3. פרוצדורות בסיס נתונים (Stored Procedures)
4. סיכום

מבוא

בפרק הקודם הצגנו את צורת העבודה עם תוכניות יישום העובדות עם בסיס נתונים יחסי. עם הופעת טכנולוגיית שרת/לקוח (Client/Server) צריך היה לאפשר לבסיס הנתונים לבצע גם לוגיקה עסקית בנוסף לניהול הנתונים ואבטחת שלמות ואמינות בסיס הנתונים. בפרק זה נציג שתי שיטות המרחיבות את יכולת בסיס הנתונים מעבר לניהול פסיבי של נתונים: **מנגנון המזניק** (Trigger) ו**פרוצדורות בסיס נתונים** (Stored Procedures).

- **מנגנון המזניק** (Trigger): מנגנון זה של מערכת RDBMS מאפשר הפעלה אוטומטית של פרוצדורה המאוחסנת בבסיס הנתונים. פרוצדורה זו, היכולה להכיל פקודות SQL ופקודות נוספות לביצוע לוגיקה, מופעלת באופן אוטומטי על ידי מערכת RDBMS כתוצאה מאירוע כלשהו. פרוצדורה זו עוברת תהליך הידור, מאוחסנת בתוך בסיס הנתונים ומופעלת ישירות על ידי המערכת כתוצאה מהאירוע.
- **פרוצדורות בסיס נתונים** (Stored Procedures): מנגנון זה דומה למנגנון המזניק, Trigger, אולם במקרה זה הפרוצדורה אינה מופעלת אוטומטית על ידי בסיס הנתונים כתוצאה מאירוע אלא מופעלת כתוצאה מבקשה מפורשת של תוכנית יישום כלשהי. הפרוצדורה יכולה להכיל לוגיקה ופקודות SQL וניתן להדר ולאחסן אותה בבסיס הנתונים. מקובל לקרוא לתוכניות אלו בשם פרוצדורות בסיס נתונים, פרוצדורות מאוחסנות, או Stored Procedures. תוכנית היישום יכולה לבקש את הפעלת הפרוצדורה על ידי שימוש בממשק תכנות (API) ולהעביר לה פרמטרים, ובגמר הביצוע שלה – לקבל את התוצאות. הדבר שונה מהשיטות שהוצגו בפרק הקודם ושבהן כל הלוגיקה נכתבה בשפה המארחת והאינטראקציה בין תוכנית היישום ומערכת RDBMS היתה ברמה של פקודות SQL בודדות. פרוצדורות בסיס הנתונים מרחיבות מודל זה ומאפשרות את פיצול הלוגיקה בין תוכנית היישום לבין פרוצדורת בסיס הנתונים.

שני מנגנונים אלה מרחיבים את יכולות בסיס הנתונים, הן מבחינת הפעלת כללי אמינות ושלמות מתוחכמים ומורכבים שקשה לבטא אותם באמצעות הכרזות (Assertions) והן מבחינת היכולת לפצל את הלוגיקה של היישום בין תוכניות היישום לבין בסיס הנתונים. שני מנגנונים אלה, מנגנון המזניק ופרוצדורות בסיס נתונים, הביאו לכך שמקובל לכנות את מערכות RDBMS בשם **מערכות אקטיביות**.

מערכות ניהול בסיסי נתונים הראשונות התמקדו בניהול הנתונים בלבד וכל הלוגיקה היתה באחריות תוכניות היישום ומחוץ לבסיס הנתונים. לעומת גישה פסיבית זו, המערכות החדשות לניהול בסיסי נתונים טבלאיים משלבות את שני המנגנונים האלה. הן מכילות לוגיקה של יישום וגם מבצעות אותה באופן אוטומטי במקרה של אירוע, או על פי בקשת תוכנית היישום להפעלת פרוצדורות בסיס נתונים. תכונות אלו גרמו לכך שיצרנים מסוימים כינו את המערכות שלהם בשם היומרני: **בסיסי נתונים אינטליגנטיים**. כמובן שאין כאן שום אינטליגנציה, אלא רק יכולת לאחסן ולהפעיל לוגיקה, אך ללא ספק זהו צעד קדימה בכל האמור ליכולות של מערכות ניהול בסיסי נתונים.

שני מנגנונים אלה היו מחוץ לגבולות תקן SQL1 ואפילו תקן SQL2, ולכן התפתחו ויושמו בצורות שונות במערכות RDBMS המסחריות. רק תקן SQL3, הנמצא כבר מספר שנים במצב של טיוטא ועדיין לא פורסם באופן רשמי, מתייחס למנגנון המזניק ולפרוצדורות בסיס נתונים. קיים ויכוח בין מומחים שונים העוסקים בנושא בסיסי נתונים האם יש בכלל לכלול נושא כזה בתקן. לטענת אחדים מהם מנגנונים אלה אינם טבעיים לבסיס הנתונים ובמובן מסוים "מזהמים" את התפישה של בסיס הנתונים כמאגר של נתונים. המצדדים בהכללת הנושא בתקן טוענים שיכולות אלו הן הכרחיות בבסיס נתונים מודרני הפועל בסביבת שרת/לקוח, ולכן מן הדין שהנושא יקבל את תשומת הלב ואת הסטנדרטיזציה שתבוא כתוצאה מהכללת הנושא בתקן.

למרות היעדר תמיכת התקן בנושאים אלה, הם הפכו מנגנונים אלה לנפוצים ומקובלים מאוד, הנתמכים כמעט על ידי כל יצרני מערכות RDBMS וניתן לומר שרוב היישומים המודרניים המבוססים על טכנולוגיית שרת/לקוח משתמשים בהם צורה זו או אחרת. הבעיה העיקרית הנובעת ממצב זה היא כמובן חוסר התאימות הכמעט מוחלט בין צורת המימוש של היצרנים השונים ותלות כמעט מוחלטת של מפתח התוכנה במערכת בסיס הנתונים המסוימת. יכולת הניידות (Portability) בין מערכות RDBMS שונות נפגעת בצורה קשה כתוצאה משימוש במנגנונים לא סטנדרטיים אלה. התייחסות תקן SQL3 לשני מנגנונים אלה והתמיכה העתידית של היצרנים בתקן נותנת את התקווה שמצב ענינים זה ישתפר באופן משמעותי בעתיד.

בפרק זה

- נכיר דרכים לניהול אקטיבי של בסיס הנתונים: מנגנון המזניק (Trigger) ופרוצדורות בסיס נתונים (Stored Procedures).
- נציג את התועלות משימוש במזניק ובפרוצדורות, במיוחד בתחום אכיפת כללי אמינות, שלמות ואבטחה.

מנגנון המזניק (Trigger)

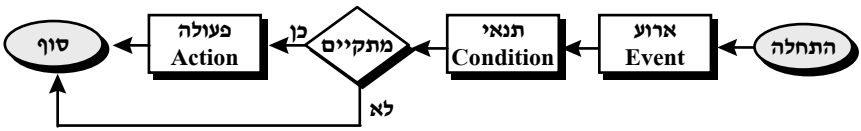
כפי שראינו בפרק שעסק בהגדרת בסיס הנתונים, בסיס הנתונים חייב לקיים ולאכוף אילוצים מסוימים כדי לשמור על אמינותו ושלמותו. עד כה ראינו שניתן לבטא את האילוצים בשתי צורות עיקריות:

- **כחלק מהגדרת הטבלה:** אילוצים אלה מוגדרים כחלק מהגדרת הטבלה ובעת הגדרתה. האילוצים יכולים להתייחס למרחב הערכים של עמודות, להגדרת המפתח העיקרי המבטא אילוץ של חד-ערכיות הערכים המופיעים במפתח העיקרי, לכללי השלמות של הקשרים בין הטבלאות (Referential Integrity), או לכללים מיוחדים שניתן לבטא אותם על ידי בדיקת קיום או אי קיום של תנאי המוגדר על ידי שאילתה.

❖ **הכרזות (Assertions):** כחלק מהגדרת בסיס הנתונים ניתן להכריז על אוסף של חוקים שבסיס הנתונים חייב לקיים בכל עת ולכן מערכת RDBMS תבדוק את קיומם בכל אירוע של עדכון.

נכיר עכשיו מנגנון נוסף של מערכת RDBMS – מנגנון המזניק – המרחיב את יכולות בסיס הנתונים מעבר לניהול נתונים בלבד. ניתן לחלק מנגנון זה לשלושה מרכיבים שונים: האירוע שגורם להפעלת המנגנון, התנאי שנבדק כדי להחליט האם לבצע את הפעולה או לא לבצעה והפעולה עצמה. מסיבה זו מקובל לקרוא למנגנון זה גם בשם Event-Condition-Action.

מזניק Trigger	***מזניק*** **מוגדר כתוכנית המאוחסנת בבסיס הנתונים ומופעלת אוטומטית כתוצאה מאירוע, כגון הוספת שורה, עדכון שורה, ביטול שורה וכד'. תוכנית זו יכולה לבצע בדיקות תקינות, לעדכן נתונים באופן נסתר, להפעיל תוכנית אחרת ועוד.**



תרשים 13.1: עקרון הפעולה של מנגנון המזניק.

האירוע המפעיל את המנגנון הזה מזכיר את פעולת הלחיצה על ההדק של אקדח הזנקה בתחרות אתלטיקה, אשר גורמת להזניק את הרצים אל המסלול.

מנגנון זה הופיע לראשונה במערכת SQL Server של חברת Sybase והפך לנפוץ מאוד ונתמך כיום על ידי רוב המערכות המסחריות. המנגנון 'מזניק' – Trigger – נוח מאוד להגדרת אילוצים שונים שעל בסיס הנתונים לקיים, או לחילופין להפעלה אוטומטית של פעולות שונות (כמו למשל שיגור הודעה בדואר אלקטרוני בכל פעם שיתרת המלאי של פריט יורדת מתחת לרמת מלאי כלשהי).

נתבונן על צורת הגדרת TRIGGER בתקן SQL3, שנמצא עוד במצב טיוטא ועלול להשתנות.

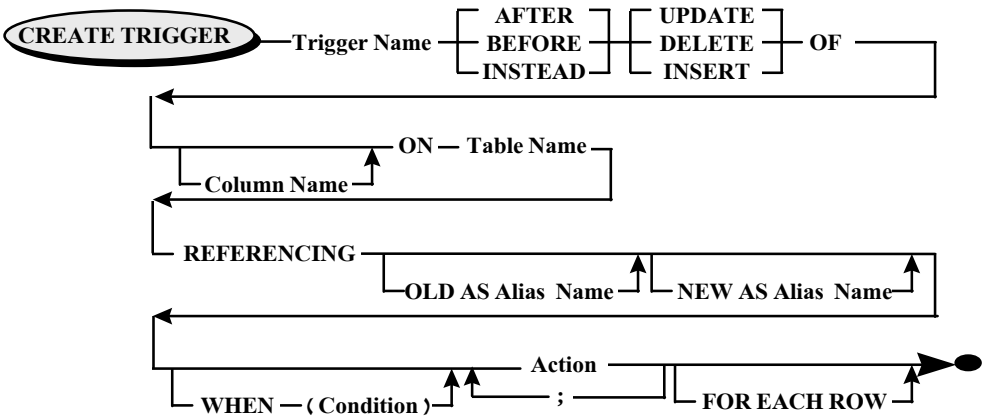
נפרט בקצרה את האפשרויות השונות בהגדרת TRIGGER, אשר מוצגת בתחביר הפקודה שבתרשים 13.2:

❖ לכל Trigger יש לתת שם חד-ערכי.

❖ ניתן לקבוע אם הפעלת ה-Trigger תהיה לפני, אחרי או בזמן האירוע. המשמעות של הפרמטר INSTEAD היא שאם יתקיים התנאי, תתבצע פעולת המזניק במקום האירוע שהפעיל אותו. לדוגמה אם האירוע שהפעיל את המזניק הוא פקודת ביטול, ניתן לקבוע שאם מתקיים תנאי מסוים תופעל פקודת עדכון שתבצע עדכון של טבלה כלשהי במקום פעולת הביטול.

- האירוע שיכול להפעיל Trigger עשוי להיות פקודת עדכון, הוספה או ביטול של שורה בטבלה.
- עבור פעולת עדכון ניתן להגדיר האם להפעיל Trigger ללא תלות בעמודה המתעדכנת, או להפעילו רק אם עמודה מסוימת מתעדכנת.
- ניתן להתייחס אל הערכים הקיימים בטבלה לפני הפעלת פקודת עדכון או ביטול או אל הערכים החדשים במקרה של הוספה או עדכון.
- ניתן להגדיר תנאי שייבדק לפני ביצוע הפעולה כך שהפעולה תבוצע רק אם התנאי מחזיר את הערך True. התנאי יכול להתייחס אל הערכים החדשים ואל הערכים הקיימים, תוך שימוש בשמות שנקבעו לכל אחד מהמצבים. לדוגמה, ניתן לבדוק תנאי אם הציון לאחר העדכון גדול ב- 50% מהציון לפני העדכון.
- הפעולה מוגדרת על ידי פקודת SQL חוקית. ניתן לבצע מספר פקודות SQL ברצף אחת אחר השנייה. יש להפריד בין פקודות SQL על ידי נקודה פסיק.
- ניתן להגדיר האם Trigger יופעל פעם אחת בלבד עבור האירוע, או עבור כל שורה. אירוע אחד, למשל עדכון, יכול לעדכן מספר רב של שורות בטבלה. אם בוחרים בפרמטר FOR EACH ROW התנאי ייבדק מחדש עבור כל שורה והפעולה תבוצע עבור כל אחת מהשורות המתעדכנות, כמובן אם התנאי מתקיים.

תחביר הפקודה:



תרשים 13.2: תרשים תחביר של הפקודה CREATE TRIGGER.

כדי להדגים את השימוש במנגנון המזניק, Trigger, נבצע שינוי קל בבסיס הנתונים לדוגמה ונוסיף לטבלת המחלקות שתי עמודות חדשות. עמודה אחת תכיל את סה"כ נקודות הזכות שהקורסים מסוג מעבדה המועברים על ידי המחלקה יכולים להעניק. העמודה השנייה תכיל את מונה מספר הקורסים שהמחלקה מעבירה. שתי העמודות האלו הן דוגמאות לנתונים שניתן לחשב באמצעות פקודות SELECT מתאימות, אולם לפעמים מתוך שיקולים של ביצועים מעוניינים להכניס כפילות נתונים מסוימת אל בסיס הנתונים.

אם השאילתה לחישוב סה"כ נקודות הזכות שכל הקורסים שמחלקה מעניקה היא שאילתה מאוד נפוצה ואולי גם יקרה מבחינת משאבי מחשב, רצוי לנהל נתון זה בתוך טבלת המחלקות ואז לענות במהירות רבה וביעילות על השאילתה. כמובן שהכנסת כפילות נתונים אל בסיס הנתונים היא מסוכנת בגלל בעיות של חוסר אמינות. השימוש במנגנון Trigger יכול להבטיח שכפילות הנתונים תנוהל באופן אוטומטי על ידי מערכת RDBMS ללא התערבות תוכניות היישום.

נציג תחילה את המבנה החדש של טבלת מחלקות:

Departments

DEPART	NAME	HEAD	TOTAL_LAB_POINTS	NO_COURSES
CS	Computer Science	Dr. Israel	8	3
MT	Mathematics	Prof. Levy	0	2
BS	Business	Dr. Eyal	0	3
CH	Chemistry	Prof. Doron	Null	Null

תרשים 13.3: מבנה טבלת מחלקות עם שתי עמודות חדשות.

דוגמה א': יש לבנות מנגנון Trigger שיעדכן את מונה מספר הקורסים של מחלקה מסוימת כתוצאה מהוספת קורס חדש לטבלת קורסים.

```
1.  CREATE TRIGGER INCREASE_NO_OF_COURSES
2.      AFTER INSERT
3.      ON COURSES
4.      REFERENCES NEW AS NEW_COURSE
5.      UPDATE DEPARTMENTS
6.          SET     NO_COURSES = NO_COURSES + 1
7.          WHERE DEPART = NEW_COURSE.DEPARTMENT_ID
```

הסבר:

- שורה 1 קובעת את שם המזניק.
- שורות 2 ו-3 קובעות שהאירוע שיפעיל את המזניק הוא פקודת הוספה של שורה חדשה לטבלת קורסים.
- שורה 4 מגדירה שם, NEW_COURSE, לשורה החדשה.
- שורות 5 עד 7 מגדירות את הפעולה שתבוצע במקרה של הוספת שורה חדשה לטבלת קורסים. כפי שניתן לראות פקודת העדכון תגדיל את מונה מספר הקורסים עבור השורה השייכת למחלקה המתאימה. בשורה 7 ניתן לראות כיצד משתמשים בשם NEW_COURSE כדי להתייחס לערכים הבאים מתוך השורה החדשה.

נשים לב שלא הגדרנו תנאי, ולכן מזניק זה מופעל בכל הוספת שורה לטבלת קורסים. באופן דומה ניתן לבנות מזניק שיעדכן את מונה מספר הקורסים כתוצאה מביטול קורס בטבלת קורסים.

דוגמה ב': יש לבנות מנגנון Trigger שיעדכן את מונה מספר נקודות הזכות לקורסים מסוג מעבדה כתוצאה מביטול קורס בטבלת קורסים.

```
1.  CREATE TRIGGER DECREASE_TOTAL_LAB_POINT
2.     AFTER DELETE
3.     ON COURSES
4.     REFERENCES OLD AS OLD_COURSE
5.     WHEN (OLD_COURSE.TYPE = 'LAB')
6.     UPDATE DEPARTMENTS
7.         SET  TOTAL_LAB_POINTS = TOTAL_LAB_POINTS - 1
8.         WHERE DEPART = OLD_COURSE.DEPARTMENT_ID
```

נשים לב שבשורה 5 הגדרנו תנאי הקובע, שהפעולה תבוצע רק אם השורה המבוטלת העמודה של סוג הקורס מכילה את הערך LAB.

דוגמה ג': יש לבנות מנגנון Trigger שיעדכן את מונה מספר נקודות הזכות לקורסים מסוג מעבדה כתוצאה מעדכון מספר נקודות הזכות שקורס מעבדה כלשהו מעניק.

```
1.  CREATE TRIGGER UPDATE_TOTAL_LAB_POINT
2.     AFTER UPDATE
3.     OF POINTS
4.     ON COURSES
5.     REFERENCES NEW AS NEW_COURSE
6.     REFERENCES OLD AS OLD_COURSE
7.     WHEN (OLD_COURSE.TYPE = 'LAB')
8.     UPDATE DEPARTMENTS
9.         SET  TOTAL_LAB_POINTS = TOTAL_LAB_POINTS +
10.            (NEW_COURSE.POINTS - OLD_COURSE.POINTS)
11.        WHERE DEPART = OLD_COURSE.DEPARTMENT_ID
```

הסבר:

- שורות 2 עד 4 קובעות שהאירוע להפעלת המזניק הוא עדכון של העמודה POINTS בטבלת קורסים. משמעות הדבר הוא שאם יבוצע עדכון של עמודה אחרת בטבלת קורסים, המזניק **לא** יופעל.
- שורות 5 ו-6 קובעות את שמות הערכים החדשים והישנים.
- שורות 9 ו-10 מעדכנות את העמודה TOTAL_LAB_POINTS על ידי הוספת ההפרש בין מספר נקודות הזכות לאחר העדכון ומספר נקודות הזכות לפני העדכון.

זוהי הגדרת מנגנון המזניק על פי טיוטת תקן SQL3, אך רוב המערכות המסחריות עדיין אינן תומכות בהגדרה זו. נתבונן כיצד ניתן לכתוב מנגנון לעדכון מונה הקורסים בשפת Transact-SQL במערכת SQL Server של Sybase, שהציגה לראשונה את מנגנון המזניק.

```
1.  CREATE TRIGGER INCREASE_NO_OF_COURSES
2.     ON COURSES
3.     FOR INSERT
4.     AS UPDATE DEPARTMENTS
5.         SET       NO_COURSES = NO_COURSES + 1
6.         WHERE DEPARTMENTS.DEPART = INSERTED.DEPARTMENT_ID
```

כפי שניתן לראות קיים דמיון בין צורת הגדרה זו לצורת ההגדרה של תקן SQL3. נשים לב שבמקום לאפשר הגדרת שם לערכים החדשים, המוסכמה כאן היא שהערכים החדשים מקבלים קידומת INSERTED ואילו אלה המבוטלים יקבלו קידומת DELETED. בנוסף ליכולת להגדיר פקודות SQL כפעולה, מאפשרת השפה גם להשתמש במבנה If-Then-Else, פקודות להפעלת פרוצדורות חיצוניות, פקודות הדפסה וכד'.

Sybase השתמשה במנגנון המזניק כדי להגדיר את כללי שלמות הקשרים (Referential Integrity), להבדיל משיטת הגדרת הכללים תוך כדי הגדרת הטבלה, כפי שדורש תקן SQL2. לדוגמה, המנגנון הבא מטפל במצב שבו מעדכנים את מספר המחלקה בטבלת מחלקות וכתוצאה מכך יש לעדכן את מספר המחלקה בכל הקורסים שהמחלקה מעבירה. זוהי דוגמה למימוש הכלל UPDATE CASCADED בתקן SQL2 וכיצד הוא ממומש ב- SQL Server באמצעות מזניק.

```
1.  CREATE TRIGGER CASCADE_UPDATE_DEP_ID
2.     ON DEPARTMENTS
3.     FOR UPDATE
4.     AS  IF UPDATE (DEPART)
5.           BEGIN
6.              UPDATE COURSES
7.                 SET      COURSES.DEPARTMENT_ID = INSERTED.DEPART
8.                 FROM     DEPARTMENTS,  INSERTED, DELETED
9.                 WHERE COURSES.DEPARTMENT_ID = DELETED.DEPART
10.          END
```

כפי שניתן לראות, שורות 2 עד 4 קובעות שהפעלת המזניק תהיה כתוצאה מעדכון העמודה DEPART בטבלת מחלקות בלבד. בעת הפעלת המזניק יתבצע עדכון של כל השורות בטבלת קורסים כך שמספר המחלקה החדש יוכנס אל העמודה DEPARTMENT_ID.

לסיכום נושא זה נוכל לומר, שמנגנון המזניק הינו מנגנון מתוחכם המאפשר לבצע בבסיס הנתונים פעולות שונות באופן אוטומטי כתוצאה מאירועים שונים. מאחר והאחריות על הפעלת המזניק היא של מערכת RDBMS נוכל להיות בטוחים שפעולות אלו תבוצענה בכל מקרה וללא כל התערבות ולעיתים גם ללא ידיעת מפתחי היישומים. יחד עם היתרונות של מנגנון זה חשוב להדגיש גם את החסרונות שלו, המתמקדים בעיקר ברמת מורכבות נוספת. מפתחי היישומים או המשתמשים לא תמיד מודעים לקיום המנגנון ובמקרה של תקלות לא תמיד ברור מה מקור התקלה. פעולה גלויה על בסיס הנתונים מפעילה אוסף של פעולות נסתרות ואוטומטיות ולכן זה מוסיף רובד נוסף של סיבוכיות.

פרוצדורות בסיס נתונים (Stored Procedures)

מנגנון המזניק, Trigger, שהוצג בסעיף הקודם הוא מנגנון המופעל אוטומטית על ידי מערכת RDBMS כתוצאה מאירוע כלשהו. ניתן לקחת את התפישה העומדת מאחורי מנגנון המזניק צעד אחד קדימה, ולאפשר את הפעלתו בצורה יזומה על ידי תוכנית היישום ולא על ידי מערכת RDBMS. על ידי הוספה של אפשרויות נרחבות בכל הקשור לפקודות רגילות לכתיבת לוגיקה עם שילוב של פקודות SQL ומתן אפשרות הפעלה ישירה על ידי תוכניות היישום, ניתן לקבל מנגנון רב עוצמה. זהו מנגנון **פרוצדורות בסיס הנתונים**.

פרוצדורת בסיס נתונים **Stored Procedure**	***פרוצדורת בסיס נתונים*** **מוגדרת כתוכנית המאוחסנת בבסיס הנתונים ומופעלת על פי בקשת תוכנית יישום, תוך שימוש הממשק תכנות כלשהו. התוכנית יכולה לבצע לוגיקה עסקית, בדיקות תקינות, עדכון בסיס הנתונים ועוד.**

מכיון שפרוצדורת בסיס נתונים מכילה פקודות SQL המשובצות בתוך לוגיקה כללית יותר, מקובל לכנות אותה גם בשם **אובייקט SQL מתמיד** – **PSO** (Persistent SQL Object). לכל אובייקט מסוג זה יש שלוש תכונות עיקריות:

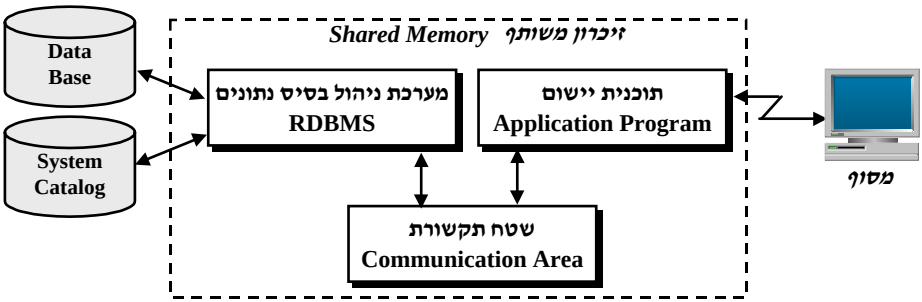
- **נשמר לאורך זמן, מתמיד (Persistent)**: כל פרוצדורה מאוחסנת בתוך בסיס הנתונים ואינה תלויה בתוכנית יישום מסוימת. להבדיל מפקודת SQL בודדת, הקיימת רק כחלק מתוכנית יישום, פרוצדורת בסיס נתונים היא אובייקט בעל קיום עצמאי משלו.
- **בעל שם חד-ערכי (Named)**: לכל פרוצדורה יש שם חד-ערכי המאפשר פנייה אליה בעת הצורך.
- **שיתופי (Shared):** מספר רב של תוכניות יכולות להפעיל אותו. להבדיל מפקודות SQL, המהוות חלק מתוכנית יישום ולכן הן אינן נגישות על ידי תוכניות יישום אחרות, פרוצדורת בסיס נתונים יכולה לשרת מספר רב של תוכניות יישום.

גישה לבסיס נתונים על ידי זיכרון משותף

השיטות שהוצגו בפרק הקודם שדן בגישה אל בסיס נתונים טבלאי מתוך שפת תכנות מארחת, היו מבוססות על הפרדה מאוד ברורה בין התפקידים: שפת התכנות מכילה ומבצעת את כל הלוגיקה העסקית ומערכת RDBMS מנהלת את הנתונים ומספקת אותם לתוכנית היישום על פי בקשתה. תוכנית יישום יכולה לגשת אל בסיס הנתונים על ידי שיבוץ פקודות SQL בתוך השפה המארחת ושימוש בקדם-מהדר, או לחילופין על ידי גישה לפונקציות שמערכת RDBMS מספקת באמצעות ממשק התכנות. ממשק התכנות מאפשר לתוכנית היישום לבקש את המערכת לבצע מטלות שונות.

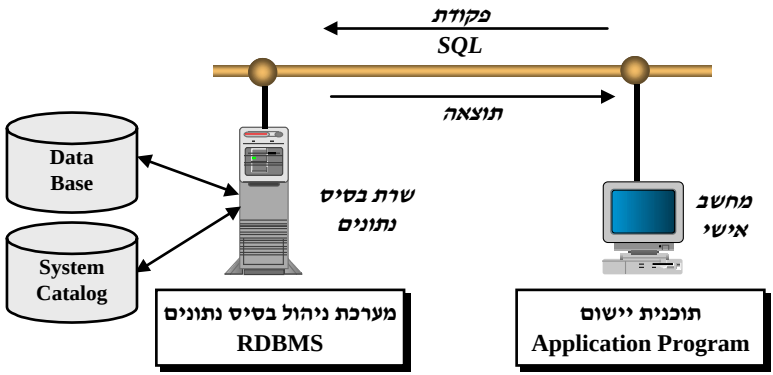
בכל מקרה, כל השיטות שהוצגו מאפשרות הידודיות (אינטראקציה) בין תוכנית היישום לבין המערכת ברמת פקודת SQL בודדת. תוכנית היישום משגרת פקודת SQL, שעברה הידור אם עובדים בשיטה סטטית, או שצריכה לעבור הידור אם עובדים בשיטה דינמית. מערכת RDBMS מקבלת את הפקודה, מבצעת אותה ומחזירה את התוצאות אל תוכנית היישום. כל בקשה מחייבת הידודיות נוספת בין תוכנית היישום לבין המערכת.

השיטה שתארנו פועלת היטב כל עוד תוכנית היישום ומערכת RDBMS נמצאות באותו מחשב. התקורה הנובעת מרמה גבוהה של הידודיות ביניהן סבירה והפעולות מהירות, כי התקשורת בין תוכנית היישום לבין המערכת מתבצעת תוך שימוש בשטח זיכרון משותף.



תרשים 13.4: הידודיות בין תוכנית יישום לבין מערכת RDBMS הפועלות בזיכרון המחשב.

השיטה המוצגת בעייתית מאוד ברגע שתוכנית היישום פועלת במחשב אחד ומערכת RDBMS פועלת במחשב אחר; ובמילים אחרות – כאשר עובדים בשיטת שרת/לקוח (Client/Server). הסבר מקיף יותר לנושא שרת/לקוח מובא בפרק 16 המציג את נושא **ביזור בסיס הנתונים**. בשלב זה נסתפק רק בהבנה שבטכנולוגיית שרת/לקוח, תוכנית היישום יכולה לפעול על מחשב אחד, למשל מחשב אישי על שולחנו של המשתמש, ואילו מערכת RDBMS יכולה לפעול על מחשב נפרד, שרת בסיס הנתונים (DataBase Server). ברגע שיוצרים הפרדה זו, התקשורת בין תוכנית היישום לבין מערכת המבוססת על רשת התקשורת המחברת את שני המחשבים.

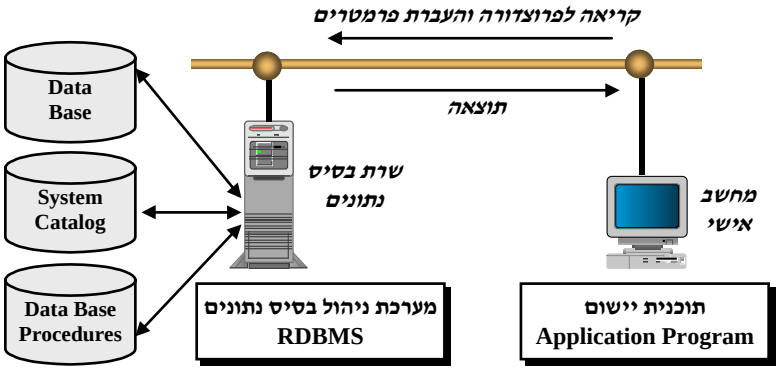


תרשים 13.5: הידודיות בין תוכנית יישום לבין מערכת RDBMS בסביבת שרת/לקוח.

פרוצדורות בסיס נתונים

רשת התקשורת היא תווך איטי יחסית למהירות הפעולה של זיכרון המחשב, ולכן הידודיות רבה עלול לגרום בעיית ביצועים קשה. כדי לטפל בבעיה זו התפתחה שיטה נוספת לעבודה עם בסיסי נתונים יחסיים, שיטת **פרוצדורות בסיס הנתונים**, או **פרוצדורות מאוחסנות**.

שיטה זו מאפשרת כתיבת לוגיקה – תוכנית – המשלבת בתוכה פקודות SQL, הידור התוכנית ושמירתה כחלק מבסיס הנתונים. מערכת RDBMS מספקת ממשק תכנות המאפשר לתוכנית היישום להפעיל את הפרוצדורה ולמסור לה פרמטרים. בגמר ביצוע הפרוצדורה, המערכת מחזירה אל תוכנית היישום את התוצאה. תוצאה זו יכולה להיות פרמטר אחד או יותר, או אוסף של שורות בדומה לתוצאה שמציגה הפקודה SELECT. יתרון השיטה ברור ובולט: ניתן לכתוב לוגיקה, או תוכנית, פעם אחת, להדר אותה, וכך להכין מראש את כל תוכניות הגישה לבסיס הנתונים ולאחסן אותן בבסיס הנתונים עצמו. הלוגיקה יכולה להיות מורכבת מאוד ויכולה לדרוש ביצוע של מספר רב של פקודות SQL שונות. תוכנית היישום יכולה להפעיל את הפרוצדורה על ידי קריאה לפרוצדורה על פי שמה, להעביר לה פרמטרים ולהמתין לקבלת התוצאה.



תרשים 13.6: הידודיות בין תוכנית יישום לבין מערכת RDBMS עם פרוצדורות בסיס נתונים.

במובן מסוים ניתן להתייחס לשיטת עבודה זו כשיטה **מבוססת אובייקטים** (Object Based). תוכנית היישום אינה מכירה את הלוגיקה ואת הגישות הנדרשות אל בסיס הנתונים. התוכנית רק יודעת מה הפונקציה שהפרוצדורה מבצעת ואת הפרמטרים שיש להעביר לה ומה התוצאה המתקבלת. נשתמש בדוגמה כדי להמחיש את האמור לעיל.

דוגמה: נניח שלצורך חישוב הזכאות לתואר של סטודנט יש לבצע בדיקה בכמה קורסים הסטודנט למד. אם למד ב-20 קורסים לפחות יש לחשב את הממוצע המשוקלל של ציון הקורס לפי מספר נקודות הזכות. אם הממוצע המשוקלל גדול מ-700 הוא זכאי לתואר. זו דוגמה ללוגיקה מורכבת יחסית שתחזור על עצמה בכל תוכנית יישום המבקשת לבדוק את הזכאות לתואר. במקום זאת, ניתן לבנות פרוצדורה בשם check_degree, למשל, שתקבל כפרמטר את מספר הסטודנט ותחזיר כפרמטר תשובה (כן או לא) ואת הציון

המשוקלל של הסטודנט. פרוצדורה זו מכילה לוגיקה ולפחות שתי פקודות SQL שונות. הפרוצדורה תעבור הידור ותאוחסן בבסיס הנתונים. תוכנית יישום המבקשת לבדוק זכאות לתואר תפעיל את check_degree ותקבל כתשובה שני ערכים. בדרך זו התעבורה ברשת התקשורת קטנה באופן משמעותי לעומת מצב שבו תוכנית היישום היתה משגרת בעצמה את פקודות SQL, מקבלת את טבלאות התוצאה ומבצעת את החישובים בעצמה.

הלוגיקה של פרוצדורת בסיס הנתונים יכולה להיכתב בשפה מיוחדת של יצרן בסיס הנתונים או בשפת תכנות כללית. עם הופעת שיטה זו ובגלל היעדר כל התייחסות של תקן SQL לשיטה הידודית זו, פתר כל יצרן בסיס נתונים את הבעיה באמצעות שפה מיוחדת וייעודית משלו. לדוגמה, מערכת Oracle תומכת בשפה הייעודית PL/SQL, ומערכות Sybase SQL ו- Microsoft SQL Server תומכות בשפה הייעודית Transact-SQL. לעומת מערכות אלו, DB2 מאפשרת כתיבת פרוצדורות בסיס נתונים בשפת תכנות רגילה כמו C. במהדורות החדשות יותר של רוב היצרנים מופיעה שפת Java כאחת השפות בהן ניתן לכתוב פרוצדורות בסיס נתונים.

כל מערכות RDBMS המסחריות הפועלות כיום, למעט SQL Server של מיקרוסופט, תומכות בכתיבת הלוגיקה של פרוצדורות בסיס נתונים בשפת Java. לדוגמה, DB2 מכילה רכיב חדש בשם Stored Procedure Builder, המסוגל לחולל אוטומטית מחלקות (Classes) עבור Java ומכילות בתוכן את פקודות SQL.

דוגמה לבניית פרוצדורה

נציג דוגמה לבניית פרוצדורה פשוטה במערכת SQL Server המבצעת חישוב של ציון משוקלל לסטודנט. בדוגמה זו לא השתמשנו בלוגיקה כלשהי מעבר לפקודת SQL רגילה.

```
1 .  create procedure weighted_grade (@student_number char) as
2.   if @student_number is null
3 .        print“Student Number is Missing” 624
4.   else
5 .     SELECT NAME, SUM (GRADE * POINTS /100) AS W_GRADE
6.     FROM    STUDENTS  S, GRADES  G, COURSES  C
7 .     WHERE GRADE >=    60  AND
8.                 S.STUDENT_ID = G.STUDENT_ID AND
9.                 G.COURSE_ID = C.COURSE_ID AND
10.                S.STUDENT_ID = @student_number
11.   GROUP BY  NAME;
```

כפי שניתן לראות, פרוצדורה זו מקבלת כפרמטר רק את מספר הסטודנט ומבצעת את הפקודה SELECT. אם מספר הסטודנט מכיל Null, הפרוצדורה מחזירה הודעת שגיאה. תוכנית היישום המפעילה פרוצדורה זו משתמשת בממשק התכנות של SQL Server המאפשר להפעיל פרוצדורת בסיס נתונים.

הסבר לתוכנית שלהלן:

- שורות 15 ו-16 מבקשות את הפעלת פרוצדורת בסיס הנתונים עם הפרמטר של מספר הסטודנט.
- שורה 17 מבצעת את הפרוצדורה.
- שורה 19 ו-20 מבצעות את הקישור בין העמודות בתוצאה לבין שמות המשתנים בתוכנית.

```
1.  main
2.  {
3.  LOGINREC  *loginrec;
4.  DBPROCESS     *dbproc;
5.   int       sql_return_code;
6.  char student_number[5];
7.  char student_name[20];
8.  short w_grade;
9.  loginrec = dblogin ( );
10. DBSETLUSER (loginrec, “RON”0;
11. DBSETLPWD  (loginrec, “121234”);
12. dbproc = dbopen (loginrec, “ “);
13. printf (“\n Enter Student Number: “);
14. scanf (“%s”, student_number);
15. dbcmd (dbproc, “execute weighted_grade”);
16. dbcmd (dbproc, student_number);
17. dbsqlexec (dbproc);
18. sql_return_code = dbresults (dbproc);
19.  dbbind (dbproc, 1, NTBSSTRINGBIND, 20, &student_name);
20. dbbind (dbproc, 2, FLT4BIND, 0, &w_grade);
21. if (sql_return_code != SUCCEED)
22. {
23.    printf (“Unknown Student Number %s\n”, student_number);
24.    exit ( );
25.  }
26.  printf (“Student Name: %s   Weighted Grade:
27.  %f\n”, student_name, w_grade);
28.  dbexit (dbproc);
29. exit();
```

כפי שניתן לראות, מנקודת המבט של תוכנית היישום ההפעלה של פרוצדורת בסיס נתונים הינה תהליך פשוט מאוד. המשך העבודה לאחר הפעלת הפרוצדורה דומה מאוד לשיטת העבודה עם ממשק תכנות רגיל. אם תוכנית היישום הזו היתה פועלת בסביבת שרת/לקוח, הביצועים שלה היו טובים יותר מאשר אותה תוכנית שעובדת עם ממשק תכנות רגיל. ברשת התקשורת נשלחת בכיוון אחד בקשה פשוטה עם פרמטר, במקום להעביר את כל פקודת SQL. מנקודת המבט של שרת בסיס הנתונים, הפרוצדורה כבר עברה הידור ותוכנית הגישה לבסיס הנתונים כבר הוכנה. עם קבלת הבקשה להפעלתה, ניתן מיידית להפעיל את תוכנית הגישה עם הפרמטר המתאים ולהחזיר חזרה דרך רשת התקשורת את שתי העמודות.

החסרון העיקרי של פרוצדורות בסיס נתונים היא שפרוצדורות אלו מבוססות על קשר הדוק מאוד בין השפה שבה היא נכתבת לבין מערכת RDBMS. השוני הגדול בין השפות השונות לכתיבת פרוצדורות בסיס נתונים, היעדר התקינה בנושא והקשר ההדוק בין השפה לבין בסיס הנתונים, כל אלה גורמים לפגיעה ביכולת הניידות (Portability) של יישומים המבוססים על פרוצדורות בסיס נתונים לעבודה עם בסיסי נתונים שונים. יישום שעובד עם SQL Server ומשתמש בפרוצדורות הכתובות בשפת Transact_SQL ידרוש מאמץ שיכתוב משמעותי כדי להעבירו לעבודה עם Oracle ופרוצדורות הכתובות בשפת PL/SQL. יצרני תוכנה המבקשים לכתוב יישומים הפועלים עם בסיסי נתונים שונים, נמנעים מלהשתמש בפרוצדורות בסיס הנתונים ומעדיפים להשתמש בשיטות אחרות. למשל, שימוש במוניטור תנועות (TP Monitor) כמו Tuxedo וכתיבת לוגיקה, הפועלת על מחשב השרת והכתובה בשפה סטנדרטית כמו COBOL או C.

סיכום

פרק זה הציג שני מנגנונים לביצוע פעולות בסיס נתונים. מנגנון המזניק, Trigger, המופעל באופן אוטומטי על ידי מערכת RDBMS כתוצאה מאירוע כלשהו ומנגנון פרוצדורת בסיס נתונים המופעל על פי בקשה של תוכנית יישום. אין ספק, ששני מנגנונים אלה מעשירים את יכולות בסיס הנתונים ומתאימים במיוחד ליישומים הפועלים בסביבת שרת/לקוח.

השימוש בפרוצדורות בסיס נתונים מתאים מאוד ליישומים בסביבת שרת/לקוח, כאשר תוכנית היישום ומערכת RDBMS פועלות על מחשבים נפרדים הקשורים ביניהם ברשת תקשורת. למרות שיפור הביצועים והגמישות בעבודה עם פרוצדורות בסיס נתונים, יש לשקול היטב שימוש בהן בגלל היעדר תקן אחיד ומחויבות של היצרנים לסביבת עבודה זו.

מנגנון המזניק שימושי מאוד בסביבת שרת/לקוח והוא שימושי גם בסביבה רגילה שבה תוכנית יישום ומערכת RDBMS פועלות באותו שרת. המנגנון מאפשר בניית חוקים וכללים מתוחכמים לשמירה על אמינות ושלמות בסיס הנתונים. כפי שהראינו, טיוטת תקן SQL3 מתייחסת למנגנון זה ומציעה תחביר מסוים להגדרתו. סביר מאוד להניח שעם הזמן יותר ויותר מערכות מסחריות תתמוכנה בהגדרה הדומה לתקן המוצע.

שאלות חזרה ותרגילים

שאלות חזרה

1. מהו ההבדל בין פרוצדורת בסיס נתונים (Stored Procedure) לבין מזניק (Trigger)?
2. מדוע יש קושי בניוד פרוצדורות בסיס נתונים בין מערכות RDBMS שונות?
3. הסבר את היתרון של השימוש בפרוצדורות בסיס נתונים בסביבת שרת/לקוח.
4. תן מספר דוגמאות למצבים בהם מומלץ להשתמש במנגנון המזניק (Trigger).
5. מה ההבדל בין כתיבת לוגיקה בתוכנית יישום לבין כתיבת אותה לוגיקה בפרוצדורת בסיס נתונים?

פרק 14

עיבוד תנועות בסביבת SQL (Transaction Processing)

1. מבוא
2. מהי תנועה
3. הפקודה COMMIT
4. ביצוע Commit בעבודה עם Cursor
5. הפקודה ROLLBACK
6. מודל התנועות (Transaction Model)
7. בדיקה מושהית של אילוצים
8. יומן האירועים (Log File)
9. נקודת מבדק (Checkpoint)
10. עדכון בו-זמני (Concurrent Updates)
11. מנגנון נעילות (Locking)
12. גיבוי והתאוששות (Backup and Recovery)
13. סיכום
14. שאלות חזרה ותרגילים

מבוא

בפרק זה נציג את נושא עיבוד התנועות, נושא בעל חשיבות מרובה בפיתוח יישומים בכלל ויישומים מבוססי SQL בפרט. התמיכה בעיבוד תנועות ושמירה על אמינות ושלמות בסיס הנתונים בסביבה עתירת תנועות ומרובת משתמשים, הינה אחד השירותים הבסיסיים והחשובים ביותר שמערכת RDBMS צריכה לספק. כפי שנראה, עדכונים של בסיס הנתונים אינם מתבצעים על ידי פקודות עדכון בודדות אלא על ידי רצף מסוים של פקודות. מערכת RDBMS צריכה להבטיח את הצלחת רצף פקודות העדכון ולא רק את הצלחת הפקודה הבודדת. המנגנונים שמערכת RDBMS מפעילה כדי לתמוך בסביבה של תנועות הנם מנגנונים מורכבים יחסית ודיון מלא בכל ההבטים של ניהול תנועות הוא מעבר למטרות ספר זה. יחד עם זאת, על מהנדס התוכנה להבין ולהכיר את העקרונות של מודל התנועות של מערכת RDBMS. נושא זה רלוונטי במיוחד לתוכניות יישום המשתמשות בפקודות SQL משובצות בתוך שפה מארחת, או משתמשות בממשק התכנות של מערכת RDBMS. כאשר אופן העבודה הוא אינטראקטיבי, כלומר המשתמש משגר פקודות SQL בודדות, נושא עיבוד התנועות אינו רלוונטי.

בפרק זה

- נסביר את המושג תנועה בסביבת בסיס נתונים טבלאי ונכיר את מודל התנועות.
- נראה את שתי פקודות SQL התומכות במודל עיבוד התנועות.
- נלמד על בדיקות אילוצים.
- נסקור את המנגנונים השונים שמערכת RDBMS מפעילה כדי לשמור על אמינות ושלמות בסיס הנתונים בסביבה עתירת תנועות ומשתמשים: יומן התנועות ומנגנון לנעילת הנתונים.
- נציג תהליכים לגיבוי והתאוששות מערכת.

מהי תנועה (Transaction)

מערכות המידע המודרניות, שרובן **מערכות מקוונות (On Line)**, מורכבות ממספר גדול של תוכניות יישום שכל אחת מהן מבצעת פעולות מסוימות על בסיס הנתונים. לדוגמה, מערכת מידע העוסקת במנהל אקדמי של מכללה צריכה לאפשר ביצוע פעולות כגון פתיחת מחלקה חדשה, פתיחת קורס חדש, רישום של סטודנט למכללה, רישום של סטודנט לקורס, דיווח הציון שסטודנט קיבל בקורס מסוים, בדיקת זכאות הסטודנט לתואר והדפסת תעודת זכאות, ביטול קורס, עדכון ציון של סטודנט כתוצאה מערעור, שינוי כתובת של סטודנט וכד'. פעולות אלו נקראות גם **תנועות עסקיות (Business Transactions)**. אלו פעולות שהמשתמשים מבצעים ומטרתן לשמור על עדכניות בסיס הנתונים, כך שיהיה שיקוף נאמן של המציאות. מכיון שרוב הפעולות האלו מתבצעות באמצעות מסופים או מחשבים אישיים, מקובל לקרוא גם למערכות מידע אלו **מערכות עיבוד תנועות מקוונות (On Line Transaction Processing -** מערכות **OLTP)**.

דוגמה לביצוע תנועה

לצורך הבהרת נושא עיבוד התנועות נרחיב במקצת את טבלת הקורסים ונוסיף לה שתי עמודות חדשות: מספר הסטודנטים המקסימלי שיכול להירשם לקורס ומספר הסטודנטים שכבר נרשמו לקורס. מטרת עמודות אלו היא לנהל את מצב הרישום לקורס בכל נקודת זמן.

Courses

COURSE_ID	COURSE_NAME	TYPE	POINTS	DEPARTMENT_ID	MAX_ENROLL	CURR_ENROLL
מס .קורס	שם קורס	סוג קורס	נק .זכות	קוד מחלקה	מס .נרשמים מקסימלי	מס .נרשמים נוכחי
C-200	Programming	LAB	4	CS	85	38
C-300	Pascal	LAB	4	CS	110	80
C-55	Data Base	CLASS	3	CS	45	10
M-100	Linear Algebra	CLASS	3	MT	90	88
M-200	Numeric Analysis	CLASS	3	MT	125	100
B-10	Marketing	CLASS	2	BS	40	40
B-40	Operations Res.	SEMIN	3	BS	40	36

תרשים 14.1: מבנה טבלת הקורסים לאחר הוספת שתי עמודות.

אם נתבונן לרגע על אירוע של רישום סטודנט לקורס, הרי שהוא יתבטא בעדכון שתי טבלאות בבסיס הנתונים:

- הוספת שורה חדשה לטבלת ציונים. שורה זו תציין שסטודנט נרשם לקורס מסוים בסמסטר מסוים. כמובן שבשלב הזה, כל יתר העמודות של הטבלה יכילו את הערך Null. נתונים אלה יעודכנו בשלב מאוחר יותר, לאחר שהסטודנט ייבחן בבחינות של הקורס.
- עדכון מספר הסטודנטים שנרשמו לקורס. מספר הסטודנטים שנרשמו לקורס צריך לגדול באחד, לאחר כל הרשמה.

נציג בהמשך שתי פקודות SQL שיש לבצע ברצף כדי שהאירוע יירשם בצורה נכונה בתוך בסיס הנתונים. הפקודות מציגות את הרישום של סטודנט שמספרו 210 לקורס שהקוד שלו הוא M-100 לסמסטר קיץ 1999.

```
1. INSERT INTO GRADES
2.              (COURSE_ID, STUDENT_ID, SEMESTER)
3.              ('M-100', '210' , 'SUM1999')
4.  UPDATE   STUDENTS
5.              SET       CURR_ENROLL = CURR_ENROLL + 1
6.              WHERE COURSE_ID = 'M-100' AND
7.                        STUDENT_ID = '210' AND
8.                        SEMESTER = 'SUM1999'
```

הרצף של שתי פקודות אלו חייב להתבצע כאילו הן פקודה אחת, כלומר אין להרשות מצב שבו פקודה אחת תבוצע בהצלחה ואילו השנייה תיכשל. אם למשל הפקודה הראשונה תבוצע בהצלחה ותוך כדי ביצוע הפקודה שנייה תקרה תקלה כלשהי: הפסקת חשמל, תקלה של מערכת ההפעלה שתגרום להפסקת פעולת תוכנית היישום, תקלת חומרה וכד', ייגרם נזק לבסיס הנתונים כי פעולה לוגית אחת הופסקה באמצע. בדוגמה שלנו אנו עלולים להמצא במצב בו שורה חדשה התווספה לטבלת ציונים אבל מספר הנרשמים לקורס לא עודכן. אם נספור את מספר הסטודנטים שנרשמו לקורס זה נמצא 89 ואילו בטבלת הקורסים רשום 88. זו כמובן בעיה חמורה שיכולה להביא תוך פרק זמן קצר לשיבוש מוחלט של אמינות בסיס הנתונים.

למען האמת, הבעיה במערכות RDBMS חמורה אפילו עוד יותר. כפי שראינו, פקודות העדכון של שפת SQL מסוגלות לעדכן מספר שורות רב תוך כדי ביצוע פקודה אחת. למשל, הפקודה הבאה מעדכנת בו-זמנית את כל הציונים של כל הסטודנטים שלמדו בקורס M-100 בסמסטר קיץ 1998 ומוסיפה להם חמש נקודות לציון.

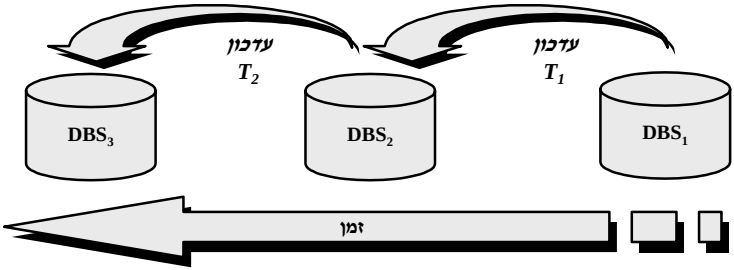
```
1. UPDATE   GRADES
2.          SET     GRADE = GRADE + 5
3.          WHERE COURSE_ID = 'M-100' AND
4.                  SEMESTER = 'SUM1998'
```

הפסקת ביצוע הפקודה באמצע, מסיבה כלשהי, עלול להביא למצב שבו חלק מהשורות בטבלת הציונים כבר עודכנו וחלק לא. לאחר חידוש פעולת בסיס הנתונים, כיצד נדע איזה שורות כבר הספיקו להתעדכן ואיזה לא. זהו שוב מצב שבו בסיס הנתונים משובש ולא אמין, למרות שביצענו פקודה אחת בלבד. בעיה זו בדרך כלל לא קיימת במערכות ניהול קבצים ואפילו במערכות DBMS וותיקות יותר, מאחר ורובן תומכות בפקודות עדכון המעדכנות רשומה אחת בלבד.

האתגר של שמירה על אמינות בסיס הנתונים בסביבה מסוג זה היא מורכבת מאוד אבל למזלנו מערכת RDBMS מבצעת משימה זו מאחורי הקלעים ובאופן כמעט שקוף מבחינתנו. נתחיל בהגדרה קצת יותר פורמלית של המונח מצב בסיס הנתונים.

מצב בסיס נתונים **Database State**	**מצב *בסיס נתונים* מוגדר כאוסף כל הערכים המופיעים בכל הטבלאות של בסיס הנתונים בנקודת זמן מסוימת.**

מצב בסיס הנתונים משקף את התוכן שלו בכל נקודת זמן. אנו נסמן את מצב בסיס הנתונים בקיצור DBS_t כאשר t מסמן את הנקודה בזמן. כל פעולת עדכון משנה את מצב בסיס הנתונים. לדוגמה הוספת שורה חדשה, עדכון עמודה בשורה כלשהי, ביטול מספר שורות: פעולות עדכון אלו משנות את מצב בסיס הנתונים.



תרשים 14.2: שינוי מצב בסיס הנתונים כתוצאה מפעולות עדכון.

תרשים 14.2 מציג כיצד בסיס הנתונים עובר ממצב DBS_1 למצב DBS_2 כתוצאה מהפעלת עדכון T_1 ולאחר מכן כיצד בסיס הנתונים עובר ממצב DBS_2 למצב DBS_3 כתוצאה מהפעלת עדכון T_2.

ניתן להבחין בין שני מצבים של בסיס הנתונים: **מצב תקין** (Consistent State) **ומצב לא תקין** (Inconsistent State). אנו נאמר שמצב בסיס נתונים הוא תקין כאשר כל הערכים המופיעים בו מייצגים באופן נאמן את מודל הנתונים על כל אילוציו. כפי שהראינו מקודם, הפסקת פעולת עדכון בודדת או רצף כלשהו של פעולות עדכון, עלול להעביר את מצב בסיס הנתונים ממצב תקין למצב לא תקין. משמעות הדבר היא שתוך כדי פעולות העדכון, מצב בסיס הנתונים לא תקין באופן זמני ורק סיום מוצלח יביאו חזרה למצב תקין.

נתבונן שוב בתרשים 14.2, ונניח ששתי פעולות העדכון, T_1 ו- T_2, שייכות לרצף לוגי אחד. פעולת העדכון הראשונה, T_1, מתחילה כאשר בסיס הנתונים הוא במצב DBS_1 שהוא מצב תקין, מעבירה אותו למצב DBS_2 שאינו מצב תקין ורק לאחר פעולה T_2 הוא מגיע למצב DBS_3 שהוא שוב מצב תקין. הבעיה של מערכת RDBMS היא כיצד להבטיח שבסיס הנתונים לא ישאר במצב DBS_2 מסיבה כלשהי. אם במצב הזה נרשה לבצע שאילתות על הטבלאות השונות, ברור שנקבל תוצאות שגויות ומבלבלות.

הגדרה ותכונות של תנועה

לפני שנסביר כיצד מערכת RDBMS יכולה להבטיח דבר זה ובאיזה מנגנונים היא משתמשת, נגדיר בצורה פורמלית יותר את המושג **תנועה**.

תנועה **Transaction**	***תנועה*** **מוגדרת כרצף פקודות SQL לעדכון בסיס הנתונים שמטרתן לבצע מטלה לוגית מסוימת. למטלה הלוגית נקרא בשם** ***יחידת עבודה לוגית*** **(Logical Unit of Work), ועליה להסתיים בהצלחה או בכישלון. במקרה של כישלון, כל הפעולות שכבר הספיקו לעדכן את בסיס הנתונים חייבות להתבטל. התנועה היא בעלת אופי של** ***הכל-או-כלום*** **(All-Or-Nothing), כלומר זוהי יחידה אטומית אחת שאסור לבצע אותה באופן חלקי.**

כל תנועה חייבת לקיים את ארבע התכונות הבאות:

- **אטומיות** (Atomicity): תכונה זו מציינת את התפישה של "הכל או לא כלום". תנועה היא יחידת עבודה לוגית שלא ניתן לפרק אותה ליחידות קטנות יותר ולכן היא חייבת להתבצע בשלמותה.
- **עקביות** (Consistency): תנועה חייבת להעביר את בסיס הנתונים ממצב תקין אחד למצב תקין אחר. רק בסיום התנועה, בסיס הנתונים חוזר למצב תקין. כפי שכבר הדגשנו, הבעיה היא שתנועה מפירה באופן זמני את תקינות בסיס הנתונים תוך כדי פעולתה. רק עם גמר התנועה, מצב בסיס הנתונים חוזר לתקינותו.
- **אי-תלות** (Independency): תנועות חייבות להתבצע באופן בלתי תלוי זו מזו. משמעות הדבר שמצב הביניים שבו נמצא בסיס הנתונים תוך כדי ביצוע התנועה חייב להיות שקוף מבחינת תנועות אחרות.
- **נצחיות** (Durability): ברגע שתנועה סיימה בהצלחה, העדכונים שבוצעו על ידה חייבים להירשם בבסיס הנתונים ואסור לאבדם כתוצאה מתקלה כלשהי.

מקובל לקרוא לתכונות אלו של התנועה בשם **ACID**, על פי ראשני התיבות של ארבע התכונות. שם זה הוגדר בשנת 1983 על ידי שני חוקרים (Haerder and Reuter).

תנועה מורכבת מרצף פעולות עדכון ולכן השאלה הראשונה שעולה היא כיצד יודעת מערכת RDBMS מתי תנועה מתחילה ומתי היא מסתיימת. הרי תוכנית היישום מורכבת ממספר רב של פקודות בשפה מארחת ומספר רב של פקודות SQL. אותה תוכנית יישום יכולה לבצע מספר רב של תנועות ולכן הבעיה הראשונה היא כיצד לסמן את גבולות התנועה בתוך תוכנית יישום. בהמשך נסביר בפירוט שתי פקודות SQL מיוחדות לטיפול בתנועות:

- **הפקודה COMMIT** מאפשרת לתוכנית היישום לסמן את סוף התנועה ולהודיע למערכת RDBMS על סיום מוצלח של התנועה.
- **הפקודה ROLLBACK** שמודיעה למערכת RDBMS על כישלון התנועה. במצב זה המערכת מבטלת את כל הפעולות שבוצעו עד כה במסגרת התנועה.

שאלה מעניינת היא כיצד מערכת RDBMS יודעת היכן מתחילה התנועה. כפי שנסביר בהמשך בסעיף מודל התנועות, התחלת התנועה יכולה להיות מפורשת או לא-מפורשת.

מנקודת מבטו של מהנדס התוכנה, ניתן לומר שמודל התנועות מתבטא רק בשתי פקודות אלו. כפי שנראה בהמשך, מנקודת מבט מערכת RDBMS שתי פקודות אלו "מסתירות" מנגנון מורכב מאוד. במילים אחרות, שתי פקודות אלו הן רק קצה הקרחון.

שתי הפקודות אינן משנות את מצב בסיס הנתונים, כמו שעושות זאת הפקודות INSERT או UPDATE. פקודות אלו הן למעשה הוראות למערכת RDBMS ולמעשה – לרכיב **מנהל התנועות** (Transaction Manager, או Transaction Monitor) שלה.

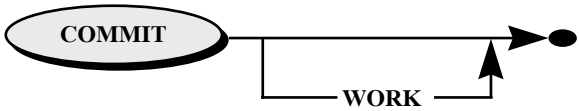
מערכת RDBMS יכולה לפעול באחד משני האופנים האלה:

- **באופן עצמאי** – כאשר בסיס הנתונים פועל באופן עצמאי, ומערכת RDBMS מקבלת באופן ישיר את ההוראות COMMIT ו-ROLLBACK מתוכניות היישום.
- **תחת מנהל תנועות חיצוני** – בסיס הנתונים יכול לפעול גם תחת מנהל תנועות חיצוני. לדוגמה, בסביבת מחשבי יבמ מנהל התנועות יכול להיות מערכת CICS ובסיס הנתונים יכול להיות DB2; בסביבת Unix, מנהל התנועות יכול להיות Tuxedo ובסיס הנתונים יכול להיות Oracle. במקרים אלה הפקודה Commit המתבצעת על ידי תוכנית היישום הינה הוראה למנהל התנועות והוא זה שמודיע למערכת RDBMS לבצע Commit. צורת עבודה זו מאפשרת למנהל תנועות אחד לפקח על תנועות המעדכנות אפילו מספר מערכות RDBMS של יצרנים שונים. נושא זה יוסבר בפרק 15, המציג את העבודה בסביבה מבוזרת.

הפקודה COMMIT

הפקודה Commit מאפשרת לתוכנית היישום להודיע למערכת RDBMS שהתנועה הסתיימה בהצלחה. כלומר, היא מודיעה שכל פקודות העדכון שהיו צריכות להתבצע כחלק מהתנועה בוצעו ומצב בסיס הנתונים תקין. הצורך בפקודה זו ברור: אין למערכת RDBMS כל דרך לדעת מתי תנועה מסתיימת. תנועה אחת יכולה להיות מורכבת מארבע פקודות עדכון, ותנועה אחרת – מעשרים פקודות עדכון. תוכנית יישום אחת יכולה לבצע תנועה אחת או מספר תנועות. רק מי שכותב את היישום יודע מתי תנועה מסתיימת ולכן יש צורך בפקודה מפורשת להודעה על סיום מוצלח.

תחביר הפקודה:



תרשים 14.3: תרשים תחביר של הפקודה COMMIT.

דוגמה: נראה את רצף פעולות העדכון לרישום סטודנט לקורס יחד עם הפקודה COMMIT, המופיעה בשורה 9, ומודיעה על סיום מוצלח של התנועה.

```
1. INSERT INTO GRADES
2.              (COURSE_ID, STUDENT_ID, SEMESTER)
3.              ('M-100', '210' , 'SUM1999')
4.  UPDATE   STUDENTS
5.              SET       CURR_ENROLL = CURR_ENROLL + 1
6.              WHERE COURSE_ID = 'M-100' AND
7.                        STUDENT_ID = '210' AND
8.                        SEMESTER = 'SUM1999'
9.  COMMIT
```

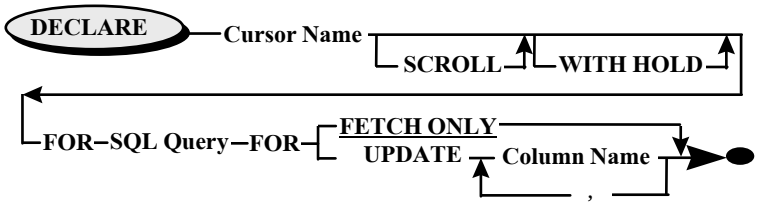
ביצוע Commit בעבודה עם Cursor

בפרק 12, שעסק בתכנות יישומים המשתמשים בפקודות SQL לגישה לבסיס נתונים טבלאי, הצגנו את המנגנון של **הסמן**, Cursor, המגשר בין גישת המודל הטבלאי המטפל בבת אחת במספר רב של שורות לבין שפת תכנות הפועלת באופן סדרתי ומסוגלת לעבד בכל פעולה שורה אחת בלבד. נתייחס לסוגיה של ביצוע COMMIT בעת עבודה עם Cursor. חשוב להדגיש בנקודה זו שביצוע COMMIT מסמן את הסיום המוצלח של התנועה וגורם, בין השאר, לסגירת ה-Cursor. אם תוכנית היישום סורקת את שורות ה-Cursor (כזכור, Cursor הוא טבלה) ובמהלך הסריקה היא תבצע COMMIT, ה-Cursor ייסגר ובאופן טבעי התוכנית תאבד את המיקום היחסי שלה בתוך הסמן. סגירת הסמן יוצרת בעיה מיוחדת במצב שבו תוכנית היישום סורקת את שורות הסמן ומבצעת תנועות לעדכון בסיס הנתונים במהלך הסריקה.

יש מצבים שבהם תוכנית יישום מבצעת מספר רב של עדכונים ורוצה לבצע COMMIT לאחר מספר מוגדר של עדכונים, כדי להבטיח שבמקרה של תקלה לא יהיה צורך לבצע גלילה לאחור (Rollback) של כל העדכונים שבוצעו מתחילת התוכנית. אם התוכנית תבצע Commit, הסמן ייסגר והיא תאבד את המיקום בסמן, ולא תוכל להמשיך אחר כך אם תהיה תקלה. להדגמה נניח שתוכנית שלפה בפקודה SELECT טבלה בת 100,000 שורות, ובמהלך סריקת הסמן היא רוצה לעדכן אותן. כדי שלא יהיה צורך לחזור על כל 100,000 העדכונים במקרה של תקלה, התוכנית רוצה לבצע Commit לאחר כל 5,000 עדכונים.

כדי להתמודד עם מצב מיוחד זה, מערכות RDBMS תומכות בפרמטר WITH HOLD שיש להצהיר עליו בעת הגדרת הסמן (Cursor).

תחביר הפקודה:



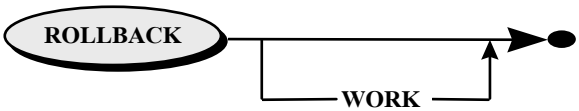
תרשים 14.4: הפקודה DECLARE CURSOR עם הפרמטר WITH HOLD.

שימוש בפרמטר זה בעת ההצהרה על Cursor תגרום לכך שביצוע COMMIT לא יסגור את הסמן, ותוכנית היישום לא תאבד את מיקומה היחסי. לאחר ביצוע COMMIT ניתן לבצע FETCH לשליפת שורה נוספת, או – CLOSE לסגירת הסמן. אין לבצע את הפקודות UPDATE CURRENT OF או DELETE CURRENT OF מייד לאחר COMMIT.

הפקודה ROLLBACK

שפת SQL כוללת את הפקודה ROLLBACK, **גלילה לאחור**, המאפשרת לתוכנית היישום לבקש מבסיס הנתונים לבטל את כל העדכונים שבוצעו מתחילת התנועה. תוכנית היישום יודעת בכל נקודת זמן את מצב התנועה, ולכן היא יכולה לאתר מצב שגוי גם לאחר שחלק מהעדכונים בוצעו ולבקש את ביטולם על ידי הפקודה ROLLBACK.

תחביר הפקודה:



תרשים 14.5: תרשים תחביר לפקודה ROLLBACK.

דוגמה : נניח שיש לבצע רצף של שלוש פקודות לעדכון ציונים של סטודנט שמספרו 210. ההנחה היא שסטודנט זה נבחן בשלושה קורסים וועדת ההוראה קיבלה החלטה לתקן את ציוניו בקורסים אלה. נניח שהפקודה הראשונה והשנייה בוצעו בהצלחה ואילו הפקודה השלישית מחזירה קוד שגיאה שהשורה לא נמצאה. זהו מצב לא תקין ולכן יש צורך לבטל גם את שני העדכונים הקודמים. הדרך לעשות זאת היא על ידי ביצוע הבדיקה של קוד ההחזר, SQLCODE, לאחר כל פעולת עדכון. אם הערך המתקבל לאחר ביצוע העדכון אינו אפס, יש לבצע גלילה לאחור של כל העדכונים שכבר בוצעו.

```
.
EXEC SQL UPDATE GRADES SET GRADE = 95
    WHERE COURSE_ID = 'M-100' AND
            STUDENT_ID = '210' AND SEMESTER = 'AUT1999';
IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
EXEC SQL UPDATE GRADES SET GRADE = 87
    WHERE COURSE_ID = 'C-200' AND
             STUDENT_ID = '210' AND SEMESTER = 'AUT1999';
IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
EXEC SQL UPDATE GRADES SET GRADE = 68
    WHERE COURSE_ID = 'B-10' AND
            STUDENT_ID = '210' AND SEMESTER = 'WIN1999';
IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
EXEC SQL COMMIT;
  .
  .
ERROR_ROUTINE:
   EXEC SQL ROLLBACK;
   PRINTF ("SQL ERROR: %LD/N, SQLCA.SQLCODE);
   EXIT ();
  .
```

כפי שניתן לראות, התוכנית מבצעת בדיקה של קוד ההחזר SQLCODE לאחר כל פקודה. אם הוא קטן מאפס, מתבצע דילוג לשיגרה ERROR_ROUTINE שתפקידה לטפל במצב חריג זה. דבר ראשון שהשיגרה מבצעת הוא ביטול העדכונים ולאחר מכן היא מדפיסה את קוד השגיאה ומפסיקה את התוכנית. אם כל פקודות העדכון מסתיימות בהצלחה, תתבצע הפקודה COMMIT לסימון סוף תנועה תקין.

מודל התנועות (Transaction Model)

מודל התנועות מגדיר את האופן שבו מערכת RDBMS מזהה את תחילת התנועה, את סיומה המוצלח או את כישלון ביצוע התנועה. תקן SQL מגדיר את מודל התנועות בצורה הבאה:

- **תחילת תנועה:** כל תנועה מתחילה עם פקודת העדכון הראשונה בתוכנית או פקודת העדכון הראשונה לאחר הפקודה COMMIT. נשים לב שאין פקודה מפורשת המציינת את תחילת התנועה אלא המערכת מניחה את התחלתה. משמעות הדבר היא שמערכת RDBMS מניחה שתוכנית היישום נמצאת תמיד במצב של תנועה ואין כל דרך לסמן לה באופן מפורש מצב זה.
- **סיום תנועה תקין:** תנועה מסתיימת בהצלחה על ידי ביצוע הפקודה COMMIT, או אם תוכנית היישום מסתיימת בהצלחה. נשים לב שהמודל מניח שתנועה הסתיימה בהצלחה אם תוכנית יישום מסתיימת בהצלחה למרות שלא ביצעה במפורש COMMIT.
- **סיום תנועה לא תקין:** תנועה מסתיימת באופן לא תקין על ידי ביצוע הפקודה ROLLBACK, או אם תוכנית היישום אינה מסתיימת בהצלחה, כלומר "עפה", כמו שמקובל לומר בסלנג של אנשי המחשב. משמעות הדבר שאם תוכנית יישום מסיימת באופן לא תקין כתוצאה מתקלה כלשהי, מערכת RDBMS מפעילה באופן אוטומטי את הפקודה ROLLBACK.

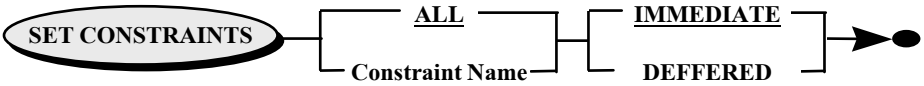
לא כל המערכות המסחריות תומכות במודל תנועות זה, ומתבססות על מודל תנועות שונה. לדוגמה, מערכת Sybase מספקת פקודה בשם BEGIN TRANSACTION שמסמנת בצורה מפורשת את תחילת התנועה, ופקודה COMMIT TRANSACTION שמסמנת את סיום התנועה. ביצוע COMMIT TRANSACTION ללא BEGIN TRANSACTION לפנייה יגרום להודעת שגיאה. בנוסף, המערכת מספקת פקודה שמאפשרת לשמור את מצב התנועה בכל נקודה: SAVE TRANSACTION SP. פקודה זו בונה נקודת ייחוס מיוחדת ששמה הלוגי הוא SP, וניתן לחזור אל מצב בסיס הנתונים כפי שהוא היה בנקודה זו. כדי לחזור לנקודה זו קיימת הפקודה ROLLBACK savepoint המאפשרת להחזיר את מצב בסיס הנתונים לנקודה זו, ולאו דוקא לתחילת התנועה. שיטה זו טובה במיוחד עבור תנועות ארוכות מאוד שחזרה לתחילתן דורשת משאבי מחשב רבים.

בדיקה מושהית של אילוצים

במצבים מסוימים איננו רוצים שמערכת RDBMS תבדוק אילוצים שונים מייד לאחר ביצוע כל פקודת SQL, אלא רק בעת סיום מוצלח של התנועה, כלומר בעת ביצוע COMMIT. נזכיר ש-SQL מאפשרת להגדיר אילוצים בעת הגדרת טבלה ומחייבת שם חד- ערכי לכל אילוץ. לדוגמה, בפרק 9 שעסק בהגדרת נתונים הוצגה דוגמה להגדרת טבלה עם האילוץ Max_Number_Of_Exams המוודא שסטודנט לא נבחן מעל מספר נתון של בחינות.

כל בדיקת אילוץ צורכת משאבי מחשב ולכן משפיעה על זמן התגובה של התנועה. אם תוך כדי ביצוע התנועה נבצע את בדיקות האילוצים, אולם לבסוף התוכנית תחליט לא לבצע Commit אלא Rollback, בזבזנו משאבי מחשב יקרים. שפת SQL מכילה פקודה מיוחדת המאפשרת לקבוע האם בדיקת האילוץ תתבצע מייד לאחר כל פקודת עדכון של טבלה או להשהות את בדיקת האילוץ רק לאחר סיום מוצלח של התנועה, כלומר רק אם התוכנית ביצעה Commit.

תחביר הפקודה:



תרשים 14.6: תרשים תחביר לפקודה SET CONSTRAINTS.

פקודה זו מאפשרת לקבוע את שיטת בדיקת האילוצים בכל תנועה בנפרד. ברירת המחדל היא לבדוק את כל האילוצים המוגדרים בטבלה עם כל עדכון. אם אחד האילוצים מופר, העדכון לא יבוצע. ניתן לבקש בדיקה מושהית של כל האילוצים או של אילוץ מסוים על ידי ציון שם האילוץ. כמובן שאם בדיקת האילוץ תיכשל, מערכת RDBMS לא תסכים לבצע את התנועה והיא תבוטל.

דוגמה : יש להגדיר שבדיקת אילוץ המספר המקסימלי של בחינות לסטודנט שהוגדרה עבור טבלת הציונים, תושהה עד לסיום מוצלח של התנועה.

SET CONSTRAINTS MAX_NUMBER_OF_EXAMS DEFFERED;

יומן אירועים (Log File)

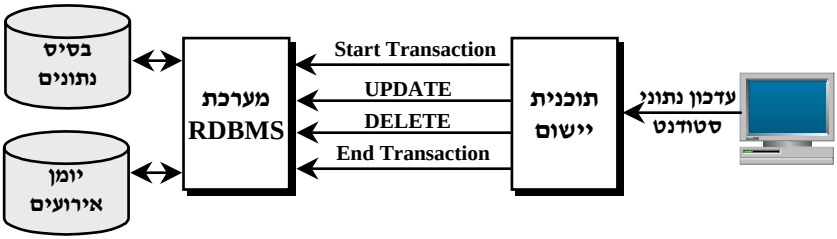
המפתח לביצוע תהליך ההתאוששות כתוצאה מתקלה והמנגנון המאפשר להפעיל את הפקודה ROLLBACK הינו **קובץ יומן האירועים**.

יומן אירועים **Log File**	***יומן אירועים*, הנקרא לפעמים גם בשם Journal, הינו קובץ מיוחד המנוהל על ידי מערכת RDBMS ומכיל רשומה אחת עבור כל עדכון שבוצע בבסיס הנתונים, וכך הוא מאפשר התאוששות מתקלות ושחזור המצב המקורי לפני העדכון.**

יומן האירועים מכיל את כל השינויים שבוצעו בבסיס הנתונים וגם נתונים נוספים הדרושים לשחזור המצב הקודם. עבור כל עדכון המתבצע בבסיס הנתונים, מערכת RDBMS רושמת רשומה אחת ביומן האירועים המכילה בדרך כלל את הנתונים האלה:

- שם התנועה, ולמעשה שם תוכנית היישום.
- הזמן בו בוצעה התנועה.
- התאריך בו בוצעה התנועה.
- זיהוי המשתמש ותחנת העבודה שממנה בוצעה התנועה.
- הפעולה שבוצעה (ביטול רשומה, עדכון רשומה, הוספת רשומה חדשה, תחילת תנועה, סוף תנועה).
- שם הטבלה שבה בוצעה הפעולה.
- תוכן השורה לפני העדכון (Before Image).
- תוכן השורה לאחר העדכון (After Image).

כל תנועה רושמת את כל פעולותיה ליומן האירועים, בנוסף לרישום הנעשה בבסיס הנתונים עצמו, ולכן, ניתן להשתמש בקובץ מיוחד זה למטרות התאוששות במקרה של תקלה. לדוגמה, אם תוכנית יישום מסיימת את פעולתה באופן לא תקין, כלומר לא מבצעת COMMIT, תתחיל מערכת RDBMS לבצע את תהליך Rollback. המערכת תקרא את קובץ יומן האירועים בסדר כרונולוגי הפוך, ותחזיר את בסיס הנתונים למצבו לפני העדכון. אם נרשמה שורה חדשה, היא תבוטל. אם שורה בוטלה, היא תוחזר לבסיס הנתונים. אם אחת העמודות של שורה עודכנה, תוחזר לבסיס הנתונים השורה עם התוכן כפי שהוא היה לפני העדכון.



תרשים 14.7: יצירת יומן אירועים על ידי מערכת RDBMS.

כפי שניתן לראות מתרשים זה, המערכת מנהלת שני מאגרים: מאגר אחד המכיל את הטבלאות של בסיס הנתונים ומאגר נפרד המכיל את יומן האירועים. קובץ זה יכיל רשומה אחת עבור כל אירוע של עדכון והוא משמש לצרכי התאוששות מתקלות.

יומן האירועים משמש את מערכת RDBMS גם להתאוששות במקרה של נפילת מערכת. מייד עם חזרת המחשב לפעולה, המערכת מפעילה תוכנית שירות מיוחדת הסורקת את יומן האירועים ומחפשת את כל התנועות שלא הספיקו לבצע COMMIT. לכל תנועה כזאת מתבצע תהליך ביטול ושחזור בסיס הנתונים למצבו לפני תחילת התנועה. מקובל לקרוא לתהליך זה **שחזור לאחור** (Backward Recovery). יומן האירועים יכול לשמש גם לשחזור

מהיר של בסיס הנתונים במקרה של תקלה. ניתן לקחת את קובץ הגיבוי האחרון של בסיס הנתונים, להחזירו לדיסק ולהפעיל עליו את כל השורות שאחרי העדכון (After Image) לפי הסדר הכרונולוגי שבו הן בוצעו. בגמר התהליך בסיס הנתונים חוזר למצב שלפני התקלה. מקובל לקרוא לתהליך זה גם בשם **שחזור לפנים** (Forward Recovery).

תרשים 14.8 מציג דוגמה של תוכן קובץ יומן אירועים.

Tran. Id.	Date	Time	User Id.	Terminal Id.	Action Type	Table	Before Values	After Values
Grd-01	1999/08/12	07:30:35	Dan	Ter-05	Start			
Grd-01	1999/08/12	07:31:01	Dan	Ter-05	Update	Grades	.....	.....
Dpt-08	1999/08/12	07:31:28	Ron	Ter-08	Start			
Grd-01	1999/08/12	07:32:02	Dan	Ter-05	Insert	Grades	.....	.....
Dpt-08	1999/08/12	07:32:25	Ron	Ter-08	Delete	Departments	.....	.....
Grd-01	1999/08/12	07:32:54	Dan	Ter-05	Commit			
Dpt-07	1999/08/12	07:33:10	Eyal	Ter-10	Start			
Dpt-08	1999/08/12	07:33:18	Ron	Ter-08	Commit			
Dpt-07	1999/08/12	07:33:34	Eyal	Ter-10	Insert	Departments	.....	.....

תרשים 14.8: דוגמה של קטע מקובץ יומן אירועים.

מהתבוננות ביומן אירועים זה נוכל להסיק את העובדות האלו:

- המשתמש Dan הפעיל את התנועה Grd-01 בשעה 7:30:35, סיים אותה בהצלחה בשעה 7:32:54 לאחר שעדכן שורה והוסיף שורה לטבלת הציונים.
- המשתמש Ron הפעיל את התנועה Dpt-08 בשעה 7:31:28 וסיים אותה בהצלחה בשעה 7:33:18, לאחר שביטל שורה בטבלת המחלקות.
- המשתמש Eyal הפעיל את התנועה Dpt-07 בשעה 7:33:10 והספיק להוסיף שורה לטבלת המחלקות. התנועה לא הספיקה להסתיים מכיון שלא נרשמה רשומת Commit ביומן האירועים. כדי לבצע Rollback נצטרך לבטל את השורה מטבלת המחלקות שהמפתח שלה רשום ביומן האירועים כחלק מהעמודה Before Values.

כל פעולה שנעשית בבסיס הנתונים נרשמת גם ביומן האירועים, ולכן קובץ זה יכול לשמש גם כנתיב ביקורת (Audit Trail) כדי לבדוק איזה תוכניות ואיזה משתמשים עשו מה ומתי בבסיס הנתונים. נפח יומן האירועים יכול לגדול מהר מאוד, במיוחד בסביבה רבת משתמשים, ועל כן מקובל לגבות אותו (לקלטת או סרט מגנטי למשל) מדי פעם, ולרוקן את השטח שהוקצה לו בדיסק (במידה ויש צורך בכך). יש לזכור, שהפעלת יומן האירועים משפיעה על ביצועי מערכת המחשב, מאחר וכל פעולת עדכון בבסיס הנתונים גורמת לפעולת עדכון מקבילה של יומן האירועים. תפקידו של מנהל בסיס הנתונים הוא לקבוע מה יירשם ביומן האירועים (למשל מצב "לפני" בלבד, או גם "לפני" וגם "אחרי").

לרוב, מערכות RDBMS מעדכנות תחילה את יומן האירועים ורק לאחר מכן – את בסיס הנתונים. שיטה זו מכוונת למנוע מצב שבו נרשם עדכון בבסיס הנתונים ותקלת חומרה כלשהי מנעה את עדכון יומן האירועים. לשיטה זו, שבה מעדכנים תחילה את יומן האירועים, מקובל לקרוא בשם **פרוטוקול רישום מראש** (Write-Ahead Log Protocol). שיטה זו מבטיחה שבכל מצב, קודם מתעדכן יומן האירועים ורק לאחר מכן בסיס הנתונים עצמו. במקרה שעדכון יומן האירועים בוצע וארעה תקלה שמנעה את עדכון בסיס הנתונים עצמו, יש למערכת כלים לזהות מצב זה בזמן הפעלת תוכנית ההתאוששות ולהבטיח את כתיבת העדכונים לבסיס הנתונים לפני החזרת בסיס הנתונים לעבודה.

נקודת מבדק (Checkpoint)

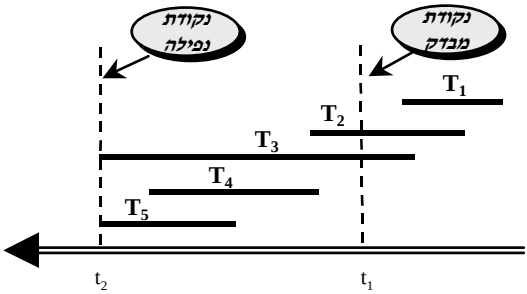
כאשר מבצעים תהליך של גלילה לאחור של בסיס הנתונים (Rollback), נשאלת השאלה "עד היכן לחזור" במהלך שחזור המצב לקדמותו. ייתכן שתנועה מסוימת החלה לפני זמן רב ועדיין לא סיימה, ולכן עקרונית צריך לחזור עד לתחילת יומן האירועים. תהליך כזה הוא מאוד לא יעיל וצורך זמן רב. כדי להתגבר על בעיה זו פותחה שיטה היוצרת מעת לעת נקודה בזמן, שבה בודאות קיימת התאמה מלאה בין מצב בסיס הנתונים לבין יומן האירועים. המניחים שכל התנועות שלא היו פעילות בנקודת זמן זו בודאי ביצעו Commit ולכן אין צורך לחזור אליהן.

נקודת מבדק **Checkpoint**	***נקודת מבדק*** **מוגדרת כנקודת ציון במהלך ביצוע התוכנית שבה מובטח הסנכרון בין בסיס הנתונים לבין יומן האירועים. כדי לבצע נקודת מבדק, מערכת RDBMS מפסיקה לקבל תנועות חדשות, כותבת את תוכן כל המאגרים לבסיס הנתונים וליומן האירועים, רושמת רשומה מיוחדת ביומן האירועים המכילה את זיהוי כל התנועות שהיו פעילות באותה עת ומחדשת את הפעילות.**

רוב מערכות RDBMS מכילות מנגנון אוטומטי של **נקודת מבדק** (Checkpoint) הגורם לכך שמזמן לזמן (לדוגמה, כל רבע שעה) המערכת דואגת לסנכרון מלא בין מצב בסיס הנתונים לבין מצב יומן האירועים. הפעולות שהמערכת מבצעת לסנכרון בין בסיס הנתונים לבין יומן האירועים:

- המערכת כותבת רשומה מיוחדת ליומן האירועים, רשומת Checkpoint. רשומה זו מציינת מתי החל תהליך הסנכרון.
- המערכת עוצרת את ביצוע התנועות הפעילות באותו זמן, ומסרבת לקבל תנועות חדשות.
- המערכת כופה על מערכת ההפעלה לכתוב (Force Write) את כל המאגרים שעדיין לא נכתבו לדיסק.
- המערכת רושמת ביומן האירועים ברשומה מיוחדת, Checkpoint Record, את הזיהוי של כל התנועות שהיו פעילות בזמן נקודת המבדק.
- המערכת מחדשת את פעילות בסיס הנתונים ומקבלת תנועות חדשות.

נקודת המבדק היא מושג חשוב בתהליך השחזור, כי זוהי נקודת זמן שבה מובטח קיום סינכרון מלא בין מצב בסיס הנתונים לבין יומן האירועים. נשתמש בדוגמה כדי להדגים את רעיון נקודת המבדק. תרשים 14.9 מדגים את מצב המערכת והתנועות לאורך ציר הזמן. בנקודת זמן t_1 נרשמה נקודת מבדק ובנקודת זמן t_2 היתה נפילה של המערכת.



תרשים 14.9: מצב מערכת בין נקודת מבדק לנקודת נפילה.

מתוך התרשים ניתן להבין את הדברים הבאים:

- התנועה T_1 הסתיימה בהצלחה לפני נקודת המבדק.
- התנועות T_2 ו- T_3 היו פעילות בזמן נקודת המבדק.
- התנועה T_3 היתה פעילה בזמן הנפילה.
- התנועה T_4 החלה אחרי נקודת המבדק והסתיימה בהצלחה לפני הנפילה.
- התנועה T_5 החלה אחרי נקודת המבדק והיתה פעילה בזמן הנפילה.

תרשים 14.10 מציג את תוכן יומן האירועים המשקף את האירועים המתוארים בתרשים הקודם. נשים לב לרשומה חדשה ביומן האירועים, רשומת נקודת המבדק, רשומה שנרשמה בשעה 07:32:02. רשומה זו כוללת את רשימת התנועות שהיו פעילות בזמן נקודת המבדק, תנועות T_2 ו- T_3 בדוגמה שלנו.

Tran. Id.	Date	Time	User Id.	Terminal Id.	Action Type	Table	Before Values	After Values
T_1	1999/08/12	07:30:35	Dan	Ter-05	Start			
T_1	1999/08/12	07:31:01	Dan	Ter-05	Commit			
T_2	1999/08/12	07:31:14	Ron	Ter-08	Start			
T_3	1999/08/12	07:31:28	Ziv	Ter-12	Start			
Check	1999/08/12	07:32:02				T_2, T_3		
T_2	1999/08/12	07:32:25	Ron	Ter-08	Delete	Departments	.....	.....
T_3	1999/08/12	07:32:54	Ziv	Ter-12	Update	Grades	.....	.....
T_2	1999/08/12	07:33:02	Ron	Ter-08	Commit			
T_4	1999/08/12	07:33:10	Eyal	Ter-10	Start			
T_4	1999/08/12	07:33:18	Eyal	Ter-10	Commit			
T_5	1999/08/12	07:33:34	Dan	Ter-05	Start			

תרשים 14.10: מצב יומן האירועים לאחר נקודת מבדק.

לאחר שתנועה T_5 החלה לעבוד, ארעה תקלה ומערכת RDBMS הפסיקה לפעול. מייד עם חידוש פעילות בסיס הנתונים, מתבצעת פעולה של שחזור בסיס הנתונים למצב תקין. מערכת RDBMS מכילה תוכנית שרות מיוחדת לביצוע פעולת ההתאוששות (Recovery Utility) מהתקלה. תוכנית שרות זו תחפש בתוך יומן האירועים את רשומת נקודת המבדק האחרונה שנרשמה. עכשיו תוכנית השרות יודעת שבזמן נקודת המבדק רק התנועות T_2 ו- T_3 היו פעילות. מכאן, תתחיל תוכנית השרות להתקדם קדימה ביומן האירועים. התנועות T_2 ו- T_4 הספיקו לבצע Commit לאחר נקודת המבדק ולכן יש צורך לבצע Rollback רק לתנועות T_3 ו- T_5. תנועות T_2 ו- T_4 ביצעו Commit לפני הנפילה סיימו בהצלחה, ולכן אין צורך לבטל אותן.

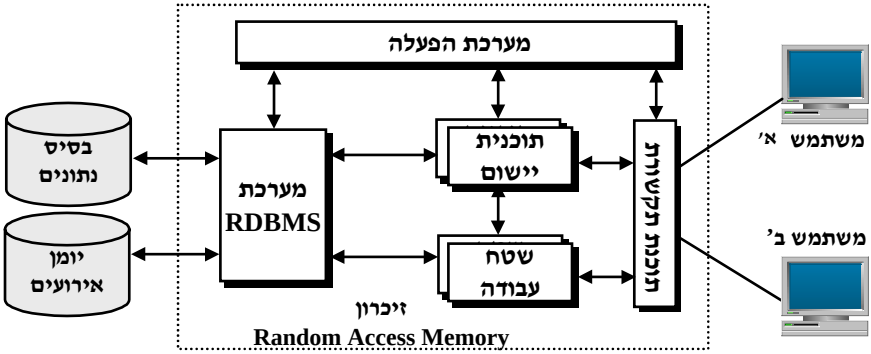
אך יש לזכור שייתכן מצב, שבו התנועות אכן סיימו בהצלחה אולם בזמן הנפילה המערכת עדיין לא הספיקה לעדכן את בסיס הנתונים מתוך מאגרי הקלט/פלט בזיכרון. כפי שהוסבר, פרוטוקול Write-Ahead Log מבוסס על העקרון שהכתיבה מתבצעת קודם ביומן האירועים ורק לאחר מכן בבסיס הנתונים. לכן ייתכן שבסיס הנתונים במצב פגום, למרות שיומן האירועים לא מראה זאת. אם רוצים להבטיח אמינות מלאה, יש לבצע Rollback גם לתנועות T_2 ו- T_4. התנועה T_1 לא תבוטל, מאחר והיא סיימה לפני נקודת המבדק, ולכן היא עדכנה בודאות הן את בסיס הנתונים והן את יומן האירועים.

עדכון בו-זמני (Concurrent Updates)

עד עתה הנחנו, שאם תוכנית היישום פועלת כשורה, בסיס הנתונים יכיל נתונים אמינים ונכונים, כלומר הוא יהיה במצב תקין. הנחה זו אינה מדויקת, מכיון שייתכנו מצבים שבהם תוכניות היישום פועלות כשורה, מנגנון ניהול התנועות פועל כשורה ובכל זאת בסיס הנתונים יהיה במצב לא תקין. בעיה זו מתעוררת בעקבות עדכון בו-זמני, כלומר עבודה במקביל של מספר משתמשים על אותה קבוצה של נתונים.

עדכון בו-זמני **Concurrent Update**	***עדכון בו-זמני*** **מוגדר כמצב, שבו שני משתמשים או יותר מנסים לעדכן בסיס נתונים אחד באותו זמן.**

לפני שנסביר איזה בעיות יכולות להתעורר כתוצאה ממצב של עדכון בו-זמני, חשוב להבין כיצד מערכת ההפעלה של המחשב תומכת בעבודה **בסביבה מרובת משתמשים** (Multi User Environment). שיטת העבודה הרגילה היא שכל משתמש אשר מתחבר ליישום כלשהו, כלומר מבצע תהליך של הזדהות – Login, ומבקש להפעיל את היישום, מקבל עותק משלו של תוכנית היישום בזיכרון. אם נזכר לרגע במודל העבודה של מערכת RDBMS שהוצג בפרק 2, הרי שכל תוכנית יישום תופסת שני שטחי זיכרון – שטח אחד עבור הפקודות של התוכנית ושטח שני הוא שטח העבודה שבתוכו התוכנית שומרת את המשתנים ואת הנתונים שהיא מעבדת.



תרשים 14.11: סביבת העבודה של תוכנית יישום הפועלת בסביבת תקשורת.

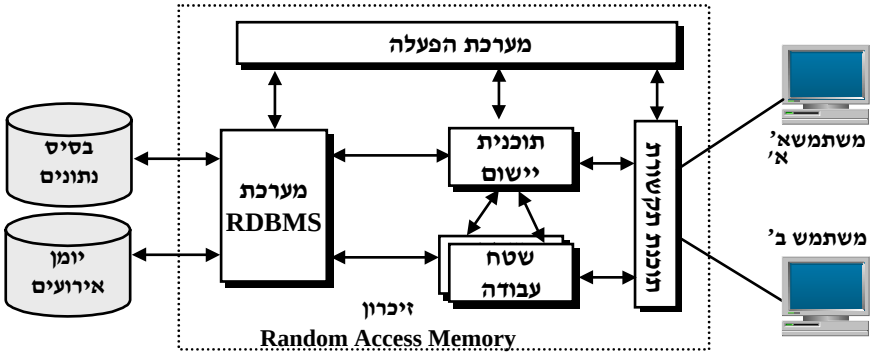
כפי ניתן לראות מתרשים 14.11, מערכת ההפעלה מפקחת על כל הפעולות המתרחשות במחשב ומנהלת את כל המשאבים (זיכרון, CPU, דיסקים וכד'). תוכנת התקשורת מנהלת את כל התקשורת עם תחנות העבודה המרוחקות. זהו מודל העבודה במערכת הפועלת בצורה מרכזית. בפרק 15 העוסק בבסיסי נתונים מבוזרים נוכל לראות שקיימים מודלים נוספים וביניהם מודל שרת/לקוח שבו חלק מהלוגיקה יכולה להתבצע גם בתחנת העבודה. בכל אופן, לצורך הבנת שיטת העבודה בסביבה מרובת משתמשים, מודל העבודה של מערכת מרכזית או מערכת שרת/לקוח צריך להתמודד עם אותן בעיות.

עכשיו נראה מה קורה כאשר שני משתמשים שונים מבצעים Login ומבקשים להפעיל את אותה תוכנית יישום. במצב זה, שני עותקים של אותה תוכנית היישום יימצאו בזיכרון, כל תוכנית עם שטח העבודה הפרטי שלה. למרות שהלוגיקה של תוכנית היישום היא זהה, כל משתמש יכול לשלוף נתונים אחרים מבסיס הנתונים, להזין נתונים שונים ולכן המשתנים בשטח העבודה של כל עותק יהיו שונים.

מערכות ההפעלה המודרניות פועלות בשיטה של Multitasking, שיטה שבה מספר משימות מתבצעות במקביל על אותו מעבד. המערכת מקצה את המעבד לכל משימה למשך זמן מסוים, מפסיקה את ביצוע המשימה ומקצה את המעבד למשימה אחרת וכך חזור חלילה בין כל המשימות. מערכת ההפעלה מנהלת לכל משימה **מראה מקום** מיוחד המצביע על הפקודה האחרונה שבוצעה, כך שעם קבלת המעבד חזרה, המשימה יכולה להמשיך לפעול בדיוק מהפקודה הבאה שהיא צריכה לבצע. בגלל מהירותו הגבוהה של המעבד, למשתמשים השונים נוצרת התחושה שתוכנית היישום שלהם עובדת כל הזמן, אולם זה איננו המצב. כל פעם תוכנית יישום אחרת פועלת, מתקדמת עד לנקודה מסוימת, מופסקת והפיקוח מועבר לתוכנית השנייה. בצורה זו, יכול מחשב אחד לשרת מאות ואלפי משתמשים שונים בו-זמנית.

כפי שהקורא בוודאי שם לב, יש כאן בזבוז של זיכרון בשמירת אותה תוכנית יישום מאות ואולי אלפי פעמים בזיכרון, עותק עבור כל משתמש. מערכות ההפעלה המודרניות תומכות בשיטת עבודה הקרויה Reentrant (רב כניסות) שבה נשמר רק עותק אחד של התוכנית שתוכניות אחדות יכולות להשתמש בו בו-זמנית. לכל משתמש נוסף באותה תוכנית מקצה

מערכת ההפעלה רק את שטחי העבודה. המערכת יודעת לנהל מראה מקום נפרד עבור כל משתמש, כך שאותה תוכנית יכולה לבצע פקודה שונה עבור כל משתמש.



תרשים 14.12: סביבת עבודה של תוכנית יישום הפועלת בשיטת Reentrant.

עכשיו לאחר שהבנו שלכל משתמש שמפעיל תוכנית יישום יש עותק משלו של שטח העבודה, נוכל לנתח מצב שבו שני סטודנטים שונים, סטודנט א' וסטודנט ב', מנסים להירשם באותו זמן לאותו הקורס. נניח שברחבי הקמפוס במכללה מוצבות תחנות עבודה המאפשרות לכל סטודנט לבצע פעולות שונות: בירורים של קורסים, מצב הציונים שלו וכן להירשם לקורסים. נתבונן בעיקר במה שקורה עם טבלת הקורסים המתוארת בתרשים 14.1, שהוספנו לה את שתי העמודות החדשות: מספר סטודנטים מקסימלי לקורס ומספר הסטודנטים שכבר נרשמו לקורס.

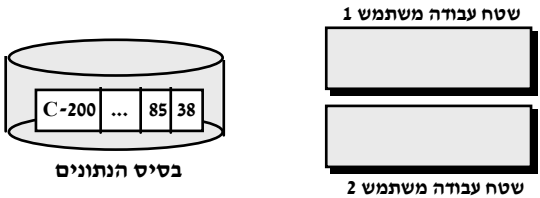
מצב של עדכון בו-זמני יכול לגרום לבעיות שונות, שמקובל לחלק אותן לשלוש קטגוריות:

- בעיית העדכון האבוד.
- בעיית עדכון רשומה הנמצאת בתהליך עדכון.
- בעיית ניתוח נתונים לא עקבי.

נפרט את המצבים הבעייתיים האלה באמצעות דוגמאות. כפי שנראה בהמשך, הפתרון לבעיות אלו הוא שימוש ב**מנגנון הנעילות** (Locking).

בעיית העדכון האבוד (Lost Update)

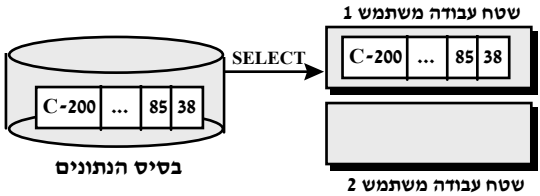
בעיה ראשונה היכולה להיגרם כתוצאה מעדכון בו-זמני נקראת **בעיית העדכון האבוד**. תרשים 14.13 מציג את מצב בסיס הנתונים לפני תחילת העבודה.



תרשים 14.13: בסיס הנתונים לפני תחילת עבודה.

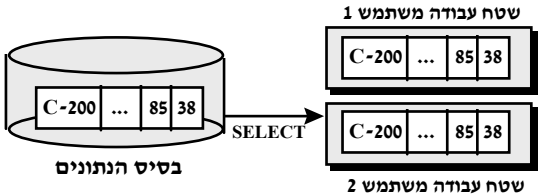
כפי שניתן לראות, טבלת הקורסים מכילה שורה עבור קורס C-200 והמצב הוא שמספר הסטודנטים המקסימלי הוא 85 ומספר הסטודנטים שכבר נרשמו הוא 38. להלן האירועים המתרחשים לאורך ציר הזמן:

- **זמן t1** – סטודנט א' מבצע Login ובוחר מתוך תפריט את פעולת הרישום לקורס.
- **זמן t2** – תוכנית היישום המטפלת ברישום סטודנטים לקורס נטענת לזיכרון ומבקשת מהסטודנט את מספר הקורס שאליו הוא רוצה להירשם.
- **זמן t3** – הסטודנט מבקש להירשם לקורס C-200.
- **זמן t4** – תוכנית היישום מבצעת SELECT ושולפת את השורה של קורס C-200 מבסיס הנתונים אל שטח העבודה שהוקצה במיוחד עבור סטודנט א'.



תרשים 14.14: בסיס הנתונים ושטחי העבודה לאחר קריאה של משתמש א'

- **זמן t5** – בנקודת זמן זו, סטודנט ב' מבצע Login בתחנת עבודה אחרת ובוחר את אותו יישום ומבקש גם הוא להירשם לאותו הקורס.

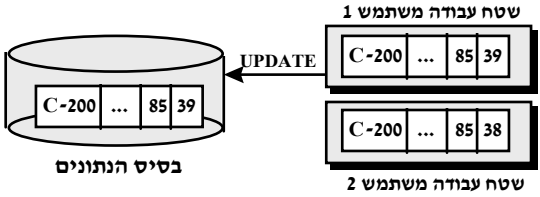


תרשים 14.15: בסיס הנתונים ושטחי העבודה לאחר קריאה של משתמש ב'

בנקודת זמן זו, הנתונים של קורס C-200, נמצאים פעמיים בזיכרון המחשב. פעם אחת בשטח העבודה של המשתמש הראשון ופעם שנייה בשטח העבודה של המשתמש השני. כפי שהסברנו, במערכת המשרתת מספר רב של משתמשים, אותם נתונים יכולים להימצא במספר רב של שטחי עבודה שונים בזיכרון, כמספר המשתמשים שהפעילו תוכנית יישום המבקשת את הנתונים האלה. נדגיש שאלה לא בהכרח אותן תוכניות יישום. ייתכן שגם תוכנית יישום אחרת משתמשת באותם הנתונים. לדוגמה גם תוכנית היישום המאפשרת למזכירות האקדמית לעדכן את המספר המקסימלי של סטודנטים לקורס, משתמשת באותם נתונים. אם באותו זמן שהסטודנטים מבקשים להירשם לקורס גם המזכירות האקדמית מבקשת לעדכן את נתוני הקורס, נקבל מצב שנתוני הקורס נמצאים במספר שטחי עבודה שונים באותה נקודת זמן.

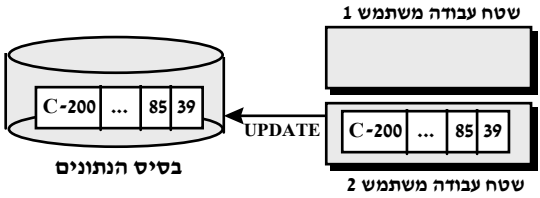
זוהי דוגמה לפעולת קריאה הנקראת Dirty Read מכיון שהיא קראה נתונים שנמצאים באמצע העדכון על ידי תנועה שעדיין לא ביצעה Commit.

- **זמן t6** – במצב זה, סטודנט א' לאחר שראה שיש מקום בקורס, מאשר את ההרשמה. תוכנית היישום מגדילה את מספר הסטודנטים שנרשמו כבר ומייד אחר כך תעדכן את בסיס הנתונים על ידי הפקודה UPDATE ומסיימת את עבודתה על ידי ביצוע Commit. פעולה זו משחררת את שטחי העבודה שהוקצו עבור המשתמש הראשון.



תרשים 14.16: בסיס הנתונים ושטחי העבודה לאחר רישום לקורס של משתמש א'.

- **זמן t7** – עכשיו גם סטודנט ב' מאשר את הרישום לאותו הקורס. בדומה למה שקרה עבור הסטודנט הראשון, תוכנית היישום מגדילה את מספר הסטודנטים שנרשמו ומבצעת את העדכון. מכיון שבשטח העבודה של התוכנית מספר הסטודנטים שכבר נרשמו הוא 38, היא תגדיל מספר זה ל-39 ותעדכן את בסיס הנתונים.



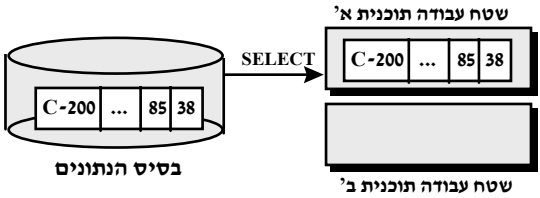
תרשים 14.17: בסיס הנתונים ושטחי העבודה לאחר רישום לקורס של משתמש ב'.

כפי שניתן לראות, תוכנית היישום המטפלת ברישום סטודנטים לקורס פעלה נכון, אולם התוצאה הסופית היא ששני סטודנטים נרשמו לאותו קורס אבל בבסיס הנתונים מספר הסטודנטים שנרשמו גדל רק באחד, כלומר איבדנו עדכון אחד של בסיס הנתונים. הטעות נובעת כמובן מהעובדה שאותה תוכנית היישום הופעלה משתי תחנות עבודה שונות, וכל משתמש קרא את אותם הנתונים מבסיס הנתונים לשטחי העבודה שבזיכרון המחשב, מבלי להתייחס לעובדה שהנתונים עודכנו בינתיים על ידי משתמש אחר.

עדכון רשומה שנמצאת בתהליך עדכון (Uncommited Update)

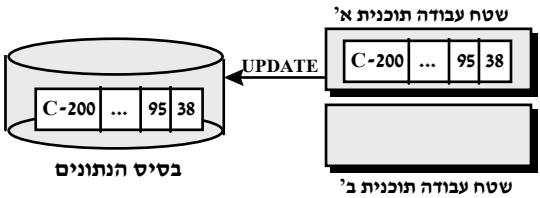
בעיה זו נוצרת כאשר תוכנית יישום אחת, להלן תוכנית א', יכולה לעדכן שורה שעודכנה על ידי תוכנית יישום אחרת, להלן תוכנית ב', שעדיין לא סיימה את פעולתה, כלומר עדיין לא ביצעה פעולת Commit. הבעיה יכולה להיווצר אם תוכנית ב' תבצע גלילה לאחור (Rollback) ואז הנתונים שהיו בתהליך העדכון יוחזרו למצבם לפני העדכון. בינתיים תוכנית א' כבר השתמשה לצרכיה בנתונים כפי שהיו באמצע העדכון.

כדי להדגים את הבעיה נניח שמופעלות שתי תוכניות יישום שונות, אחת לעדכון המספר המקסימלי של סטודנטים לקורס והשנייה לרישום לקורס. הראשונה מופעלת על ידי משתמש במזכירות האקדמית של המחלקה השנייה מופעלת על ידי סטודנט בתחנת עבודה בקמפוס.



תרשים 14.18: מצב בסיס הנתונים ושטחי העבודה לאחר תחילת תוכנית א'.

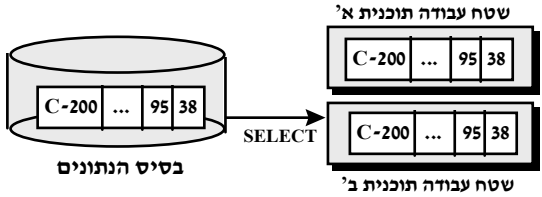
- **זמן t1** – תוכנית א' המעדכנת את מספר הסטודנטים המקסימלי, התחילה לפעול ושלפה את נתוני קורס C-200. נניח שתוכנית זו מבצעת תנועה הכוללת שתי פעולות עדכון ברצף: עדכון נתוני הקורס ולאחר מכן עדכון של טבלת המחלקות.



תרשים 14.19: מצב בסיס הנתונים ושטחי העבודה לאחר עדכון ראשון של תוכנית א'.

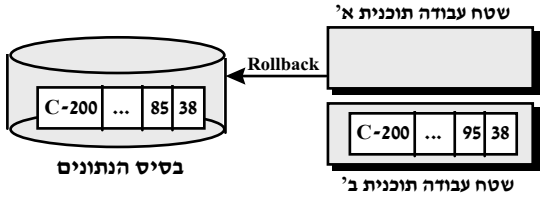
- **זמן t2** – תוכנית א' החלה לבצע את פעולת העדכון הראשונה בתנועה, כלומר היא עדכנה את מספר הסטודנטים המקסימלי בטבלת הקורסים. נשים לב שמספר הסטודנטים המקסימלי הוגדל מ-85 ל-95.
- **זמן t3** – בנקודת זמן זו, תוכנית ב' שמבצעת רישום סטודנטים לקורס הופעלה על ידי סטודנט כלשהו ושלפה גם היא את נתוני הקורס. נחזור ונדגיש שהפעלת תוכנית ב' בוצעה בנקודת זמן שבה תוכנית א' הספיקה לבצע את פעולת העדכון הראשונה ועדיין לא ביצעה את השנייה, כלומר היא נמצאת באמצע התנועה. זוהי דוגמה

לפעולת קריאה הנקראת Dirty Read, מאחר והיא קראה נתונים שעודכנו על ידי תנועה שעדיין לא ביצעה Commit. למצב בסיס נתונים במצב של אמצע תנועה קראנו כאמור מצב בסיס נתונים לא תקין.



תרשים 14.20: מצב בסיס הנתונים ושטחי העבודה לאחר תחילת תוכנית ב'.

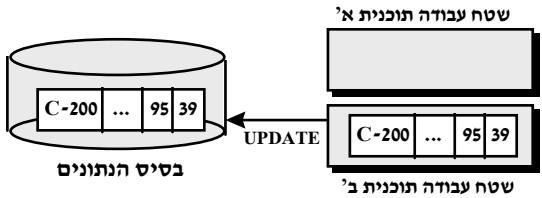
- **זמן t4** – תוכנית א' מנסה לבצע את העדכון השני, אבל מסיבה כלשהי היא אינה מצליחה ומחליטה לבצע באופן יזום פעולת Rollback – גלילה לאחור. אגב, אותה בעיה היתה מתרחשת אם המשתמש היה מבקש להפסיק את פעולת התוכנית או שהיא היתה נופלת ומפסיקה לפעול מסיבה כלשהי. מערכת RDBMS תבצע גלילה לאחור ומצב טבלת הקורסים מוחזר לקדמותו.



תרשים 14.21: מצב בסיס הנתונים ושטחי העבודה לאחר Rollback של תוכנית א'.

נשים לב, שלמרות שתוכנית א' ביצעה גלילה לאחור והחזירה את מצב בסיס הנתונים לקדמותו, הנתונים כבר נמצאים בשטח העבודה של תוכנית ב' וכוללים את העדכון שתוכנית א' ביצעה. בבסיס הנתונים רשום כרגע שמספר הסטודנטים המקסימלי לקורס הוא 85 ואילו בשטח העבודה של תוכנית ב' רשום 95 סטודנטים.

- **זמן t5** – במצב זה, הסטודנט מאשר את הרישום לקורס ולכן תוכנית היישום מגדילה את מספר הסטודנטים הרשומים ומבצעת עדכון של בסיס הנתונים.



תרשים 14.22: מצב בסיס הנתונים ושטחי העבודה לאחר סיום תוכנית ב'.

כפי שניתן לראות, בסיס הנתונים משובש. מזכירות הסטודנטים לא הצליחה לבצע את התנועה להגדלת מספר הסטודנטים המקסימלי לקורס, אולם בבסיס הנתונים עדכון זה כן נרשם.

ניתוח נתונים לא-עִקבי (Inconsistent Analysis)

ניתוח נתונים לא עקבי יכול לקרות אפילו במצב שבו רק תוכנית אחת מעדכנת את בסיס הנתונים. לדוגמה נניח שפיתחנו תוכנית להפקת דוח המציג ניתוח מסוים על מספר הסטודנטים שנרשמו עד כה לפי המחלקות השונות. נקרא לתוכנית זו בשם תוכנית א'. תוכנית זו מתחילה לעבוד ולשלוף נתונים מבסיס הנתונים. תוך כדי עבודת התוכנית, מתחילה לפעול תוכנית ב' (התוכנית לרישום סטודנטים לקורסים), ומעדכנת את נתוני אחד הקורסים, שכבר נקראו על ידי תוכנית א'. כאשר תוכנית ב' מסיימת לעבוד היא מציגה ניתוח שכבר אינו נכון, מאחר ותוך כדי פעולתה, בסיס הנתונים התעדכן. זו דוגמה למצב שבו תוכנית א' מבצעת ניתוח של הנתונים ותוך כדי פעולתה תוכניות יישום אחרות מעדכנות את בסיס הנתונים. זהו שוב מצב של Dirty Read, מאחר והתוכנית חשופה לשורות שעודכנו על ידי תנועות שעדיין לא ביצעו Commit.

אם תוכנית א' תפעיל תוך כדי פעולתה מספר פעמים את אותה פקודת SELECT לשליפת נתונים, היא עלולה לקבל בכל פעם תוצאה אחרת, כי בסיס הנתונים מתעדכן תוך כדי פעולתה. מכיון שלא ניתן לחזור על הפקודה SELECT מספר פעמים ולהבטיח שתתקבל אותה תוצאה, מקובל לקרוא למצב זה בשם Unrepeatable Read (קריאה שאינה חוזרת על עצמה). לפעמים הניתוח שתוכנית מסוימת מבצעת רגיש, ולכן צריך ליצור מצב שבו בסיס הנתונים יהיה במצב תקין במהלך כל פעולתה ולא תתבצע אף תנועה אחרת באותו זמן.

כפי שנראה מייד, הפתרון לשלוש הבעיות שהצגנו הוא על ידי שימוש במנגנון מיוחד של מערכת RDBMS, מנגנון הנעילות.

מנגנון נעילות (Locking)

אחד היתרונות הגדולים של מערכת מקוונת הינו זמן תגובה קצר. הדבר מושג על ידי אפשרות הפעלת מספר תוכניות יישום שונות במקביל ובאופן בו-זמני (Concurrently) על בסיס נתונים אחד, כאשר כל תוכנית יישום יכולה לשרת בו-זמנית מספר רב של משתמשים. מערכת RDBMS ערוכה לפעילות מקבילה זו והיא המפקחת על הביצוע. אם המערכת היתה מארגנת את כל הגישות לבסיס הנתונים לפי תור מסוים כדי למנוע מצב של עדכון בו-זמני, זמן התגובה היה ארוך ובלתי נסבל. מצד שני הראינו בדוגמאות קודמות שגישה ללא תיאום ופיקוח עלולה לגרום לפגיעה באמינות הנתונים המנוהלים בבסיס הנתונים.

מערכת RDBMS צריכה לאפשר פעולה בו-זמנית ובמקביל של תוכניות יישום רבות ככל האפשר, כדי שזמן התגובה לכל משתמש יהיה קצר ככל הניתן. אם תוכניות היישום קוראות את הנתונים בלבד, אין כל חשש לתקלה מסוג זו שהודגמה, והאפשרות להפעיל

מספר רב של תוכניות במקביל אמנם תורמת לשיפור זמן התגובה. מאידך, אם תוכניות היישום גם מעדכנות את בסיס הנתונים, עלולה להיווצר בעיה של עדכון שגוי. כאן נכנס לפעולה מנגנון מיוחד, **מנגנון הנעילות**.

אחד הפתרונות המקובלים לבעיה שהוצגה כאן הוא הפעלת **מנגנון נעילה** (Locking Facility), כך שנתונים שנמצאים בתהליך עדכון "ינעלו", כלומר יהיו חסומים לגישה על ידי משתמשים נוספים. התוכנית הראשונה המבקשת לעדכן את הנתונים הופכת באופן זמני להיות **הבעלים** (Owner) של הנתונים, וכל שאר תוכניות היישום המבקשות לעדכן את אותם הנתונים תצטרכנה להמתין עד שהתוכנית הראשונה תשחרר אותם. נגדיר תחילה את מושג הנעילה.

נעילה **Lock**	***נעילה* מוגדרת כפעולה המבצעת סימון של אובייקט בבסיס הנתונים (שורה, טבלה וכד'), כדי להצביע על כך שהאובייקט נמצא כרגע בתהליך של עדכון ולכן הוא חסום לגישה על ידי משתמשים נוספים.**

מנגנון הנעילה פועל כמנגנון סינכרון בין תוכניות היישום השונות המבקשות לעדכן את אותו אובייקט בבסיס הנתונים. כאשר תוכנית מבקשת לנעול אובייקט לא נעול, היא תורשה לעשות זאת ולהמשיך בפעולתה. אם האובייקט כבר נעול, היא תיכנס לתור התוכניות הממתינות לאובייקט זה עד אשר התוכנית הקודמת לה תשחרר את הנעילה.

נדגיש שקיימות שיטות נוספות לפתרון בעיית העדכון הבו-זמני, שיטות שאינן מבוססות על נעילות אלא על מנגנונים אחרים. שיטות אלו מאפשרות לתנועות לבצע את עבודתן ולפני עדכון לגלות אם תוך כדי פעולת התנועה, הנתונים שנקראו על ידה עודכנו בינתיים על ידי תנועות אחרות. אחת השיטות מבוססת על **חותמת הזמן** (Timestamping) שבה כל אובייקט שנשלף מקבל חותמת חד ערכית המציינת את שם התנועה ששלפה את האובייקט ואת הזמן היחסי להתחלת התנועה. המערכת מרשה לעדכן אובייקטים רק אם היא מזהה שחותמת הזמן של האובייקט לא עודכנה על ידי תנועה אחרת. אם זה המצב, המערכת מבצעת Rollback של התנועה. שיטות אלו פחות נפוצות במערכות RDBMS מסחריות ולכן לא נפרט אותן. נרחיב את הדיון בשיטה הנפוצה ביותר במערכות RDBMS, שיטת הנעילות.

רמת הנעילה (Locking Granularity)

רמת נעילה **Lock Granularity**	***רמת הנעילה* מוגדרת כתחום בסיס הנתונים שכפוף להוראת הנעילה.**

נעילת תחום קטן יותר, תתיר פעילות מקבילה ובו-זמנית רבה יותר. עם זאת, נעילה ברמה הנמוכה ביותר (עמודה בתוך שורה לעומת טבלה, למשל) הינה מורכבת יותר. תרשים 14.23 מציג את רמות הנעילה האפשריות.

תרשים 14.23: רמות הנעילה של אובייקטים בבסיס הנתונים.

נפרט כל אחת מרמות הנעילה האפשריות:

- **עמודה בתוך שורה (Column Level Locking):** זוהי רמת הנעילה הגבוהה ביותר האפשרית. בדוגמה ראינו, שתוכנית היישום לרישום סטודנט לקורס מעדכנת רק את העמודה "מספר הסטודנטים שנרשמו" בשורת הקורס. נעילה ברמת העמודה בשורה מסוימת מאפשרת מניעת גישה מתוכניות אחרות המבקשות לעדכן את אותה עמודה באותה שורה, אולם אינה חוסמת את הגישה של תוכניות אחרות לאותה שורה כדי לעדכן בה עמודות אחרות. בגלל המורכבות הרבה של נעילת עמודה בודדת בתוך שורה, או אפילו קבוצת עמודות בשורה, רוב מערכות RDBMS המסחריות אינן תומכות בנעילה ברמה זו. בדוגמה שלנו, תוכנית היישום המעדכנת את המספר המקסימלי של סטודנטים בקורס אינה מעדכנת את מספר הסטודנטים שנרשמו ואילו תוכנית היישום המבצעת רישום של סטודנטים לקורס אינה מעדכנת את מספר הסטודנטים המקסימלי. עקרונית ניתן היה לאפשר לשתי תוכניות היישום האלו לפעול בו-זמנית על אותה שורה, כמובן אם ניתן היה לבצע נעילה של עמודה בודדת.
- **רשומה (Row Level Locking):** כל הפעולות של תוכנית יישום בבסיס נתונים מתבצעות על שורות מסוימות. על כן ניתן לקבוע את השורה הבודדת כיעד הנעילה. זוהי רמה נמוכה יותר מרמת העמודה בתוך שורה, מאחר ומערכת RDBMS תבצע נעילה של כל השורה, מבלי להתייחס לשאלה איזה מהעמודות בשורה מתעדכנות. שיטה זו תמנע גישה לשורה נעולה, גם אם תוכנית היישום מתכוונת לעדכן עמודה שאינה מתעדכנת על ידי התוכנית שנעלה את השורה. בדוגמה שלנו, תוכנית היישום הראשונה היתה נועלת את השורה ומונעת מתוכנית היישום השנייה לעדכן את אותה שורה אבל מאפשרת לה לעדכן שורות אחרות. רוב מערכות RDBMS המסחריות תומכות בשיטת נעילה זו, אם כי לא כולן.
- **דף (Page Level Locking):** פעולות הקלט/פלט מתבצעות תמיד ברמה של דף פיסי (Page), שיכול להכיל שורה אחת או יותר. על כן ניתן לקבוע את הדף כיעד הנעילה. זוהי נעילה ברמה נמוכה יותר מאשר רמת השורה, מאחר ובשיטה זו ננעלות גם שורות שאינן משתתפות כלל בתהליך העדכון, אך נמצאות באותו דף פיסי שבו נמצאת השורה שמתעדכנת. בגלל הנוחות במימוש נעילה ברמת הדף, חלק מהמערכות המסחריות מיישמות נעילה ברמה זו. בדוגמה שלנו, נניח שטבלת הקורסים נשמרת בבסיס הנתונים בדפים המכילים 10 שורות כל אחד. תוכנית יישום שתרצה לעדכן שורה כלשהי, תשלוף את השורה לזיכרון ותסמן את כל הדף כנעול. משמעות הדבר היא שתשע שורות של טבלת הקורסים ננעלו למרות שתוכנית היישום אינה מתכוונת לעדכן אותן. כמובן ששיטה זו תאפשר רמת עבודה במקביל נמוכה יותר מזו של נעילה ברמת שורה.

- **טבלה (Table Level Locking):** זוהי רמת נעילה נמוכה יותר מרמת הדף. במצב זה המערכת תבצע נעילה של כל הטבלה, גם אם רק שורה בודדת מתוכה משתתפת בעדכון. זוהי שיטה קלה למימוש, אולם היא מטילה הגבלות חמורות על המשתמשים ומקטינה באופן דרמטי את רמת העבודה במקביל שבסיס הנתונים יכול לספק. בדוגמה שלנו, משמעות השיטה היא שכל טבלת הקורסים ננעלת כל אימת שתוכנית יישום תבצע עדכון של שורה כלשהי. למרות ששיטה זו אינה טובה למצבי עבודה בסביבה מרובת משתמשים, קיימים מצבים בהם מבקשים לנעול בצורה יזומה את כל הטבלה. לדוגמה, אם רוצים להבטיח שתוך כדי ריצת תוכנית יישום כלשהי, תוכניות אחרות לא ישנו את מצב בסיס הנתונים. אם ניזכר לרגע בבעיית הניתוח הלא עקבי, נוכל לנעול את טבלת הקורסים כל עוד תוכנית הניתוח פועלת. מערכות RDBMS מסוימות מספקות את הפקודה LOCK TABLE, המאפשרת לנעול טבלה שלמה למטרות תחזוקה או למטרות אחרות.
- **בסיס הנתונים (Database Level Locking):** זוהי רמת הנעילה הנמוכה ביותר האפשרית וכמובן הקלה ביותר למימוש. עם תחילת תהליך העדכון, ננעל כל בסיס הנתונים ואף תוכנית אחרת אינה מורשית לגשת אליו. שיטת נעילה זו דרושה למצבים מאוד מיוחדים ונדירים.

סוג הנעילה (Lock Type)

כדי להתמודד עם הצורך לאפשר רמה גבוהה ככל שניתן של עבודה במקביל, מיישמות רוב מערכות RDBMS המסחריות מספר סוגי נעילה שונים. מקובל להבחין בין שני סוגי נעילה: נעילה בלבדית ונעילה שיתופית.

- **נעילה בלבדית (Exclusive Lock)**: תוכנית יישום המקבלת נעילה בלבדית על אובייקט בבסיס הנתונים, יכולה לעדכן אותו. אף תוכנית יישום אחרת אינה מורשית לעדכן את הנתונים באותו זמן, ואפילו לא לקרוא אותם. לדוגמה, תוכנית היישום המעדכנת את מספר הסטודנטים שנרשמו לקורס, יכולה לבצע נעילה בלבדית של השורה (או הדף) שהיא מעדכנת ולמנוע מתוכניות יישום אחרות אפילו לקרוא את נתוני הקורס ובודאי למנוע מהן לעדכן את נתוני הקורס. מקובל לקרוא לסוג נעילה זה גם בשם **נעילת כתיבה** (Write Lock).
- **נעילה שיתופית (Shared Lock):** תוכנית יישום המקבלת נעילה שיתופית על אובייקט בבסיס הנתונים, תורשה רק לקרוא את הנתונים אבל לא לעדכן אותם. למעשה, תוכנית יישום המבקשת לקרוא שורה מסוימת מחזיקה אותה בסטטוס של נעילה שיתופית (Shared) ובכך מתירה לתוכניות אחרות לקרוא את אותה השורה, אך לא לעדכן אותה. ברגע שהתוכנית מבקשת לעדכן את השורה, התוכנית צריכה להעביר את הנעילה מנעילה שיתופית לנעילה בלבדית. במצב זה, תוכנית יישום אחרת אינה מורשית לקרוא או לעדכן את השורה וגם לא לנעול את השורה, אפילו לא נעילה שיתופית, עד שהתוכנית שנעלה את השורה מסיימת את העדכון. מקובל גם לקרוא לסוג נעילה זה בשם **נעילת קריאה** (Read Lock).

נתבונן בצורת העבודה עם מנגנון הנעילה, כאשר תוכנית יישום מבקשת נעילת קריאה:

- התוכנית תבקש נעילת קריאה אם היא רוצה לקרוא את השורה אולם אינה מעוניינת לעדכן אותה.
- אם השורה לא נעולה, השורה תִישָלף ותימסר לתוכנית; מערכת RDBMS תרשום שהשורה נמצאת במצב של נעילת קריאה.
- אם השורה נמצאת בנעילת קריאה על ידי תוכנית אחרת, השורה תישלף ותימסר לתוכנית; מערכת RDBMS תרשום שהשורה ננעלה לקריאה על ידי תוכנית נוספת.
- אם השורה נמצאת במצב של נעילת כתיבה, התוכנית המבקשת תוכנס למצב המתנה עד שהתוכנית שנעלה את השורה תסיים את העדכון.

נתבונן בצורת העבודה עם מנגנון הנעילה כאשר תוכנית יישום מבקשת נעילת כתיבה:

- התוכנית תבקש נעילת כתיבה אם היא מעוניינת לעדכן את הנתונים.
- אם השורה לא נעולה, השורה תישלף ותימסר לתוכנית והמערכת תרשום שהשורה נמצאת במצב של נעילת כתיבה.
- אם השורה נמצאת כבר במצב של נעילת קריאה או במצב של נעילת כתיבה, התוכנית המבקשת תוכנס למצב של המתנה עד שכל הנעילות של השורה תשוחררנה.

הטבלה בתרשים 14.24 מציגה את האלגוריתם לטיפול בנעילות בשני המצבים האלה:

תוכנית יישום ב' \ תוכנית יישום א'	לא נעול	נעילת קריאה	נעילת כתיבה
לא נעול		בצע נעילת קריאה	בצע נעילת כתיבה
נעילת קריאה	בצע נעילת קריאה	בצע נעילת קריאה	המתן
נעילת כתיבה	בצע נעילת כתיבה	המתן	המתן

תרשים 14.24: טבלת החלטות לקבלת נעילות.

נציג עכשיו את התנהגות תוכנית היישום לרישום סטודנטים לקורס, המופעלת על ידי שני סטודנטים שונים בו-זמנית, אבל הפעם היא משתמשת במנגנון הנעילה. מהלך העניינים במקרה זה יהיה:

- סטודנט א' מפעיל את תוכנית היישום השולפת את שורת קורס C-200 ומבצעת נעילת כתיבה על השורה.
- סטודנט ב' מפעיל את תוכנית היישום המנסה לשלוף את שורת הקורס ולבצע נעילת כתיבה. מכיון שלשורה כבר יש נעילת כתיבה, תוכנית היישום מוכנסת למצב של המתנה לנעילה (Wait Lock).
- סטודנט א' מאשר את ההרשמה לקורס. תוכנית היישום מגדילה את מונה מספר הסטודנטים שנרשמו לקורס, מעדכנת את השורה ומשחררת את הנעילה שלה.

- תוכנית היישום של סטודנט ב' יוצאת ממצב ההמתנה, מקבלת את השורה ונועלת אותה בנעילת כתיבה. השורה תשוחרר רק לאחר שסטודנט ב' יאשר את ההרשמה או יתחרט.

כפי שניתן לראות, מנגנון הנעילות דאג ששתי תוכניות היישום תבוצענה באופן עוקב ולא במקביל, כלומר זו אחר זו, ולכן מצב בסיס הנתונים נשאר תקין. כמובן שאם סטודנט היה מבקש להירשם לקורס אחר, הוא יכול היה לבצע את ההרשמה ללא הפרעה כלשהי.

נעילה דו-שלבית (Two Phase Locking)

נתייחס למצב שבו תוכנית יישום מבצעת תנועה המעדכנת מספר שורות שונות, זו אחר זו. לפני העדכון של כל שורה, יש לנעול אותה. נשאלת השאלה, מתי לשחרר את השורה שננעלה. קיימות שתי חלופות:

- מייד עם גמר עדכון השורה ועוד לפני שהתנועה הסתיימה.
- רק בגמר העדכון של כל השורות, כלומר בסיום ביצוע התנועה.

היתרון של החלופה הראשונה הוא בכך שהשורה משוחררת ממצב הנעילה מייד עם גמר העדכון ותוכניות יישום אחרות יכולות לגשת אליה מהר יותר. כלומר, זמן הנעילה קצר יחסית. החיסרון של חלופה זו הוא בכך, שבמקרה של כישלון התנועה וצורך בשחזור המצב על ידי גלילה לאחור (Rollback) נוצרת בעיה. בעיה זו תוארה ונקראת Uncommitted Update.

כאשר משחררים שורה מנעילה לפני השלמת התנועה, ייתכן שתוכנית יישום אחרת כבר הספיקה לנעול אותה ואפילו לעדכן אותה. במקרה של כישלון של התנועה הראשונה וצורך בשחזור, מצב השורה "לפני" יוחזר לבסיס הנתונים. שורה זו אינה כוללת את העדכון של תוכנית היישום השנייה ועל כן מצב בסיס הנתונים יהיה לא תקין. נשים לב שכל זה התרחש למרות שהשתמשנו במנגנון נעילה שאמור היה למנוע תקלה כזו.

פרוטוקול נעילה דו-שלבי **Two Phase Locking**	***פרוטוקול נעילה דו-שלבי* קובע שנעילות כל האובייקטים חייבות להתבצע לפני שמתבצע השחרור מנעילה של אובייקט כלשהו.**

כדי להתמודד עם בעיה זו, מערכות RDBMS מיישמות מנגנון נעילה דו-שלבית. מנגנון זה מבוסס על מדיניות פשוטה: אסור לתנועה לבצע נעילה כלשהי, אם היא כבר שחררה אובייקט כלשהו.

בשלב ראשון המערכת תבצע רק את הנעילות, כלומר מספר האובייקטים הנעולים הולך וגדל. מקובל לקרוא לשלב זה בשם Growing Phase. רק לאחר ביצוע Commit של התנועה מתחיל השלב השני, הוא שלב שחרור הנעילות. לשלב זה שבו מספר הנעילות הולך וקטן, מקובל לקרוא בשם Shrinking Phase. הפרוטוקול הדו-שלבי לניהול נעילות אמנם מאריך את זמן הנעילה של שורה, אולם הוא מבטיח שבמקרה של תקלה לא יימחקו מבסיס הנתונים, עדכונים שלא היו אמורים להמחק.

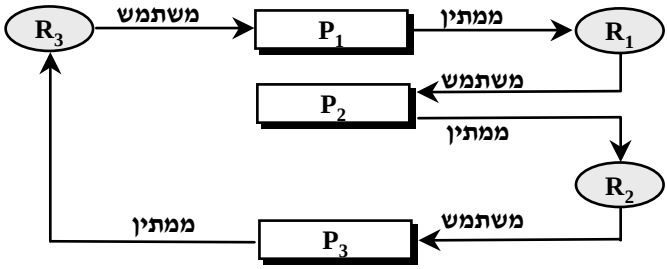
נעילה ללא מוצא (Deadlock)

השימוש במנגנון הנעילה עלול להביא למצבים שבהם שתי תוכניות יישום (או יותר) ייכנסו למצב של **נעילה ללא מוצא**, וזאת מכיון שכל אחת מהן תנעל אובייקט שהתוכנית השנייה מבקשת להשתמש בו. נתבונן לדוגמה על שתי תוכניות יישום, המעדכנות שתי שורות שונות אולם בסדר הפוך. תוכנית יישום הראשונה מעדכנת תחילה את שורה A ולאחר מכן את שורה B, ואילו תוכנית היישום השנייה מעדכנת תחילה את שורה B ולאחר מכן את שורה A. הפעלה בו-זמנית של שתי תוכניות יישום אלו עלולה להביא למצב, שבו שתיהן תיכנסנה למצב נצחי של המתנה לשורה. מקובל לקרוא למצב זה בשם **נעילה ללא מוצא**. מהלך הפעולות הוא:

- תוכנית א' מתחילה לפעול ושולפת שורה A ומבצעת לה (לשורה) נעילת כתיבה לקראת עדכון.
- תוכנית ב' מתחילה לפעול ושולפת שורה B ומבצעת לה נעילת כתיבה לקראת עדכון.
- תוכנית א' מבקשת לקרוא את שורה B ולבצע לה נעילת כתיבה. מכיון ששורה זו כבר נעולה, התוכנית מוכנסת למצב של המתנה (Wait Lock).
- תוכנית ב' מבקשת לקרוא את שורה A ולבצע נעילת כתיבה. אך מכיון ששורה זו כבר נעולה, התוכנית מוכנסת למצב של המתנה.

כלומר, לפנינו שתי תוכניות שנמצאות במצב המתנה לשחרור הרשומות, שלא יתרחש לעולם! היחלצות מצב של נעילה ללא מוצא יכול להתבצע במספר צורות:

- **ביצוע כל הנעילות לפני העדכון:** כל תוכנית יישום צריכה להכריז מראש על כל השורות שהיא מבקשת לנעול. מערכת RDBMS מתחילה לבצע את הנעילות. אם תוך כדי הנעילות, שורה מסוימת כבר נעולה ולא ניתן לנעול אותה, משחררים את כל השורות שכבר ננעלו והתהליך מתחיל מחדש. שיטה זו אמנם מבטיחה שמצב של נעילה ללא מוצא יימנע, אולם לא ניתן ליישם אותה כי בדרך כלל תוכנית היישום אינה יודעת מראש את כל השורות שהיא תרצה לעדכן.
- **איתור מצב נעילה ללא מוצא:** מערכת RDBMS מאפשרת לתוכניות היישום לבצע את הנעילות על פי הצורך ומנהלת מעקב אחר כל האובייקטים הנעולים ואחר כל האובייקטים שתוכנית כלשהי אחרת ממתינה לשחרורם. המערכת קובעת **זמן המתנה** (Time Out) כלשהו כך שאם תוכנית יישום ממתינה מעבר לו, תופעל פעולה לפתרון הבעיה. המערכת עוקבת אחר זמן ההמתנה של כל תוכנית וברגע שזמן זה עבר, מופעל אלגוריתם **לזיהוי נעילות ללא מוצא** (Deadlock Detection Algorithm). האלגוריתם סורק את תור התוכניות הממתינות לאובייקט כלשהו ואת רשימת האובייקטים הנעולים על ידי כל תוכנית. הוא בונה גרף התאמה בין האובייקטים לבין התוכניות וכך הוא מאתר מצבי נעילה ללא מוצא. מצב זה מיוצג על ידי מצב של לולאה: מעגל סגור (Cycle) הנבנה על ידי חץ היוצא מהתוכנית אל האובייקט שהיא ממתינה לו וחץ מהאובייקט אל התוכנית המשתמשת בו. נעיין לדוגמה, במצב של נעילה ללא מוצא בין שלוש תוכניות יישום שונות (P_1, P_2, P_3) המבקשות לעדכן שלוש שורות (R_1, R_2, R_3) שונות.



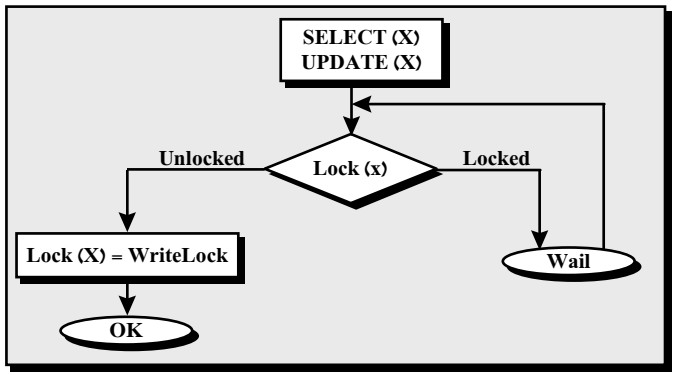
תרשים 14.25: גרף לאיתור נעילה ללא מוצא.

תוכנית P_1 נעלה את שורה R_3 וממתינה לשורה R_1. תוכנית P_2 נעלה את שורה R_1 וממתינה לשורה R_2. תוכנית P_3 נעלה את שורה R_2 וממתינה לשורה R_3. אם נצייר מצב זה, נקבל לולאה בין שלושת האובייקטים ותוכניות היישום המשתמשות או ממתינות להם. כפי שניתן לראות, מאחר ויש לנו כאן מעגל סגור, זהו מצב של נעילה ללא מוצא.

כאשר המערכת מגלה מצב זה כתוצאה מהפעלת האלגוריתם לאיתור מצבי נעילה ללא מוצא, היא תבחר תוכנית יישום כלשהי ותפסיק את פעולתה. הפסקת הפעולה כרוכה בביצוע שחזור המצב על ידי גלילה לאחור וביטול כל העדכונים שהתוכנית כבר הספיקה לבצע. גלגול חזרה זה ישחרר את כל האובייקטים שננעלו על ידי אותה תוכנית. בחירת התוכנית שתופסק יכולה להתבסס על קריטריונים שונים כמו התנועה האחרונה שהתחילה לפעול, או על פי התנועה שהספיקה לבצע הכי מעט עדכונים, או על פי קריטריון אחר כלשהו.

ביצוע הנעילה על ידי תוכנית היישום

עד כאן ראינו כיצד מנגנון הנעילות של מערכת RDBMS פועל, אבל לא הראינו פקודת SQL כלשהי שבאמצעותה תוכנית היישום מבצעת נעילות או משחררת אותן. אם כן, כיצד מבצעים את הנעילות? עד כמה שהדבר יישמע מפתיע, אין כל צורך בפקודות מפורשות לנעילות, מכיון שמערכת RDBMS מבצעת אותן באופן אוטומטי. מקובל לומר שמנגנון הנעילות פועל בצורה לא-מפורשת (Implicit). נציג תחילה את עיקרון הפעולה בסביבה התומכת בנעילת כתיבה בלבד:



תרשים 14.26: עקרון הפעולה בסביבה התומכת בנעילות כתיבה בלבד.

כאשר תוכנית יישום מתחילה לפעול, המערכת עוקבת אחר כל פקודה שהתוכנית מבצעת. אם התוכנית מבצעת פקודה כלשהי על שורה בטבלה, המערכת בודקת אם קיימת נעילה כלשהי על השורה. אם קיימת, התוכנית מוכנסת להמתנה. אם אין נעילה, המערכת רושמת נעילת כתיבה על השורה ומוסרת את השורה לתוכנית היישום. עם ביצוע הפקודה COMMIT או עם סיום התוכנית, כל הנעילות השייכות לתוכנית משתחררות. עם ביצוע הפקודה ROLLBACK כל העדכונים שהתוכנית ביצעה מתבטלים וכל הנעילות שלה משתחררות.

כפי שניתן לראות שיטה פשוטה זו מתבססת על נעילות כתיבה בלבד. כל נגיעה בשורה כלשהי על ידי תוכנית היישום מבצעת מייד נעילת כתיבה על השורה.

פעולת נעילה בין שתי תוכניות

נדגים את האופן שבו המערכת מנהלת את הנעילות כאשר שתי תוכניות יישום שונות, TR_1 ו- TR_2 **פועלות בו-זמנית** (ראה תרשימים 14.27 א' ו- 14.27 ב'). תוכנית היישום TR_1 קוראת שורה של קורס C-200 מהטבלה קורסים ומעדכנת אותה. תוכנית היישום TR_2 קוראת את השורה של מחלקה CS מהטבלה מחלקות, קוראת את השורה של קורס C-200 מהטבלה קורסים ולבסוף מעדכנת את השורה של מחלקה CS בטבלה מחלקות.

תוכנית יישום א'

```
.
EXEC SQL SELECT *
            FROM COURSES
            WHERE COURSE_ID = 'C-200';
IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
EXEC SQL UPDATE COURSES
             SET POINTS = 3
            WHERE COURSE_ID = 'C-200';
IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
EXEC SQL COMMIT;
.
```

תרשים 14.27 א': דוגמה לתוכנית יישום TR_1 המעדכנת את טבלת הקורסים והמחלקות.

תוכנית יישום ב'

```
.
EXEC SQL SELECT *
            FROM DEPARTMENTS
            WHERE DEPART_ID = 'CS';
EXEC SQL SELECT *
             FROM COURSES
            WHERE COURSE_ID = 'C-200';
IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
EXEC SQL UPDATE DEPARTMENTS
             SET HEAD = 'Prof. Eyal'
            WHERE DEPART_ID = 'CS';
IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
EXEC SQL COMMIT;
.
```

תרשים 14.27 ב': דוגמה לתוכנית יישום TR_2 המעדכנת את טבלת הקורסים והמחלקות.

הטבלה שבתרשים 14.28 מציגה את מצב הנעילות של שני אובייקטים בבסיס הנתונים: שורה של קורס C-200 בטבלת הקורסים ושורה של מחלקה CS בטבלת המחלקות.

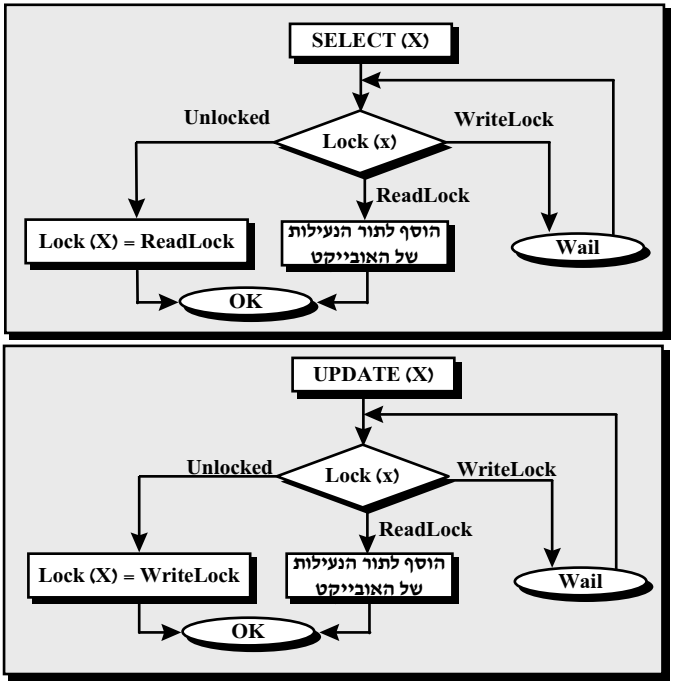
שורה	תנועה	פקודה	סטטוס בקשה	סטטוס קורס C-200	סטטוס מחלקה CS
01				Unlocked	Unlocked
02	TR_1	Select (Courses)	OK	WriteLock	Unlocked
03	TR_2	Select (Depart)	OK	WriteLock	WriteLock
04	TR_2	Select(Courses)	Wait	WriteLock	WriteLock
05	TR_1	Update (Courses)	OK	WriteLock	WriteLock
06	TR_1	Commit	OK	Unlocked	WriteLock
07	TR_2	Select (Courses)	OK	WriteLock	WriteLock
08	TR_2	Update (Depart)	OK	WriteLock	WriteLock
09	TR_2	Commit	OK	Unlock	Unlock

תרשים 14.28: פעולת שתי תוכניות יישום בסביבה התומכת רק בנעילת כתיבה

הסבר:

- שורה 1 – בשלב זה שני האובייקטים נמצאים במצב לא נעול.
- שורה 2 – תוכנית יישום א' מתחילה לפעול ומבצעת את הפקודה SELECT מטבלת הקורסים. השורה של קורס C-200 ננעלת.
- שורה 3 – תוכנית יישום ב' מתחילה לפעול ומבצעת SELECT על טבלת המחלקות. השורה של מחלקה CS ננעלת.
- שורה 4 – תוכנית ב' מבצעת SELECT על טבלת הקורסים. מאחר והשורה המבוקשת נמצאת במצב של נעול, תוכנית ב' נכנסת למצב המתנה. נשים לב שלמרות שתוכנית ב' לא מתכוונת כלל לעדכן את טבלת הקורסים, היא בכל זאת נכנסת להמתנה, מאחר ובדוגמה זו הנחנו שהמערכת RDBMS משתמשת רק בנעילות כתיבה.
- שורה 5 – תוכנית יישום א' מבצעת עדכון של שורת הקורס. מכיון שהשורה כבר נעולה, העדכון מתבצע.
- שורה 6 – תוכנית א' מסיימת את פעולתה ומבצעת Commit. פעולה זו משחררת את השורה של הקורס ממצב של נעילה והיא חוזרת לסטטוס Unlocked.
- שורה 7 – מערכת RDBMS מודיעה לתוכנית ב' שהשורה המבוקשת השתחררה והיא יכולה להתחיל לעבוד. מאחר והתוכנית שולפת את אותה שורה, השורה נכנסת מחדש למצב של נעילה.
- שורה 8 – תוכנית יישום ב' מעדכנת את טבלת המחלקות. מכיון שהשורה כבר נעולה, העדכון מתבצע.
- שורה 9 – תוכנית יישום ב' מסיימת את פעולתה ומבצעת Commit. פעולה זו משחררת את שתי השורות שננעלו על ידה.

כפי שניתן לראות מדוגמה זו, למרות ששתי תוכניות היישום פועלות במקביל, מנגנון הנעילות האוטומטי מבטיח שכל השורות הנשלפות על ידי תוכנית כלשהי ננעלות ולכן אין סכנה לאמינות הנתונים. כדי לשפר את רמת העבודה במקביל, המערכות המסחריות משתמשות גם במנגנון של נעילת קריאה, הנרשם לאחר הפקודה SELECT. תרשים הזרימה 14.29 מציג את עיקרון הפעולה בסביבה התומכת בנעילת קריאה וכתיבה.



תרשים 14.29: עקרון הפעולה בסביבה התומכת בנעילות קריאה וכתיבה.

כפי שניתן לראות מהתרשים, התמיכה במנגנון של נעילת קריאה וכתיבה מחייב ניהול של רשימת נעילות קריאה לאובייקט. כל נעילת קריאה נרשמת ברשימה יחד עם זיהוי התנועה שביצעה את הנעילה. בזמן עדכון המערכת תאפשר עדכון רק אם האובייקט אינו נעול או שרשימת נעילות הקריאה אינה מכילה תנועות אחרות.

הטבלה שבתרשים 14.30 מציגה את אותן שתי תוכניות יישום, אבל הפעם בסביבה התומכת גם בנעילת קריאה.

שורה	תנועה	פקודה	סטטוס בקשה	סטטוס קורס C-200	סטטוס מחלקה CS
01				Unlocked	Unlocked
02	TR_1	Select (Courses)	OK	ReadLock	Unlocked
03	TR_2	Select (Depart)	OK	ReadLock	ReadLock
04	TR_2	Select(Courses)	OK	ReadLock	ReadLock
05	TR_1	Update (Courses)	Wait	ReadLock	ReadLock
06	TR_2	Update (Depart)	OK	ReadLock	WriteLock
07	TR_2	Commit	OK	Unlocked	Unlocked
08	TR_1	Update (Courses)	OK	WriteLock	Unlocked
09	TR_1	Commit	OK	Unlocked	Unlocked

תרשים 14.30: פעולת שתי תוכניות בסביבה התומכת בנעילת קריאה וכתיבה.

הסבר:

- שורה 1 – בשלב זה שני האובייקטים נמצאים במצב לא נעול.
- שורה 2 – תוכנית יישום א' מתחילה לפעול ומבצעת את הפקודה SELECT על טבלת הקורסים. השורה של קורס C-200 ננעלת לקריאה בלבד.
- שורה 3 – תוכנית יישום ב' מתחילה לפעול ומבצעת SELECT על טבלת המחלקות. השורה של מחלקה CS ננעלת לקריאה בלבד.
- שורה 4 – תוכנית ב' מבצעת SELECT על טבלת הקורסים. מאחר והשורה המבוקשת נמצאת במצב של נעילת קריאה, תוכנית ב' מקבלת את השורה והמערכת מוסיפה את התוכנית לרשימת התנועות שביצעו נעילת קריאה לשורה.
- שורה 5 – תוכנית יישום א' מנסה לבצע את עדכון של שורת הקורס. מאחר והשורה נעולה לקריאה על ידי תוכנית יישום נוספת, לא ניתן לבצע את העדכון ותוכנית א' נכנסת למצב המתנה.
- שורה 6 – תוכנית יישום ב' מבקשת לעדכן את שורת המחלקה. השורה נמצאת במצב של נעילת קריאה והנעילה ולכן היא רק עבור תוכנית ב', המערכת מקדמת תחילה את מצב הנעילה לנעילת כתיבה כדי להבטיח שאף תוכנית אחרת לא תקבל את השורה ומבצעת את העדכון של השורה.
- שורה 7 – תוכנית יישום ב' מסיימת את פעולתה ומבצעת Commit. פעולה זו משחררת את השורה של המחלקה ממצב של נעילת כתיבה והיא חוזרת לסטטוס Unlocked.
- שורה 8 – מערכת RDBMS בודקת את רשימת נעילות הקריאה לשורת הקורס ומאחר והרשימה מצביעה על כך שרק תוכנית א' ביצעה את הנעילה, המערכת מקדמת את מצב הנעילה של שורת הקורס ממצב של נעילת קריאה למצב של נעילת כתיבה ומבצעת את העדכון.
- שורה 9 – תוכנית יישום א' מסיימת את פעולתה ומבצעת Commit. פעולה זו משחררת את נעילת הכתיבה של שורת הקורס.

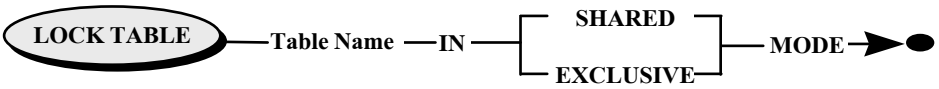
מדוגמה זו לא ניתן לראות שרמת העבודה במקביל שופרה. אם תוכניות יישום רבות יבצעו נעילות קריאה ומיעוטן יבצעו גם עדכונים, רמת העבודה במקביל תגדל באופן משמעותי כתוצאה משיפור מנגנון הנעילות.

נוסף למנגנון הפנימי המנהל את הנעילות באופן אוטומטי ונסתר מעיני מפתח היישומים, יש מספר מצבים בהם צרי לשלוט בצורה יזומה על מצב הנעילות ולשם כך מספקת מערכת RDBMS מספר אפשרויות נוספות: יכולת נעילה מפורשת של טבלה, הגדרת רמת הבידוד של התנועה ואפשרות לשלוט על הפרמטרים של הנעילה. נסקור אפשרויות אלו בהמשך.

נעילה מפורשת של טבלה (Explicit Table Locking)

יש מצבים בהם מנגנון הנעילות האוטומטי של מערכת RDBMS גורם לתקורה רבה. לדוגמה אם תוכנית יישום מסוימת מעדכנת מספר רב של שורות בטבלה כלשהי, שיטת העבודה הרגילה של נעילת כל שורה בנפרד יכולה להכביד מאוד על המערכת ולהאריך ללא צורך את זמן העיבוד. עבור מצבים מיוחדים אלה, חלק מהמערכות המסחריות תומכות בפקודה מיוחדת המאפשרת נעילה של טבלה שלמה.

תחביר הפקודה:



תרשים 14.31: תרשים תחביר לפקודה LOCK TABLE.

פקודה זו מאפשרת לתוכנית היישום לבצע נעילה באופן יזום לטבלה שלמה. ניתן לבצע נעילת כתיבה שבה המערכת תמנע גישה אל הטבלה מכל תוכנית אחרת, או נעילת קריאה שבה תורשה גישה אל הטבלה לצורך קריאת נתונים בלבד, אולם תימנע גישה מתוכניות המבקשות גם לעדכן. תוכנית היישום המבקשת לבצע את הנעילה של כל הטבלה, תוכנס למצב של המתנה אם קיימות נעילות כלשהן על הטבלה, נעילות שבוצעו על ידי תוכניות יישום אחרות. רק לאחר שחרור כל הנעילות מכל הסוגים, ניתן יהיה לנעול את כל הטבלה ולאפשר לתוכנית היישום להתחיל לפעול.

יתרון העבודה עם נעילת טבלה שלמה הוא כמובן ביעילות עיבוד רבה יותר ובמניעת האפשרות שתוכנית יישום אחרת תתפוס חלק מהשורות לצורך עדכון ותעצור את ביצוע תוכנית העדכון.

רמת הבידוד של תנועה (Isolation Level)

עקרון אי-תלות של התנועות הוא אחת מארבע התכונות של ACID. לפיו, צריכה להתקיים אי-תלות בין התנועות הפועלות בו-זמנית ואסור שתנועה תראה מצב זמני של בסיס הנתונים, מצב הנובע מהעובדה שתנועה אחרת עדיין לא סיימה לעבוד אולם כבר הספיקה לעדכן את בסיס הנתונים. ניתן גם להתייחס אל תכונה זו כאל **רמת הבידוד** שמערכת RDBMS מספקת לתנועות השונות הפועלות בו-זמנית. עקרונית, כדי לשמור על רמת הבידוד הגבוהה ביותר, על המערכת לאסור פעולה של תנועות בו-זמניות המעדכנות את בסיס הנתונים. מכיון שהמערכת אינה יודעת לצפות את ההתנהגות של תוכנית יישום מסוימת, היא צריכה לבצע נעילות כתיבה של כל השורות שתוכנית יישום מסוימת שולפת או מעדכנת, כדי להבטיח שתוך כדי ביצוע התנועה אף אחת מהשורות לא תשתנה.

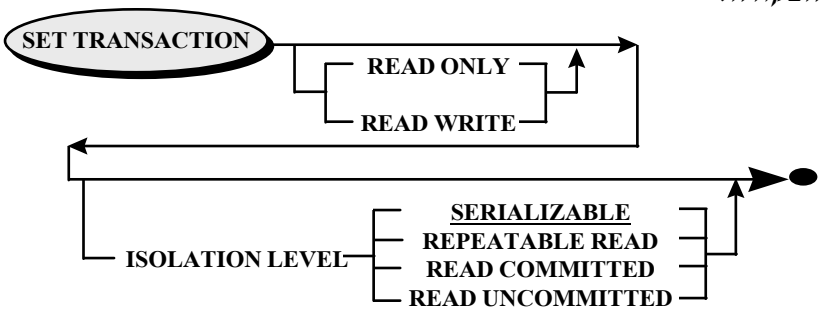
למרות ששיטה זו אפשרית, היא כמובן פוגמת ברמת העבודה במקביל שניתן להשיג, מאחר והמערכת תמנע אפילו קריאה של שורות שתוכנית אחרת כבר הספיקה לקרוא. כדי לאפשר עיבוד יעיל יותר והשגת רמת עבודה במקביל גבוהה יותר, הגדיר תקן SQL2 מונח חדש, **רמת הבידוד**. שיטה זו פותחה לראשונה על ידי חברת יבמ עבור בסיס הנתונים DB2 והפכה לאחר מכן לחלק מהתקן. התקן מגדיר ארבע רמות בידוד שונות:

- **Serializable:** זוהי רמת הבידוד הגבוהה ביותר והיא מבטיחה את רמת האמינות הגבוהה ביותר. למעשה שיטה זו מסדרת את התנועות זו אחר זו, מכיון שדי בכך שתנועה אחת תשלוף שורה כלשהי, כדי שתנועה אחרת המבקשת לשלוף את אותה שורה תכנס להמתנה עד לסיום התנועה הראשונה. רמת בידוד זו מבטיחה גם שתוך כדי ביצוע תנועה המתייחסת לטבלה כלשהי, טבלה זו לא תשתנה עד לסיום התנועה. משמעות הדבר היא שרמת בידוד זו מבטיחה שאם התנועה צריכה להפעיל מספר פעמים את אותה שאילתת SQL, היא תקבל את אותה תוצאה.
- **Repeatable Read**: זוהי רמת בידוד נמוכה יותר מהרמה הקודמת. ההבדל העיקרי בין שיטה זו והשיטה הקודמת היא שאם תנועה אחרת מוסיפה שורות חדשות או מעדכנת שורות תוך כדי פעולת התנועה, השורות החדשות או השורות המעודכנות תוצגנה לתנועה תוך כדי שליפת שורות. משמעות הדבר היא שאם תנועה מסוימת מבצעת שליפה של שורות, מבצעת עיבוד כלשהו שלהן ומפעילה מחדש את השליפה הקודמת, היא עלולה לקבל תוצאה שונה מהתוצאה המקורית, מאחר ובינתיים תנועה אחרת הוסיפה שורות חדשות או עדכנה שורות קיימות. לשורות החדשות המתווספות על ידי תנועה אחרת, מקובל לקרוא בשם Phantom Rows, מאחר והן מופיעות בשליפה השנייה למרות שהן לא הופיעו בשליפה הראשונה. אם תוכנית יישום אינה מבצעת קריאות חוזרות של אותה שאילתה, מומלץ להשתמש ברמת בידוד זו, מאחר והיא פחות מחמירה מהרמה הראשונה.
- **Read Committed:** זוהי רמת בידוד נמוכה יותר מהרמה הקודמת. שיטה זו לא תאפשר שליפת שורות הנמצאות בתהליך עדכון על ידי תנועה אחרת. שורות אלו תוצגנה רק לאחר שהתנועה השנייה סיימה את עבודתה, כלומר ביצעה Commit. יחד עם זאת, התנועה כן יכולה לראות עדכונים שבוצעו על ידי תנועות אחרות שבינתיים הספיקו לבצע Commit.

❖ **Read Uncommitted**: זוהי רמת הבידוד הנמוכה ביותר. בשיטה זו, תנועה תוכל לשלוף גם שורות שעודכנו על ידי תנועות שעדיין לא סיימו את עבודתן או על ידי תנועות שהתחילו לעבוד ובינתיים ביצעו Commit. כמובן שבשיטה זו, אם התנועה תפעיל את אותה שאילתת SQL מספר פעמים, היא עלולה לקבל תוצאות שונות בכל פעם, מאחר והיא חשופה לעדכוני הביניים של בסיס הנתונים, גם עדכונים שבוצעו על ידי תנועות שעדיין לא סיימו וגם כאלה שביצעו Rollback לאחר שעדכנו את בסיס הנתונים. בגלל החשיפה לעדכונים הנובעים מתנועות שעדיין לא ביצעו Commit, מקובל לקרוא לשיטה זו גם בשם Dirty Read.

תקן SQL2 מאפשר את הגדרת רמת הבידוד באמצעות הפקודה SET TRANSACTION.

תחביר הפקודה:



תרשים 14.32: תרשים תחביר של הפקודה SET TRANSACTION.

הפרמטר Read Only קובע שהתנועה קוראת נתונים אך אינה מעדכנת אותם. הפרמטר Read Write הוא ברירת המחדל וקובע שהתנועה שולפת ומעדכנת נתונים. הפרמטר Isolation Level מגדיר את רמת הבידוד המבוקשת. חובה לרשום פקודה זו לפני שהתנועה מתחילה לפעול.

```
1.  .
10. EXEC SQL SET TRANSACTION READ WRITE
11.                 ISOLATION LEVEL SERIALIZABLE;
12. EXEC SQL SELECT *
13.                 FROM COURSES
14.                 WHERE COURSE_ID = 'C-200';
15. IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
16. EXEC SQL UPDATE COURSES
17.                 SET POINTS = 3
18.                 WHERE COURSE_ID = 'C-200';
19. IF (SQLCA.SQLCODE < 0) GOTO ERROR_ROUTINE;
20. EXEC SQL COMMIT;
21. .
```

תוכנית זו מכילה בשורה 10 את הפקודה הקובעת שזוהי תנועה שתבצע עדכון ומבקשת את רמת הבידוד הגבוהה ביותר.

מערכת DB2 של יבמ מספקת רק שתיים מתוך ארבע רמות הבידוד: Repeatable Read ו- Read Committed (או Cursor Stability). רמות הבידוד נקבעות בזמן הידור תוכנית היישום.

קביעת פרמטרים לנעילה (Locking Parameters)

חלק ממערכות RDBMS המסחריות מאפשרות שליטה של מנהל בסיס הנתונים, ה-DBA, במספר פרמטרים המשפיעים על צורת הטיפול בנעילות. נציג כאן רק חלק מפרמטרים אלה:

- **Locksize** – פרמטר זה קובע את יעד הנעילה, כלומר איזה אובייקט יינעל. ניתן להגדיר בפרמטר זה שהנעילה תבוצע ברמת הטבלה, הדף, השורה וכד'.
- **Number of Locks** – קובע חסם עליון למספר הנעילות שתוכנית יישום תבצע.
- **Lock Timeout** – קובע חסם עליון לזמן שתנועה תמתין עד לקבלת הנעילה.
- **Lock Escalation** – קובע האם המערכת תבצע הסלמה של הנעילות. לדוגמה אם יש מספר רב של דפים נעולים בטבלה מסוימת, המערכת תנסה להסלים את הנעילה לרמת הטבלה כולה כדי לייעל את העיבוד.

גיבוי והתאוששות (Backup and Recovery)

מערכת RDBMS משרתת מטבעה מספר רב של משתמשים וחייבת להכיל מנגנונים שונים להעלאת זמינות המערכת על ידי הבטחת פעולתה הסדירה ללא הפרעות והפסקות. הזמינות מוגדרת כהסתברות שמשתמש ינסה לגשת לבסיס הנתונים ובסיס הנתונים יהיה במצב עבודה. נגדיר תחילה את המונח התאוששות.

התאוששות Recovery	***התאוששות*** **מוגדרת כתהליך שמטרתו להחזיר את בסיס הנתונים למצב תקין שבו היה לפני שהתרחשה תקלה. תהליך ההתאוששות יכול להתבצע באמצעות תוכנית שירות מיוחדת המסופקת על ידי יצרן בסיס הנתונים, או על ידי תוכנית המחזירה את בסיס הנתונים מתוך תַּוְוִך גיבוי (למשל, קלטת).**

זמן ההתאוששות (Recovery) מתקלה כלשהי הוא קריטי, כי יישומים רבים משתמשים באותו בסיס נתונים. בזמן ההתאוששות מהתקלה אין אפשרות לתת שירותים מבסיס הנתונים והדבר עלול לפגוע בפעילות הארגון.

למטרות התאוששות לאחר תקלה שומרות המערכות את כל הנתונים הדרושים כדי להתגבר על התקלה ביומן האירועים. נתונים אלה משמשים להחזרת בסיס הנתונים לנקודת זמן שבה הוא היה תקין. הנה סקירה קצרה של סוגי תקלות אפשריים:

- תוכנית יישום מסיימת בצורה לא תקינה, אך עד להפסקת עבודתה היא הספיקה לעדכן קטעים שונים בבסיס הנתונים. סיום לא תקין גורם לכך שבסיס הנתונים יהיה מעודכן באופן חלקי ולכן הוא במצב לא תקין.

- תוכנית יישום מסיימת בצורה תקינה, אבל יש בה שגיאה לוגית וכל העדכונים שביצעה אינם נכונים. לדוגמה, במקום להגדיל מונה כלשהו התוכנית הקטינה אותו.
- מערכת RDBMS עצמה מגלה מצב לא תקין ומסיימת את עבודתה. למשל, היא מגלה שמצב האינדקסים אינו תקין, או נתוני הבקרה שמנוהלים על ידי המערכת נמצאים במצב לא תקין.
- מערכת ההפעלה האחראית על כל פעולות המחשב, מגלה מצב לא תקין ומחליטה להפסיק את פעולת המחשב.
- תקלה בחומרה. לדוגמה, לא ניתן לקרוא את הנתונים מהדיסק בגלל תקלה פיסית כלשהי.
- נפילה פתאומית של המתח החשמלי.
- טעות הפעלה של מפעיל המחשב, שגורמת להפסקה בלתי צפויה של פעילות המחשב, של המערכת, או של תוכנית היישום.

מערכות RDBMS מבססות את מנגנון ההתאוששות מהתקלות על יומן האירועים והפעלה אוטומטית של תוכנית השירות Recovery Manager, המבצעת שחזור לכל התנועות שלא סיימו את פעולתן באופן תקין, כלומר לא ביצעו Commit. בנוסף, המערכות מכילות מנגנון גיבוי (Backup) שחזור לפנים (Forward Recovery) ושחזור לאחור (Backward Recovery).

גיבוי (Backup Utility)

תוכניות השירות לגיבוי מאפשרות להעתיק את בסיס הנתונים לאמצעי מגנטי כלשהו, בדרך כלל לקלטות או סרטים מגנטיים. בזמן הגיבוי, אין אפשרות לתת את השירות לתוכניות היישום ולכן מגבים את בסיס הנתונים בזמן שהוא אינו פעיל (למשל בלילה, בסופי שבוע וכד'). העתקת בסיס הנתונים יכולה להימשך דקות או שעות, הכל בהתאם לנפח הנתונים שהוא מכיל. משך זמן הגיבוי הינו שיקול חשוב בקביעת תדירות ביצוע הגיבוי.

כיום קיימות מערכות RDBMS שמאפשרות גיבוי של המערכת תוך כדי המשך מתן השירות. הגיבוי יכול להיות **מלא** ולהתייחס לבסיס הנתונים בשלמותו, או **מדורג** (Incremental), כלומר רק של השינויים בטבלאות, שנוצרו מאז הגיבוי האחרון.

בדומה לפעולת הגיבוי, יש תוכנית שירות המבצעת את השחזור (Restore) מקלטות או מסרטי הגיבוי אל הדיסקים. משך זמן השחזור שווה, ואף ארוך מזמן הגיבוי, כי דרושות פעולות מנהלה שונות (לעיתים בניית אינדקסים מחדש). שיטת השחזור המלא של בסיס הנתונים, כפי שהיא מוצגת כאן אינה מתאימה להתאוששות מהירה, הן בגלל משך הזמן שלה והן בגלל העובדה שהיא גורמת לאיבוד כל הפעילות שבוצעה בבסיס הנתונים מאז הגיבוי האחרון. זו הסיבה שמערכות RDBMS משתמשות ביומן האירועים ולעיתים גם בקבצי הגיבוי לצורך התאוששות מתקלה.

שחזור לפנים (Forward Recovery)

אם מסיבה כלשהי רוצים להחזיר את בסיס הנתונים למצב שבו הוא היה בנקודת זמן כלשהי, ניתן לשחזר אותו מקלטת או מסרט הגיבוי האחרון, ואחר כך להתחיל בתהליך של שחזור לפנים. תוכנית השירות המבצעת את השחזור לפנים, סורקת את יומן האירועים מנקודת זמן מסוימת וקדימה, ומחזירה לבסיס הנתונים את שורות "אחרי" (After Image), כלומר את מצב השורות לאחר העדכון. התוכנית תבצע שחזור זה רק עבור תנועות שהסתיימו בהצלחה, כלומר רשמו Commit.

שחזור לאחור (Backward Recovery)

תוכנית שירות זו מאפשרת לקחת את שורות "לפני" (Before Image) מיומן האירועים ולהחזיר את מצב בסיס הנתונים למצב שהיה לפני ביצוע תנועות מסוימות. תוכנית שירות זו סורקת את יומן האירועים בסדר כרונולוגי הפוך. מקובל להשתמש בשחזור לאחור כאשר רוצים לבטל את ההשפעות של תנועה מסוימת. לדוגמה, גילוי שגיאה בתנועה, הגורמת לכך שכל העדכונים שהיא ביצעה אינם נכונים.

סיכום

בפרק זה סקרנו שני שירותים חשובים שמערכת RDBMS מספקת, תמיכה בתנועות עדכון ותמיכה בעדכון בו-זמני. בלעדי שני שירותים אלה, היה קיים קושי רציני בהפעלת תוכניות יישום הפועלות בסביבה מרובת משתמשים והשומרות על אמינות בסיס הנתונים. כפי שהסברנו, שירותים אלה לא היו כלולים במערכות ניהול הקבצים והדבר שימש זרז לפיתוח טכנולוגיית בסיסי נתונים.

התמיכה בתנועות מאפשרת למערכת RDBMS להבטיח את תקינות בסיס הנתונים באותם מיקרים בהם מתבצעות מספר פעולות עדכון ברצף לוגי כלשהו, או במקרה שפקודה בודדת מעדכנת מספר רב של שורות בבסיס הנתונים. כדי לתמוך בעיבוד תנועות, משתמשת מערכת RDBMS במנגנון ייחודי הקרוי **יומן אירועים**. מנגנון זה רושם על קובץ מיוחד כל אירוע של עדכון בסיס הנתונים אשר יכול לשמש, במידת הצורך, לביטול פעולות העדכון שכבר בוצעו.

התמיכה בעדכון בו-זמני מבוססת על מנגנון אחר של המערכת הקרוי מנגנון **הנעילות**. מנגנון זה עוקב אחר כל פעולת עדכון של אובייקט כלשהו בבסיס הנתונים ומבטיח שרק תוכנית יישום אחת תוכל לעדכן אותו. כל התוכניות האחרות המבקשות לעדכן את אותו אובייקט מוכנסות למצב של המתנה עד לשחרור הנעילה. מנגנון זה פועל ברוב המיקרים בצורה נסתרת מעיני מפתחי היישומים, אולם קיומו קריטי לשמירת אמינות הנתונים בסביבה מרובת משתמשים.

שאלות חזרה ותרגילים

שאלות חזרה

1. הסבר את המושג תנועה (Transaction) ומדוע יש חשיבות רבה בתמיכה בעיבוד תנועות. סקור את הבעיות העיקריות שעיבוד תנועות מקוון מציב בפני מערכת RDBMS. תן דוגמה לתנועה המעדכנת מספר קבצים מיישום המוכר לך.
2. מנה לפחות שתי סיבות מדוע תנועה יכולה להסתיים בכישלון.
3. הסבר מה חשיבות קובץ יומן האירועים ואיזה שירותים הוא מספק. חווה דעתך על אורך התקופה שבה כדאי לשמור יומן אירועים ומדוע.
4. הסבר מהי נעילה דו-שלבית ומה חשיבותה.
5. הסבר את המושג "התאוששות" ואיזה תוכניות שירות נדרשות לביצוע תהליך זה.
6. הסבר מהו "שחזור לאחור" ומהו "שחזור לפנים", ומתי נדרש כל אחד מהם.
7. קרא את המשפטים הבאים וענה בכן/לא:

א. מערכות FMS אינן תומכות בעיבוד תנועות, כי אין בהן הפקודות Commit ו- Rollback.

ב. רשומה נעולה יכולה להשתחרר רק על ידי פקודה מפורשת בתוכנית היישום.

ג. נעילה ברמת רשומה טובה יותר מנעילה ברמת דף או גוש.

ד. מנגנון אבטחת הנתונים העיקרי של מערכת RDBMS הוא מנגנון ההצפנה.

תרגילים

1. תוכנית ביצעה Commit בגמר התנועה. המשתמש גילה שהוא עדכן את הנתונים הלא נכונים. הסבר מה ניתן לעשות במצב זה כדי למחוק את העדכונים שבוצעו.
2. נתון מצב הנעילות הבא:

א. משתמש Dan נעל אובייקטים R_1, R_2 ו- R_3 וממתין לאובייקט R_4.

ב. משתמש Ron נעל אובייקטים R_4, R_5 ו- R_6 וממתין לאובייקט R_8.

ג. משתמש Eyal נעל אובייקטים R_8, R_9 וממתין לאובייקט R_{10}.

ד. משתמש Zvi נעל אובייקטים R_{10}, R_{11} וממתין לאובייקט R_{12}.

הצג את גרף הנעילות של האובייקטים והראה כיצד בא לביטוי העובדה שזהו מצב של נעילה ללא מוצא.

3. נתונות שלוש תנועות T_1, T_2 ו- T_3 שיכולות להיות מופעלות בו-זמנית על ידי משתמשים שונים:

א. T_1 מבצעת פעולה A=A*2 על עמודה בשורה.

ב. T_2 מבצעת פעולה A=A+5 על עמודה בשורה.

ג. T_3 מבצעת פעולה A=A-3 על עמודה בשורה.

אם בתחילת יום העבודה הערך של שדה A ברשומה הוא 100 ושלוש התנועות מופעלות פעם אחת בלבד, הצג את כל הערכים האפשריים ש-A יכול לקבל בתום פעולת התנועות.

4. עדכן את יומן האירועים המוצג בתרשים 14.8 לאחר שיקרו אירועים אלה:

א. המשתמש Zvi הפעיל את תנועה PO-10 בתאריך 5.9.1999 בשעה 8:30 ממסוף TER-30.

ב. בשעה 8:31 התנועה ביטלה את רשומת הספק שמספרו 584 מטבלת הספקים.

ג. המשתמש Eli הפעיל את תנועה MLAY-07 בתאריך 5.9.1999 בשעה 8:32 ממסוף TER-24.

ד. המשתמש Zvi סיים את התוכנית בהצלחה בשעה 8:33.

ה. בשעה 8:34 הוסיף Eli רשומה חדשה לטבלת המלאי עם מפתח 5582.

ו. בשעה 8:35 הוסיף Eli רשומה חדשה לטבלת המלאי עם מפתח 6415.

במצב זה, התוכנית MLAY-07 כושלת ("עפה") כתוצאה מתקלה. הסבר איזה תנועות תבוטלנה כתוצאה מהפעלת גלילה לאחור (Rollback).

פרק 15

קטלוג המערכת (System Catalog)

1. מבוא
2. קטלוג המערכת כחלק ממערכת RDBMS
3. תכולת קטלוג המערכת
4. שאילתות על הקטלוג
5. קטלוג המערכת ותקן SQL
6. סיכום
7. שאלות חזרה ותרגילים

מבוא

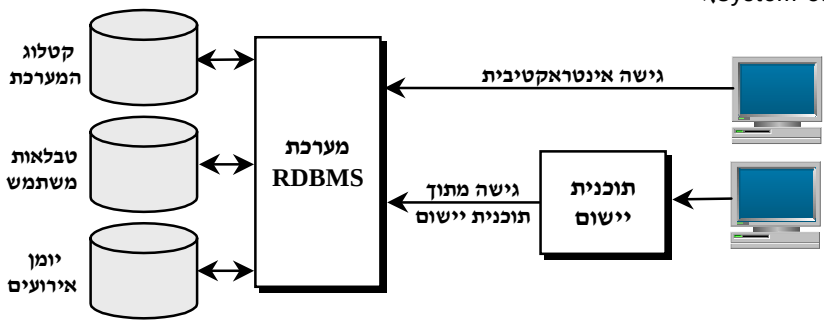
מערכת RDBMS צריכה לנהל מידע רב אודות האובייקטים השונים של בסיס הנתונים: איזה טבלאות יש בבסיס הנתונים, מהן העמודות בכל טבלה, מהו טיפוס הנתונים של כל עמודה, מהן ברירות המחדל של עמודה ומהם הערכים המותרים לעמודה, מהו המפתח העיקרי של כל הטבלה ומהם המפתחות הזרים שלה, מהם כללי האמינות שעל בסיס הנתונים לאכוף, מי הם המשתמשים המורשים ואיזה פעולות מותר להם לבצע על הטבלאות השונות, מהם האינדקסים של כל טבלה, מידע סטטיסטי על כל טבלה (כמו מספר השורות שלה) המשמש את מעבד פקודות SQL ועוד. מקובל לקרוא לכל המידע הזה בשם **מידע על הנתונים** (Meta Data), כלומר מידע המתאר את הנתונים ולא את תוכנם. התוכן של בסיס הנתונים מנוהל כמובן בתוך הטבלאות עצמן.

בפרק זה

- ❖ נסקור את קטלוג המערכת, שהינו אוסף טבלאות המנוהל על ידי מערכת RDBMS ומשרת בעיקר אותה.
- ❖ נציג את אוסף הטבלאות העיקריות המרכיבות את קטלוג המערכת.
- ❖ נראה כי ניתן לגשת לקטלוג המערכת באמצעות פקודות SELECT רגילות של שפת SQL, לצורך תחקור והכרת מבנה בסיס הנתונים.
- ❖ נציג בקצרה את אוסף הטבלאות המדומות שתקן SQL2 מגדיר עבור קטלוג המערכת.

קטלוג המערכת כחלק ממערכת RDBMS

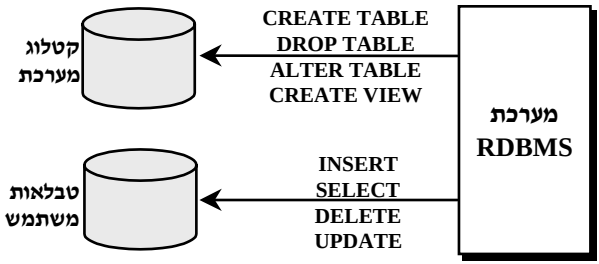
עד כאן התמקדנו בעיקר בשני סוגי טבלאות שמערכת RDBMS מנהלת: **טבלאות המשתמשים** (User Tables) **ויומן האירועים** (Log File). יומן האירועים מנוהל על ידי המערכת כדי לתמוך ביכולת שחזור והתאוששות מהירה של טבלאות הנתונים של המשתמש במקרה של הפסקה לא תקינה של תנועה או של בסיס הנתונים עצמו. נוסיף עכשיו קטגוריה נוספת של טבלאות המנוהלות על ידי מערכת RDBMS: **קטלוג המערכת** (System Catalog).



תרשים 15.1: מקומו של קטלוג המערכת.

קטלוג המערכת מנוהל בראש וראשונה עבור מערכת RDBMS עצמה. יחד עם זאת, המידע המנוהל בתוך הקטלוג הוא מידע רב ערך וניתן להעמייד אותו גם לרשות מנהל בסיס הנתונים והמשתמשים. לדוגמה, משתמש חדש יכול לדעת איזה עמודות מנוהלות במסגרת טבלת הציונים, מהו המפתח העיקרי של הטבלה וכד'.

אחת התכונות המעניינות של מערכות RDBMS היא שקטלוג המערכת הוא בעצמו אוסף של טבלאות, ולכן ניתן לגשת אליהן באמצעות פקודות SQL רגילות. כמובן שטבלאות אלו, המשמשות את המערכת RDBMS לתפקודה השוטף, אינן מתעדכנות בדרך רגילה על ידי המשתמשים אלא בדרך כלל על ידי מנהל בסיס הנתונים או על ידי המערכת עצמה. העדכון של טבלאות אלו אינו מתבצע על ידי פקודות SQL כגון INSERT או UPDATE, אלא על ידי פקודות DDL של השפה, כגון CREATE TABLE, DROP TABLE, CREATE INDEX וכד'.



תרשים 15.2: פקודות שונות מעדכנות את טבלאות המשתמש ואת טבלאות המערכת.

תכולת קטלוג המערכת

כל יצרן של מערכת RDBMS פיתח את קטלוג המערכת בצורה שונה ועל פי תפישתו, לאור העובדה שתקן SQL1 לא התייחס לנושא זה. בהמשך נשתמש במספר דוגמאות הלקוחות ממערכת DB2 של יבמ כדי להדגים את תוכן הקטלוג. הקטלוג של DB2 מורכב ממספר עשרות טבלאות, ששמותיהן מתחילים בקידומת קבועה, Sys. נסקור כמה מהטבלאות המרכיבות את קטלוג המערכת:

- **טבלת הטבלאות (SYSTABLES):** טבלה זו מכילה שורה אחת עבור כל טבלת משתמש, טבלה בסיסית או מדומה. לכל טבלה מנוהלים נתונים כגון שם הטבלה (NAME), שם המשתמש שהגדיר את הטבלה (CREATEDBY), שם המשתמש שלו שייכת הטבלה (CREATOR), מספר העמודות בטבלה (COLCOUNT) וכד'.

Name	CreatedBy	Creator	Colcount	.....
Courses	RazH	RazH	5	
Students	RazH	RazH	3	
Departments	RazH	RazH	3	
Grades	RazH	RazH	6	
.	.	.	.	
.	.	.	.	

תרשים 15.3: מבנה הטבלה SYSTABLES.

❖ **טבלת העמודות (SYSCOLUMNS):** טבלה זו מכילה שורה אחת עבור כל עמודה הנמצאת בטבלת משתמש כלשהי. לכל עמודה מנוהלים נתונים כגון שם העמודה (NAME), שם הטבלה שבה העמודה נמצאת (TBNAME), טיפוס הנתונים של העמודה (COLTYPE), האם העמודה יכולה להכיל ערכים חסרים (NULLS), אורך העמודה (LENGTH), המספר הסידורי של העמודה בתוך הטבלה (COLNO) וכד'.

Name	Tbname	Tbcreator	Coltype	Nulls	Length	Colno	.....
Course_Id	Courses	RazH	Char	N	10	1	
Course_Name	Courses	RazH	Char	Y	20	2	
Type	Courses	RazH	Char	Y	5	3	
Points	Courses	RazH	Integer	Y		4	
Department_Id	Courses	RazH	Char	N	2	5	
Student_Id	Students	RazH	Integer	N		1	
Name	Students	RazH	Char	N	20	2	
City	Students	RazH	Char	Y	15	3	
.	.	.	.	.	.	.	

תרשים 15.4: מבנה הטבלה SYSCOLUMNS.

❖ **טבלת האינדקסים (SYSINDEXES):** טבלה זו מכילה את רשימת כל האינדקסים המנוהלים על ידי המערכת. לכל אינדקס מנוהלים נתונים כגון שם האינדקס (NAME), שם המשתמש שהגדיר את האינדקס (CREATEDBY) ושם הטבלה שעבורה מוגדר האינדקס (TBNAME).

❖ **טבלת המשתמשים (SYSUSERS):** טבלה זו מכילה את רשימת כל המשתמשים המורשים לגשת לטבלה כלשהי בבסיס הנתונים.

❖ **טבלת זכויות הגישה של המשתמשים (SYSUSERAUTH):** טבלה זו מנהלת את כל זכויות הגישה שהוענקו למשתמש מסוים תוך ציון מי הוא המשתמש שהעניק את זכות הגישה. בכל שורה מנוהלים נתונים כגון שם המשתמש שהעניק את זכות הגישה (GRANTOR), שם המשתמש שהוענקה לו זכות גישה (GRANTEE), התאריך שבו הוענקו הזכויות (DATEGRANTED), השעה שבה הוענקו הזכויות (TIMEGRANTED), צורת הענקת זכות הגישה (GRANTEETYPE) וכד'.

❖ **טבלת זכויות הגישה לטבלאות (SYSTABAUTH):** טבלה זו מנהלת לכל טבלת משתמש את רשימת המשתמשים שיש להם גישה לטבלה זו וכן איזו זכות גישה הוענקה להם לטבלה (קריאה, הוספה וכד').

❖ **טבלת זכות הפעלת תוכנית גישה (SYSPLANAUTH):** טבלה זו מנהלת את רשימת המשתמשים שמותר להם להפעיל תוכנית גישה לבסיס הנתונים, תוכנית שנוצרה כתוצאה מתהליך ההידור של תוכנית יישום כלשהי.

❖ **טבלת הטבלאות המדומות (SYSVIEWS):** לכל טבלה מדומה, בנוסף לשם המנוהל בטבלת הטבלאות, יש טבלה נוספת המכילה את השאילתה המגדירה את הטבלה המדומה. לכל טבלה מדומה מנוהלים נתונים כגון שם הטבלה המדומה (NAME), שם

המשתמש שהגדיר את הטבלה המדומה (CREATOR), ציון האם הטבלה המדומה מכילה אופציית CHECK ועמודה המכילה את הגדרת הטבלה המדומה עצמה.

- **טבלת הקשרים (SYSRELS):** טבלה זו מנהלת את כל הקשרים בין הטבלאות, קשרים הנוצרים כתוצאה מהגדרת המפתחות הזרים בזמן הגדרת כל טבלה. טבלה זו נדרשת לצורך ניהול Referential Integrity. לכל קשר מנוהלים נתונים כגון שם טבלת האב (TBNAME), שם טבלת הבן (REFTABLE), שם הקשר (RELNAME), שם המשתמש שהגדיר את הקשר (CREATOR), מספר העמודות בתוך המפתח הזר (COLCOUNT), כלל ביטול השורות בטבלת הבן (DELETERULE).

Tbname	Reftable	Tbcreator	Relname	Colcount	Deleterule	.....
Departments	Courses	RazH	Dep_Courses	1	R	
Courses	Grades	RazH	Cour_Grades	1	R	
Students	Grades	RazH	Stud_Grades	1	R	
.	.	.		.	.	

תרשים 15.5: מבנה הטבלה SYSRELS.

שאילתות על הקטלוג

כפי שאמרנו, קטלוג המערכת הוא אוסף של טבלאות פנימיות המנוהלות על ידי מערכת RDBMS. אלו הן טבלאות רגילות ולכן ניתן לשלוף נתונים מתוכן באמצעות פקודות SELECT רגילות. נציג מספר דוגמאות לשאילתות על הקטלוג:

דוגמה א' : הצג את רשימת כל טבלאות המשתמשים המתחילות באות D.

```
SELECT *
FROM   SYSTABLES
WHERE NAME LIKE 'D%'
```

Name	CreatedBy	Creator	Colcount	.....
Departments	RazH	RazH	3	

דוגמה ב' : הצג את שמות העמודות ושמות הטבלאות עבור כל העמודות מסוג Integer בטבלאות שהוגדרו על ידי משתמש ששמו RazH.

```
SELECT C.NAME, T.TBNAME
FROM   SYSTABLES T, SYSCOLUMNS C
WHERE COLTYPE = 'Integer' AND
      T.CREATEDBY = 'RazH' AND
      C.TBNAME = T.NAME
```

Name	Tbname
Points	Courses
Student_Id	Students
Student_Id	Students
Grade	Departments
Grade_Sem	Departments

דוגמה ג' : הצג את רשימת הטבלאות שיש בהן יותר משלוש עמודות שטיפוס הנתונים שלהם הוא מסוג Char.

```
SELECT     TBNAME, COUNT (DISTINCT NAME AS COL_COUNT
FROM        SYSCOLUMNS
WHERE      COLTYPE = 'CHAR'
GROUP BY  TBNAME
HAVING     COUNT (DISTINCT NAME) > 3
```

Tbname	Col_Count
Courses	4

קטלוג המערכת ותקן SQL

תקן SQL1 לא התייחס כלל אל קטלוג המערכת או אל השאלה כיצד יצרן מערכת RDBMS צריך לממש אותו. מסיבה זו, כל יצרן מימש את הקטלוג בצורה שונה. מאחר וקטלוג המערכת נמצא בלב המערכת, לא קל כיום לבוא ולשנות את המבנה שלו, מאחר ודבר זה דורש כתיבה מחדש של חלקים נרחבים של המערכת. בגלל הקשיים לשנות את מבנה קטלוג המערכת ובגלל ההכרה בחשיבות המידע המנוהל בטבלאות המערכת, ניגש תקן SQL2 לבעיה זו בצורה שונה. התקן אינו כופה על יצרני המערכת לשנות את מבנה הקטלוג אלא רק מגדיר אוסף של טבלאות מדומות (Views) שיצרן המערכת צריך לתמוך בהן. התקן קורא לאוסף טבלאות מדומות אלו בשם Information Schema, או סכמת מידע. חשיבות התמיכה באוסף מוגדר זה של טבלאות מדומות נובעת מהעובדה שיצרני צד שלישי המפתחים כלים שונים שצריכים לפעול עם בסיס הנתונים – מחוללי שאילתות, סביבות פיתוח דור רביעי – יכולים להתבסס על מידע המוצג בצורה אחידה, גם אם קטלוג המערכת בנוי בצורה שונה ומורכב מטבלאות שונות. להלן מספר דוגמאות לטבלאות מדומות המוגדרות על ידי תקן SQL2 :

- TABLES – טבלת הטבלאות (ריכוז הטבלאות).
- COLUMNS – טבלת העמודות.
- VIEWS – ריכוז הטבלאות המדומות.
- VIEW_TABLE_USAGE – טבלה המכילה שורה עבור כל טבלה המופיעה בהגדרת טבלה מדומה כלשהי.
- VIEW_COLUMN_USAGE – טבלה המכילה שורה אחת עבור כל עמודה המופיעה בהגדרה של טבלה מדומה כלשהי.
- USERS – טבלת המשתמשים.
- TABLE_PRIVILEGES – טבלה המכילה שורה עבור כל זכות גישה המוענקת לטבלה כלשהי.
- COLUMN_PRIVILEGES – טבלה המכילה שורה עבור כל זכות גישה המוענקת לעמודה כלשהי.

- USAGE_PRIVILEGES – טבלה המכילה שורה עבור כל זכות שימוש המוענקת למשתמש.
- DOMAINS – טבלת מרחבי הערכים.
- DOMAIN_CONSTRAINTS – טבלת האילוצים המגדירים את מרחבי הערכים.
- TABLE_CONSTRAINTS – טבלה המכילה שורה עבור כל אילוץ המופיע בהגדרת טבלה.
- CHECK_CONSTRAINTS – טבלה המכילה שורה עבור כל אילוץ המוגדר על ידי משפט CHECK בעת הגדרת טבלה.
- REFERENTIAL_CONSTRAINTS – טבלת המפתחות הזרים והאילוצים של הקשרים בין הטבלאות.

סיכום

קטלוג המערכת הינו אוסף של טבלאות פנימיות המנוהלות על ידי מערכת ה- RDBMS לצרכים שלה. המידע המנוהל בטבלאות אלו הוא המידע על כל האובייקטים של בסיס הנתונים. מכיון שמידע זה מנוהל בטבלאות רגילות, ניתן לגשת אליהן באמצעות פקודות SQL רגילות.

מאחר ונושא מבנה קטלוג המערכת לא הופיע בתקן SQL1 נוצרה מידה רבה של אי תאימות בין מבנה הקטלוג כפי שהוא מיושם על ידי מערכות RDBMS מסחריות שונות. תקן SQL2 מגדיר סדרה של טבלאות מדומות על טבלאות הקטלוג, כך שכלי צד שלישי יוכלו לגשת בצורה אחידה אל טבלאות הקטלוג למרות השוני בצורת המימוש.

שאלות חזרה ותרגילים

שאלות חזרה

1. מהי המטרה של הקטלוג, ומה תפקידו?
2. מי משתמש בקטלוג הנתונים ומתי?
3. מדוע משתמשים אינם יכולים לעדכן באופן ישיר את קטלוג הנתונים?
4. מדוע יש חשיבות לכך שקטלוג המערכת יהיה בעצמו אוסף של טבלאות, ולא קובץ מיוחד המוכר על ידי מערכת RDBMS בלבד?

תרגילים

1. נתונה הסכימה הטבלאית הבאה:

- **יצרן** (מספר יצרן, שם יצרן, עיר, כתובת, ארץ)
- **מחשב נייח** (שם מודל, מספר יצרן, מהירות מעבד, גודל דיסק, גודל זיכרון, גודל מסך, מחיר בדולרים)
- **מחשב נייד** (שם מודל, מספר יצרן, מהירות מעבד, גודל דיסק, סוג מסך, גודל זיכרון, מחיר בדולרים)
- **מדפסות** (שם מודל, מספר יצרן, סוג מדפסת, מהירות הדפסה, צורת חיבור, מחיר בדולרים)

טען נתונים בשתי טבלאות של קטלוג המערכת, טבלת הטבלאות וטבלת העמודות, המתאימות למודל הטבלאי הזה.

2. בנה שאילתת SQL שתציג את רשימת כל העמודות בבסיס הנתונים שהן מטיפוס נומרי שלם ושם העמודה מתחיל ב- S או ב-R.

חלק ד'

נושאים מתקדמים

חלק ד' של הספר מורכב משלושה פרקים העוסקים במספר נושאים מתקדמים.

- **פרק 16 – בסיסי נתונים בסביבה מבוזרת:** פרק זה בוחן את טכנולוגיית בסיסי הנתונים על רקע המגמות העדכניות ביותר בתחומי המחשוב ודן בנושאי ביזור, הטרוגניות בניהול הנתונים ומודל שרת/לקוח, המאפשר לחלק את משימת המחשוב בין מחשבים שונים. הפרק מציג את רעיון התקשורת בין משימות כבסיס לביזור משימות מחשוב ומראה כיצד הרעיון הורחב לעבודה בסביבה מבוזרת. מאחר וסביבות המחשוב המודרניות הן לרוב מבוזרות והטרוגניות, עוסק הפרק במספר סוגיות של עבודה שיתופית (Interoperability) בין מערכות של יצרנים שונים בהתבסס על תקנים שונים.

- **פרק 17 – בסיסי נתונים מבוזרים:** פרק זה סוקר בקצרה את נושא מערכות ניהול בסיסי הנתונים המבוזרים ומציג את המורכבות הנובעת מעצם רעיון ביזור הנתונים. הפרק מגדיר את המושג של בסיס נתונים מבוזר, בסיס נתונים המנוהל במספר שרתים שונים המחוברים ביניהם ברשת תקשורת כלשהי, סוקר את היתרונות והחסרונות של מערכות מבוזרות, מסביר את ההבדל בין מערכות מבוזרות הומוגניות והטרוגניות, מתאר את הארכיטקטורה האפשרית של מערכת מבוזרת וסוקר את רעיון פיצול טבלה אחת למספר מקטעים שונים. למרות המורכבות הרבה והאתגרים הרבים העומדים בפני מפתחי מערכות מבוזרות, ולמרות שכיום לא קיימת אף מערכת מסחרית שניתן לומר שהיא מבוזרת באופן מלא, המחקר של נושאים אלה הביא למספר רב של שיפורים ושיכלולים במערכות הקיימות.

- **פרק 18 – מערכות מונחות אובייקטים**: פרק זה סוקר את הבעיות והחסרונות של מערכות RDBMS בבואן להתמודד עם סוגי נתונים בלתי שיגרתיים שיישומים חדשים, וביניהם יישומי אינטרנט ומסחר אלקטרוני, מנהלים מפות, תמונות, שרטוטים, טקסט, וידאו, וקול. כתוצאה מהקשיים של המודל הטבלאי להתמודד עם סוגי נתונים אלה וכתוצאה מההתפתחויות המהירות שחלו בתחום שפות תכנות מונחות אובייקטים, החל להתפתח מודל חדש, מודל האובייקטים. הפרק סוקר את מושגי היסוד של מודל האובייקטים – אובייקט, שירותים (Methods), כימוס (Encapsulation), קבוצות אובייקטים (Object Classes), היררכיות הורשה, ריבוי צורות (Polymorphism) ועוד.

מערכות Object Oriented DBMS שהתפתחו כתוצאה של תפישת מודל האובייקטים לא זכו להצלחה גדולה בעולם עיבוד הנתונים והתנועות אולם זכו להצלחה רבה בסדרה שלמה של יישומים ייחודים. במקביל להופעת מודל האובייקטים החלו גם יצרני מערכות RDBMS לחפש אחר דרכים לשפר את המודל הטבלאי. חלק משיפורים אלה כבר מצא את דרכו אל תקן SQL3 הנמצא בעת כתיבת שורות אלו במצב טיוטה. מקובל לקרוא למערכות המבוססות על ההרחבות והשיפורים המוצעים בתקן בשם מערכות Object Relational DBMS. הפרק סוקר את המושגים העיקריים במערכות אלו ובעיקר את הרעיון המרכזי הקרוי **טיפוס נתונים מופשט** (Abstract Data Type).

פרק 16

בסיסי נתונים בסביבה מבוזרת

1. מבוא
2. המגמות במחשוב הארגוני
3. תקשורת בין משימות – הבסיס לביזור
4. ארכיטקטורות מבוזרות
5. מרכיבי תוכנית יישום
6. מודל מסגרת לביזור מרכיבי תוכנית היישום
7. רמות התמיכה בתנועות הפועלות בסביבה מבוזרת
8. פרוטוקול Two Phase Commit לניהול תנועות מבוזרות
9. ארכיטקטורות לקישוריות בסביבה הטרוגנית
10. ניהול העתקים (Copy Management)
11. סיכום

מבוא

בפרק זה נבחן מספר סוגיות הקשורות לניהול בסיסי נתונים בסביבה מבוזרת, הן הומוגנית והן הטרוגנית. ההתקדמות הטכנולוגית בתחום בסיסי נתונים טבלאיים הביאה להתפתחות מקבילה בתחום ניהול בסיסי נתונים בסביבות מבוזרות.

משך שנים רבות ניסו יצרני מערכות RDBMS להוסיף יכולות ביזור למערכות שלהם ולבנות מערכת מבוזרת: Distributed RDBMS או בקיצור DRDBMS. מערכות אלו מוצגות בפרק הבא. מערכת DRDBMS צריכה לאפשר ביזור שקוף לחלוטין של הנתונים ברחבי הארגון. התברר שהאתגר מורכב מאוד, ולא פחות חשוב – הוא אינו עונה על המציאות שהלכה והתהוותה. ארגון יכול להפעיל מספר רב של מערכות RDBMS שונות משל יצרנים שונים ועל כן אינו יכול להגיע לסטנדרטיזציה מוחלטת ביניהן, והוא גם לא יכול להשתמש במערכת מסוג אחד בלבד.

על רקע זה החלה להתפתח ההבנה שהפתרון **אינו בביזור** בסיס נתונים, אלא במציאת דרך של **שילוב ושיתוף פעולה** (Interoperability) בין בסיסי נתונים שונים. נקודת המוצא של מגמה זו היא שארגון מפעיל אוסף של בסיסי נתונים (Federated Database) שהם גם מבוזרים וגם של יצרנים שונים וגם מסוגים שונים; כלומר, הם אינם מבוססים בהכרח על המודל הטבלאי.

קורא המעוניין להרחיב את ידיעותיו בנושאים אלה מופנה לספרו של Hackathorn [HR93] ולספרה של בוצ'ינסקי בגירסתו העברית [BB95].

בפרק זה

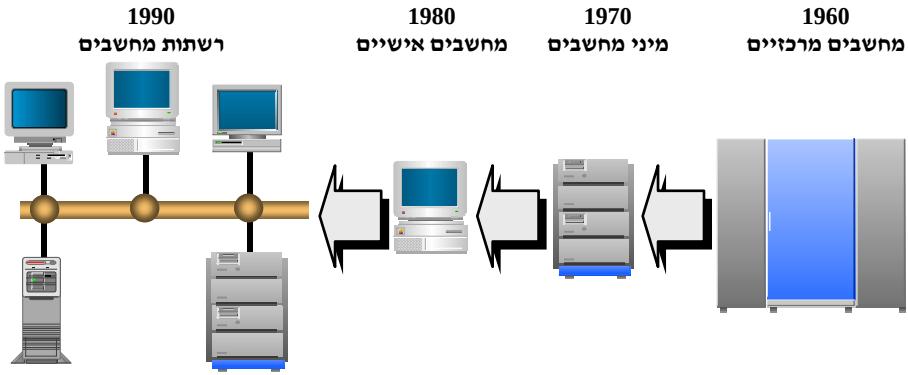
- נסקור את המגמות העיקריות במחשוב הארגוני. מגמות אלו הניעו את הארגון מעבודה עם מערכות מידע מרכזיות לעבודה בסביבה עתירת פלטפורמות מחשוב שונות, מערכות הפעלה שונות ובסיסי נתונים של יצרנים שונים. לסביבה זו אנו קוראים **סביבת המחשוב המבוזרת וההטרוגנית**.
- נסביר את העקרונות של ביזור משימות מחשוב בין מחשבים שונים. נסקור את מנגנון IPC המאפשר תקשורת והעברת הודעות בין משימות הפועלות באותו מחשב תחת אותה מערכת הפעלה. נציג גם את מנגון RPC שמרחיב את היכולות האלו לסביבה מבוזרת שבה המשימות פועלות על מחשבים שונים.
- נציג את הארכיטקטורות השונות לביזור משימות ואת מודל שרת/לקוח על תצורותיו השונות.
- נסביר את המבנה העקרוני של תוכנית יישום מקוונת ונראה שניתן להתייחס אל התוכנית כמורכבת ממספר רכיבים ייעודיים. כל רכיב מטפל בהיבט אחד של הפעילות ולפיכך נמצא רכיב המטפל בלוגיקה העסקית, רכיב המטפל בלוגיקת הנתונים, רכיב המטפל בתצוגת הנתונים על המסך ועוד. ההבחנה בין רכיבים אלה היא הבסיס למודל שרת/לקוח ולאפשרויות הביזור השונות.

- בהתבסס על הרכיבים השונים של תוכנית יישום מקוונת, נסקור את אפשרויות הביזור השונות של רכיבים אלה. נראה שניתן לקבוע שחלק מהרכיבים פועלים על מחשב אחד ואילו חלק אחר פועל על מחשב אחר. נציג ארכיטקטורות שרת/לקוח דו-שכבתיות, תלת-שכבתיות ורב-שכבתיות המבוססות על שרתי יישום ושרתי בסיסי נתונים.
- נסביר את רמות התמיכה השונות שמערכת RDBMS יכולה לספק לטיפול בתנועות הפועלות בסביבה מבוזרת. הרמות מתחילות מהרמה הפשוטה ביותר של בקשה מרוחקת שמכילה פקודת SQL בודדת הפונה לבסיס נתונים מרוחק אחד ועד לרמה הגבוהה ביותר שבה תנועה אחת יכולה להכיל מספר פקודות SQL כאשר כל פקודת SQL יכולה לפנות בו-זמנית לטבלאות מבוזרות.
- נסקור את הפרוטוקול Two Phase Commit המאפשר ביצוע תנועות מבוזרות, שמשתתפות בהן מספר מערכות RDBMS.
- נסביר את שלוש הארכיטקטורות העיקריות לעבודה שיתופית (Interoperability), המאפשרות לתוכנית יישום אחת לפנות לנתונים המנוהלים בבסיסי נתונים מרוחקים ושונים. נציג את ארכיטקטורת הממשק האחיד עליה מבוססת תפישת ODBC של מיקרוסופט, את ארכיטקטורת שער היציאה האחיד ואת ארכיטקטורת הפרוטוקול האחיד.
- נסקור טכניקות שונות לניהול העתקים (Copy Management), טכניקות המאפשרות לנהל מספר עותקים של הנתונים במספר צמתים ברשת המבוזרת.

מגמות במחשוב ארגוני

ההתפתחות המהירה של טכנולוגיית החומרה, התוכנה, בסיסי הנתונים ורשתות תקשורת הנתונים בשנים האחרונות, הביאה לשינוי משמעותי בנוף מערכות המחשוב הארגוניות. אנו נוכחים במגמה מתמדת של הגירה ממודל המחשוב ששלט בשנות ה-60 והיה מבוסס על מחשב מרכזי ועל בסיסי נתונים מרכזיים ומסופים "טיפשים", אל מודל המחשוב המבוזר של שנות ה-90. מודל זה מבוסס על שילוב של שרתים ותחנות עבודה חכמות ורבות עוצמה, על ממשקים גרפיים, על בסיסי נתונים טבלאיים ועל רשתות תקשורת מהירות.

תרשים 16.1 מציג את המהפיכה שעולם המחשוב עבר משנות ה-60 ועד לשנות ה-90. המעבר המהיר הזה (יחסית לתחומים אחרים) החל ממודל המבוסס על מחשבים מרכזיים גדולים שמחוברים אליהם מסופים טפשים ברשת תקשורת, עבר למודל המבוסס על מיני מחשבים ומחשבים אישיים וכעת אנו בתקופת רשתות תקשורת מקומיות (LAN) ורשתות מרחביות (WAN) המחברות יחדיו שרתים ותחנות עבודה רבות. מודל זה הולך ומפנה את מקומו למודל המחשוב של האינטרנט המוביל בכניסה לשנות ה-2000. האינטרנט הינה רשת תקשורת ענקית הפרוסה על פני כל כדור הארץ ומקשרת מיליוני מחשבים ומשתמשים.

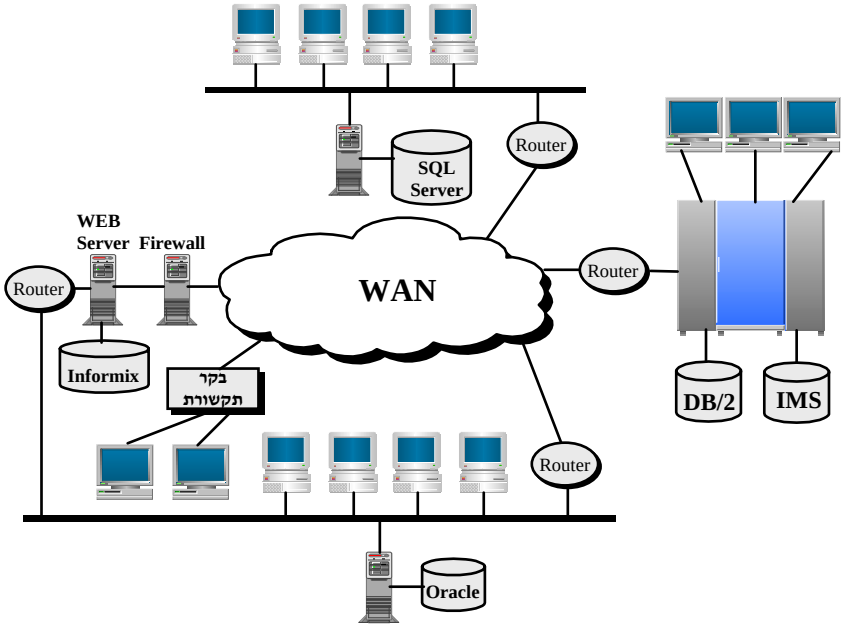


תרשים 16.1: התפתחות מודל המחשוב.

בתחילה השתמשו ברשת התקשורת בעיקר לחיבור מסופים אל מחשב מרכזי. ברשת התקשורת המקומית השתמשו בעיקר לשיתוף משאבי חומרה שונים כמו מדפסות ודיסקים. כיום אנו נמצאים בתהליך המצעיד אותנו צעד אחד קדימה: **ביזור היישומים והנתונים**. כפי שנראה בהמשך, ביזור הנתונים מוסיף רמה נוספת למורכבות שבניהול נתונים. ביצוע תנועות המשתמשות בנתונים, המאוחסנים במספר שרתים שונים, המחוברים ביניהם ברשת תקשורת, אינו פשוט כלל.

מודל המחשוב המבוזר הביא לכך שארגונים עברו ממערכות מידע המבוססות על ניהול נתונים במחשב מרכזי אחד למצב שבו הנתונים מנוהלים במספר רב של שרתים שונים הקשורים ביניהם ברשת תקשורת. המצב כיום הוא שרוב הארגונים מפעילים מערכות מידע הפועלות בסביבת מחשוב **מבוזרת והטרוגנית,** שבה פועלים שרתים שונים המבוססים על מערכות הפעלה שונות. ביניהם ניתן למצוא שרתים מרכזיים מתוצרת חברת יבמ מבוססי מערכת הפעלה OS/390, שרתי Sun, HP, יבמ ו-Compaq המבוססים על מערכת הפעלה Unix; שרתי AS/400 המבוססים על מערכת הפעלה OS/400 ושרתי Intel המבוססים על מערכת הפעלה Windows NT ו-Linux. שרתים אלה מנהלים את הנתונים במספר מערכות DBMS שונות, כמו IMS, Adabas, DB/2 או IDMS במחשבים המרכזיים; Oracle, Informix, DB/2 או Sybase על שרתי Unix; SQL Server על שרתי NT; SQL/400 על מחשבי AS/400 ועוד. הימים שבהם כל מערכות המידע של הארגון פעלו בסביבה **מרכזית והומוגנית** שבה כל היישומים פעלו על מחשב אחד, עם מערכת הפעלה אחת ועם בסיס נתונים אחד, חלפו ולא ישובו יותר.

סביבת המחשוב המודרנית, שמקובל לכנותה **סביבת מחשוב מבוזרת הטרוגנית**, היא כיום סביבת המחשוב איתה צריך להתמודד כמעט כל ארגון. הסיבות למעבר לסביבת מחשוב זו הן רבות ומגוונות כגון רכישת חבילות תוכנה הפועלות על פלטפורמות מסוימות, תהליך של Downsizing שרוב הארגונים החלו, רכישות ומיזוגים של חברות וצורך להתמודד עם מספר מערכות מידע שונות. סביבה זו יצרה לחצים רבים על יצרני מערכות RDBMS לסטנדרטיזציה וגיבוש תקנים שונים שיאפשרו שיתוף נתונים המנוהלים בפלטפורמות שונות. עם הזמן החלו להופיע תקנים כגון DRDA של יבמ, ODBC של מיקרוסופט וגם תקנים של יצרני תוכנה אחרים, שמטרתם היתה לטפל במצב מורכב זה.



תרשים 16.2: תצורה של מערך מחשוב מבוזר והטרוגני.

תקשורת בין משימות – הבסיס לביזור

לפני שנדון בביזור משימות בין מחשבים שונים, נתייחס בקצרה למנגנון התקשורת בין משימות הפועלות באותו מחשב. מערכות ההפעלה המודרניות תומכות **בריבוי מעבדים** (Multiprocessors) בארכיטקטורות שונות (כמו SMP, MPP, NUMA) **ובריבוי משימות** (Multitasking) **הפועלות במקביל** (Concurrent Processes). מערכות הפעלה אלו מאפשרות למספר משימות לפעול במקביל ולתקשר ביניהן, כלומר להעביר מסרים והודעות ביניהן.

העברת מסרים בין משימות

מקובל לקרוא למנגנון המאפשר העברת מסרים בין משימות בשם Inter Process Communication Facility או בקיצור **IPC**. מנגנונים אלה משתמשים בשטחי זיכרון משותפים (Shared Memory) ובשיטות מיוחדות המסמנות למשימה מסוימת על כוונת משימה אחרת להעביר לה מסר.

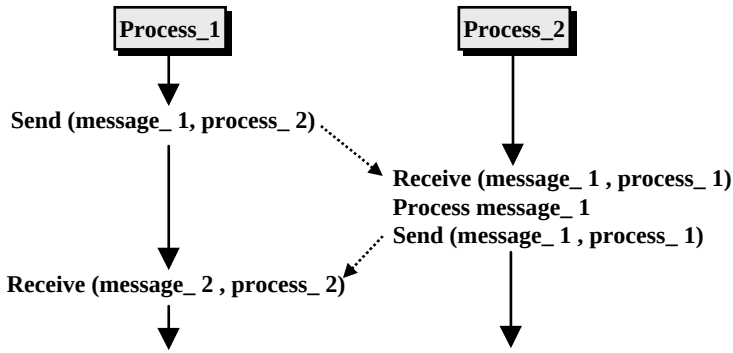
מנגנון IPC הוא מנגנון דינמי המאפשר להעביר מסרים בין שתי משימות, המשימה הקוראת והמשימה הנקראת, הפועלות בו-זמנית ושלא עברו תהליך של קישור (Link) מוקדם ביניהן. מנגנון IPC מבוסס על שתי פקודות פשוטות יחסית, אחת לשיגור המסר והשנייה – לקליטת התשובה. לשיגור מסר, המשימה הקוראת תבצע פקודה כזו:

send (message, to_process_name)

פקודה זו שולחת מסר שתוכנו נמצא במשתנה message, אל משימה ששמה to_process_name. לקבלת התשובה, המשימה הנקראת תבצע פקודה כזו:

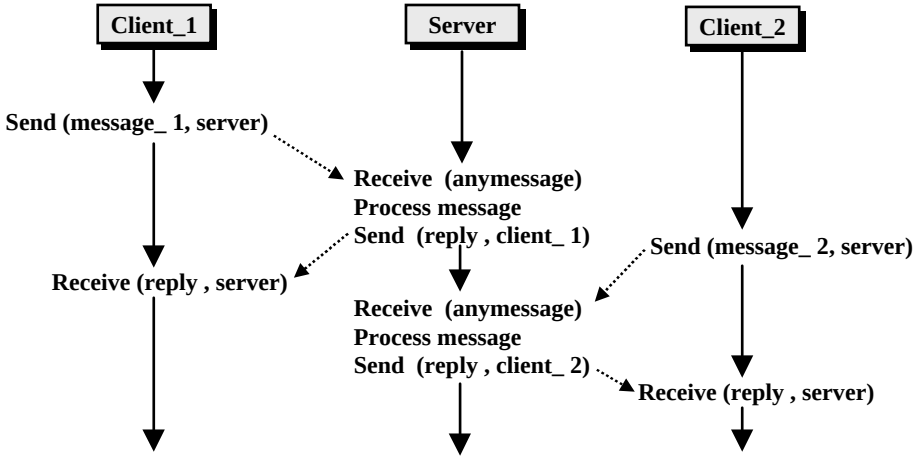
recieve (message, from_process_name)

פקודה זו מקבלת מסר שהתוכן שלו נמצא במשתנה message והיא תחזיר את התשובה למשימה ששמה from_process_name.



תרשים 16.3: העברת מסרים בין שתי משימות.

תרשים 16.3 מדגים כיצד משימה Process_1 מעבירה מסר למשימה Process_2 באמצעות הפקודה Send. המשימה Process_2 מקבלת את המסר, מבצעת עיבוד כלשהו ושולחת תשובה למשימה Process_1. למשימה הקוראת נקרא **משימת לקוח** (Client Process) ולמשימה הנקראת, זו הנותנת את השירות, נקרא **משימת שרת** (Server Process). כדי לאפשר את התקשורת בין המשימות, משימת הלקוח מכירה את שם משימת השרת ומשימת השרת מכירה את שם משימת הלקוח. משימת השרת נמצאת תמיד במצב של המתנה למתן שירות.



תרשים 16.4: משימת שרת המשרתת מספר משימות לקוח.

מה קורה כאשר משימת שרת אחת צריכה לשרת מספר רב של משימות לקוח שונות, אך אינה מכירה את שמות משימות הלקוח מראש. במצב זה, משימת השרת פועלת בשיטה של Port, שיטה המתבססת על מעין תיבת דאר אליה נשלחות כל הבקשות.

בתרשים 16.4, משימה Client_1 קוראת למשימת השרת ומעבירה לה מסר. משימת השרת רושמת לעצמה שהמשימה הקוראת היא Client_1 ועם גמר העיבוד מחזירה את התשובה למשימה זו. משימה Client_2 פונה גם היא אל משימת השרת ובצורה דומה מתבצע העיבוד ומוחזרת התשובה. כל משימות הלקוח שולחות את המסרים למשימת השרת, השומרת אותן בתיבת דאר יחד עם הזהות של השולח. משימת השרת משרתת את משימות הלקוח על פי סדר הופעתן בתיבת הדאר.

סוגי סינכרון של משימות

קיימות שתי שיטות סינכרון בין משימת הלקוח והשרת. השיטה הראשונה מבצעת **חסימה** של המשימה הקוראת עד לקבלת התשובה (Blocking Protocol), כלומר משימת הלקוח נכנסת למצב של המתנה עד לקבלת התשובה. שיטה זו נקראת שיטת תקשורת **סינכרונית** (Synchronous). שיטה שנייה היא **ללא חסימה** (Nonblocking Protocol) ומאפשרת המשך עבודה של משימת הלקוח. כך, משימת הלקוח ממשיכה לפעול במקביל למשימת השרת. שיטה זו נקראת שיטת תקשורת **אסינכרונית** (Asynchronous). בשיטה זו, משימת הלקוח צריכה לבדוק מזמן לזמן אם התקבלה תשובה.

סוגי קישור בין משימות

כל עוד המשימות פועלות תחת אותה מערכת הפעלה ובאותו מרחב זיכרון, מנגנון העברת המסרים פשוט יחסית. נבדוק מה קורה כאשר המשימות פועלות במחשבים שונים. מצב זה הוא מורכב יותר, מאחר ולא ניתן להשתמש בזיכרון משותף להעברת המסרים בין המשימות ויש להשתמש בפרוטוקולים לתקשורת המאפשרים למחשבים שונים הפועלים במערכות הפעלה שונות להעביר נתונים ביניהם.

מכיון שהמחשבים קשורים ביניהם ברשת תקשורת כלשהי, המסרים צריכים לעבור דרך רשת תקשורת זו ולהתבסס על פרוטוקול התקשורת (TCP/IP, NetBIOS, IPX, LU 6.2, ואחרים). מקובל להבחין בין שני סוגי התקשרות: **מוכוונות קישור** (Connection Oriented), או **אינן מוכוונות קישור** (Connectionless Oriented). בשיטה הראשונה, נוצר קשר קבוע בין המקור לבין היעד וכל המסרים זורמים באותו מסלול. בשיטה השנייה, המסר מפורק ונארז במספר חבילות (Packets). על כל חבילה נרשמת הכתובת המלאה של היעד וכל חבילה מוצאת את דרכה אל היעד. שיטה זו ידועה גם בשם של **מיתוג מנות** (Packet Switching). בצד השני מתבצע תהליך של פתיחת החבילות והרכבת המסר מחדש.

איתור מיקום משימת השרת

בעיה חדשה שמתעוררת בסביבה מבוזרת היא כיצד מאתרים את המחשב שבו פועלת משימת השרת. כל עוד משימת השרת פועלת באותו מרחב זיכרון כמו משימת הלקוח, כל שנדרש הוא להבטיח שמשימת השרת תהיה בעלת שם חד-ערכי, כך שכל משימת לקוח תוכל לפנות למשימת השרת ומערכת ההפעלה תוכל לזהותה בצורה חד-משמעית. ברגע שהמשימות פועלות במחשבים שונים, בעיית זיהוי משימת השרת מורכבת יותר. במצב זה, זיהוי המשימה צריך להתבצע על ידי צמד שמות: שם המשימה ושם המחשב שבו היא פועלת. לדוגמה:

check_budget, server_inv@xxx.com

שם המשימה check_budget והיא נמצאת במחשב שכתובתו היא server_inv@xxx.com. כתובת המחשב היא כתובת IP אותה ניתן לרשום בצורה של כתובת אינטרנט, כלומר מחשב ששמו הלוגי הוא serv_inv בתוך מרחב כתובות xxx.com. כדי לפנות למשימת שרת, משימת לקוח תבצע את הפקודה הבאה:

send (message, check_budget, server_inv@xxx.com)

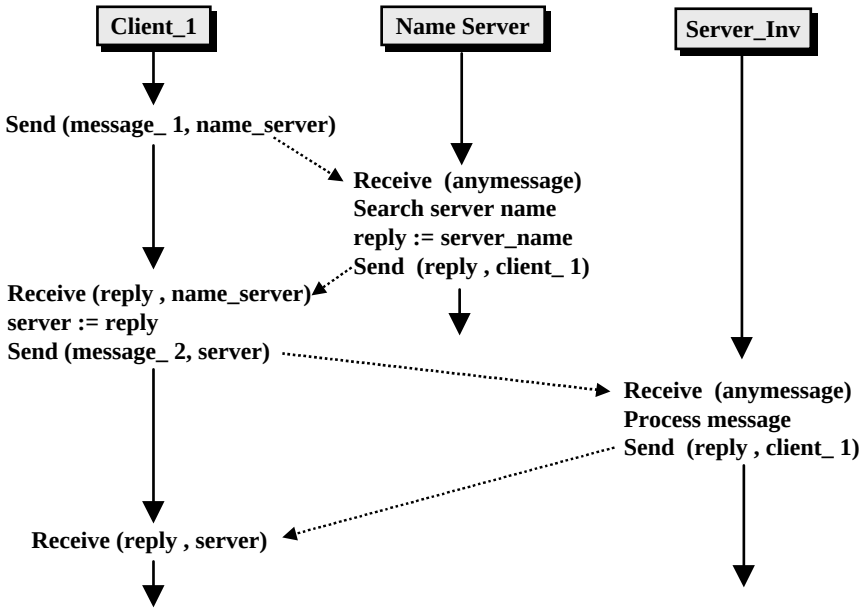
שיטה זו, שבה שם משימת השרת נרשם באופן מפורש במשימת הלקוח, נקראת **שיטת קישור שמות סטטית** (Static Binding). החסרון העיקרי של שיטת קישור השמות הסטטית הוא חוסר הגמישות. כל מי שכותב משימת לקוח חייב לדעת במדויק את שם המחשב שבו משימת השרת פועלת. הזזה של משימת השרת למחשב אחר תחייב לבצע שינויים בכל משימות הלקוח.

כדי להתגבר על בעיה זו, פותחה **שיטת קישור שמות דינמית** (Dynamic Binding). שיטה זו אינה מחייבת ידיעה מראש של שם המחשב שבו פועלת משימת השרת והיא מבצעת תהליך דינמי של **הגדרת שם המחשב** (Name Resolution). קיימות מספר שיטות קישור דינמיות לאיתור שם המחשב שבו פועלת משימת השרת:

- **קישור דינמי דרך משתנה מוסכם (Dynamic Binding via Convention)**: בשיטה זו רושמים את שם המחשב במשתנה מוסכם מראש, Environment Variable. בכל מחשב שבו פועלת משימת לקוח, רושמים את שם השרת במשתנה גלובלי כלשהו ומשימת הלקוח לוקחת את שם המחשב מתוך משתנה מוסכם זה. כאשר מבקשים להזיז את משימת השרת למחשב אחר, יש לשנות את המשתנה בכל המחשבים בהם פועלות משימות לקוח. למרות פשטות השיטה, החסרון המרכזי שלה הוא הצורך לעדכן את שם מחשב השרת בכל המחשבים בהם פועלות משימות לקוח.
- **קישור דינמי דרך שרת שמות (Dynamic Binding via Name Server)**: שיטה נפוצה יותר היא שימוש בשרת שמות, מחשב בעל שם ידוע מראש, שמריץ משימה מיוחדת של איתור שם המחשב שבו פועלת משימת שרת כלשהי. משימה מיוחדת זו מנהלת טבלה המכילה את שם המחשב שבה פועלת משימת שרת. משימת לקוח פונה תחילה לשרת השמות ומפעילה את השירות של קבלת שם המחשב שבו פועלת

משימת השרת. שרת השמות מבצע סריקה בטבלה המכילה את כתובות המחשבים בהן פועלות משימות השרת השונות ומחזיר למשימת הלקוח את שם המחשב.

תרשים 16.5 מראה כיצד פועלת השיטה המבוססת של שרת שמות. משימת הלקוח Client_1 פונה קודם כל לשרת השמות Name_Server ורושמת בתוך המשתנה message_1 את שם משימת השרת המבוקשת. שרת השמות מקבל את הבקשה, מבצע חיפוש בתוך הטבלה ומחזיר בתוך משתנה reply את שם המחשב שבו פועלת משימת השרת. משימת הלקוח מעבירה את שם השרת אל המשתנה server ומכאן ואילך משתמשת במשתנה זה בכל הפניות למשימת השרת.



תרשים 16.5: שימוש בשרת שמות לאיתור שם מחשב שבו פועלת משימת שרת.

איתור שם המחשב שבו פועלת משימת השרת הוא רק הצעד הראשון בביזור משימות בין מחשבים שונים. בנוסף, יש לטפל במיגוון גדול מאוד של בעיות המתעוררות בסביבה המבוזרת:

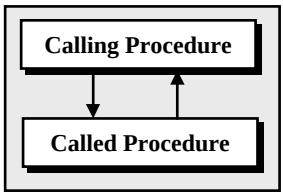
- מחשבים שונים מייצגים בצורה שונה את הנתונים (ASCII, EBCIDIC) ולכן יש צורך בביצוע המרת פורמטים של נתונים.
- טיפול במצבי תקלה והודעות שגיאה הנובעים מהעובדה ששני מחשבים שונים מתקשרים אחד עם השני. לדוגמה, יש צורך בטיפול מיוחד עבור מצב שבו משימת השרת אינה עונה לאחר פרק זמן מסוים (Timeout).
- טיפול בשידור חוזר של המסרים במידה ויש שיבושים בקווי התקשורת. ניתן להשתמש בשיטות של הוספת סיביות מיוחדות, CRC (Cyclic Redundancy Check), כדי לוודא שהמסר הגיע בצורה תקינה מצד לצד.

- ❖ טיפול בבעיות של אבטחת מידע על ידי זיהוי המשימה הקוראת, הצפנת המסר וכד'.
- ❖ פירוק המסר למספר חבילות (Packets) על פי דרישות הפרוטוקולים השונים לתקשורת ובניית המסר מחדש בצד השני.

הטיפול המלא בכל ההיבטים האלה הוא מורכב, מייגע מאוד ודורש רמת מיומנות גבוהה של מפתחי התוכנה. כדי להתגבר על מורכבות זו, פותח מנגנון **הקריאה לפרוצדורה מרוחקת**, מנגנון RPC, או בשמו המלא Remote Procedure Call. מנגנון זה מתבסס על תשתית מנגנון IPC ומטפל באופן שקוף בכל המורכבות של העברת מסרים בין משימות מחשוב הפועלות במחשבים שונים תחת מערכות הפעלה שונות, הכתובות בשפות תכנות שונות וקשורות ביניהן ברשתות תקשורת הפועלות בפרוטוקולי תקשורת שונים.

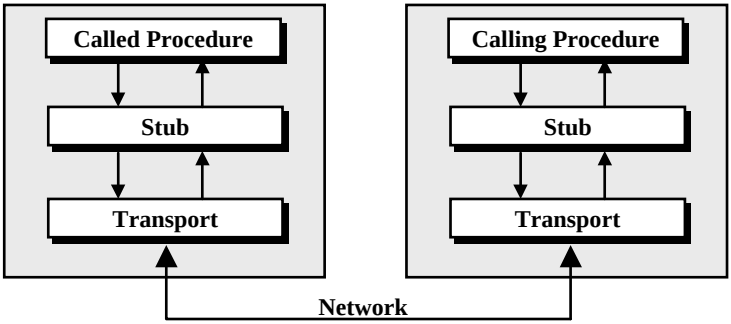
העברת מסרים בין משימות

הרעיון של הפעלת משימות שונות והעברת מסרים ביניהן אינו חדש וקיים שנים רבות. שיטה זו מאפשרת לתוכנית, שנקרא לה לצורך העניין **הפרוצדורה הקוראת** (Calling Procedure), לקרוא לתוכנית אחרת, לה נקרא **הפרוצדורה הנקראת** (Called Procedure). הפרוצדורה הקוראת פונה לפרוצדורה הנקראת באמצעות הפקודה Call, פקודה המאפשרת העברת פרמטרים בין שתי הפרוצדורות. הפרוצדורה הקוראת והפרוצדורה הנקראת יכולות להיות כתובות בשפות תכנות שונות וכל אחת מהן עוברת הידור בנפרד. לאחר ההידור של שתיהן, הן עוברות תהליך של קישור (Bind/Link) שבו הפרוצדורה הנקראת נקשרת אל הפרוצדורה הקוראת ונוצר קובץ ריצה (Object) אחד. בדרך כלל עותק של הפרוצדורה הנקראת נטען לזיכרון יחד עם הפרוצדורה הקוראת והפיקוח עובר אליה עם ביצוע הפקודה Call. שיטה זו קלה יחסית למימוש, מאחר ושתי הפרוצדורות פועלות באותו המחשב.



תרשים 16.6: קריאה לפרוצדורה מקומית.

מנגנון RPC מרחיב מודל פשוט זה למודל מורכב יותר המטפל בהפעלה של פרוצדורות מרוחקות. הוא מטפל בהפעלת הפרוצדורה המרוחקת ובכל הבעיות של המרות צורות ייצוג הנתונים, מצבי שגיאה, הצפנת נתונים במידת הצורך, פירוק המסר לחבילות על פי פרוטוקול התקשורת וכד'.



תרשים 16.7: קריאה לפרוצדורה מרוחקת.

להגדרת הממשק בין הפרוצדורות הקוראת והנקראת לבין מנגנון RPC, מספק יצרן RPC **שפה להגדרת הממשק** (Interface Definition Language) ובקיצור – IDL. פקודות השפה עוברות הידור נפרד בשני המחשבים ומייצרות את הממשק הנקרא Stub. ממשק זה מאפשר לפרוצדורה הקוראת ולפרוצדורה הנקראת להפעיל את מנגנון RPC שנמצא בספריה RPC Library בשני המחשבים השונים. הממשק שנוצר לאחר ההידור עובר תהליך קישור (Link) עם הפרוצדורה הקוראת, ובנפרד – עם הפרוצדורה הנקראת. בצורה זו ניתן לטפל בכל התקשורת בין שתי פרוצדורות הפועלות על מחשבים שונים, ולגרום לכך שמנקודת מבט מפתח התוכנה קריאה לפרוצדורה מרוחקת נראית בדיוק כמו קריאה לפרוצדורה מקומית.

מנגנוני RPC הראשונים פותחו בין השאר על ידי חברת Sun (Transport Independent RPC), על ידי חברת HP ואחרות. ארגון OSF (Open Software Foundation) אימץ בשלב מאוחר יותר את המנגנון והפך אותו לחלק מתקן DCE (Distributed Computing Environment).

נשתמש בדוגמה של הפעלת פרוצדורה מרוחקת לבדיקת תקציב. להפעלת הפרוצדורה המרוחקת, נרשום פקודה כגון:

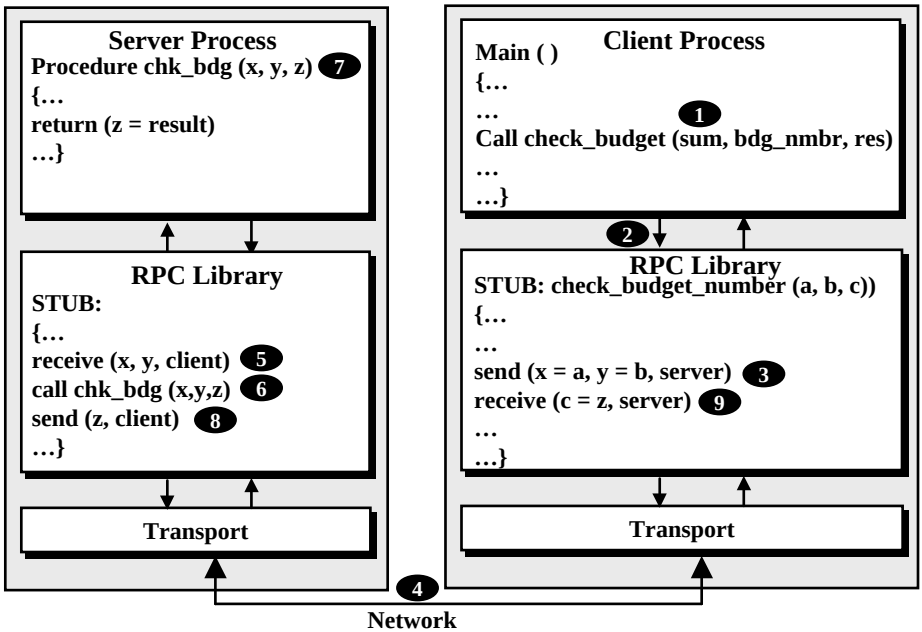
call check_budget (sum, budget_number, result)

הפרמטרים sum ו- budget_number מכילים את הסכום ואת סעיף התקציב, והפרמטר result מכיל את התשובה.

נסקור בקצרה את הצעדים להפעלת הפרוצדורה המרוחקת, כמודגם בתרשים 16.8.

- **צעד 1** – הפרוצדורה הקוראת הפועלת על מחשב כלשהו מבצעת את הקריאה לפרוצדורה check_budget.
- **צעד 2** – קריאה זו מועברת דרך הממשק אל מנגנון RPC.
- **צעד 3** – מנגנון RPC מפעיל את מנגנון IPC לתקשורת בין משימות. נשים לב שהפנייה היא לשרת בשם server. כפי שהסברנו, כדי שלא ניצור קישור סטטי עם שם המחשב שבו פועלת הפרוצדורה הנקראת, ניתן בשלב זה להפעיל את הקישור הדינמי, למשל על ידי פנייה לשרת השמות. קטע זה אינו מופיע בתרשים.

- **צעד 4** – המסר מועבר באמצעות פרוטוקול התקשורת אל מחשב השרת.
- **צעד 5** – במחשב השרת פועל מנגנון RPC שנמצא כל הזמן במצב המתנה לבקשות ממשימות אחרות.
- **צעד 6** – עם קבלת הבקשה, הפנייה מועברת אל הפרוצדורה המבוקשת עם הפרמטרים המתאימים.
- **צעד 7** – הפרוצדורה הנקראת מופעלת, מבצעת את המשימה ומחזירה את התוצאה בפרמטר המתאים.
- **צעד 8** – מנגנון RPC במחשב השרת מפעיל את מנגנון IPC לתקשורת בין משימות והתוצאה מועברת חזרה אל המחשב הקורא.
- **צעד 9** – מנגנון RPC במחשב הלקוח מקבל את המסר, מפענח אותו ומחזיר את התוצאה לפרוצדורה הקוראת.



תרשים 16.8: הפעלת פרוצדורה מרוחקת.

מנקודת מבט מפתח התוכנה, אין כל הבדל בין הפעלה פרוצדורה נקראת באותו מחשב שבה פועלת הפרוצדורה הקוראת לבין הפעלתה במחשב אחר. כמובן שמאחורי הקלעים, יש הבדל משמעותי בין שני מצבים אלה.

במשך השנים התפתח נושא מנגנוני התקשורת בין משימות בסביבה מבוזרת והפך לאחד המרכיבים העיקריים של סביבה זו. מקובל לקרוא למרכיבים אלה בשם הכוללני של **תוכנת ביניים** או **תוכנת תווך** (Middleware). מרכיבים אלה משמשים כדבק המאפשר להדביק ביחד את המרכיבים השונים של היישום המבוזר. בין מנגנונים אלה נציין את מנגנוני העברה אמינה של מסרים (Message Oriented Middleware), מוניטור תנועות מבוזר (Distributed Transaction Monitor) ואחרים. בצורה זו או אחרת, מנגנונים אלה משתמשים ביישומים מתוחכמים של רעיון ה- RPC תוך הרחבתו ושיפורו.

ארכיטקטורות מבוזרות

תהליך הביזור החל מהמצב הראשוני שבו כל מרכיבי התוכנה פעלו במחשב אחד והתפתח למצב שבו מרכיבי תוכנה שונים בוצעו במחשבים שונים, תוך שיתוף פעולה בין המחשבים השונים. למודל המחשוב, המבוסס על רעיון ביצוע מרכיבים שונים במחשבים שונים מקובל לקרוא **מודל שרת/לקוח** (Client/Server).

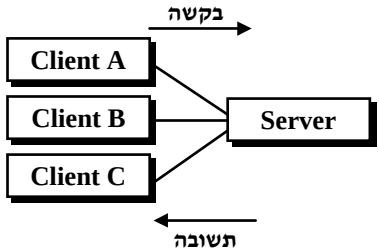
באופן בסיסי, מודל זה מתייחס למחשבי שרת המבצעים מספר משימות מרכזיות ולמחשבי לקוח המבצעים משימות אחרות. מחשבי הלקוח פונים למחשב השרת לקבלת שירותים שונים. בבסיס המודל הזה עומד רעיון חלוקת נטל העיבוד בין מחשבים שונים, דבר המאפשר בניית מערכת מחשוב מאוזנת יותר. בתפישה ריכוזית, הוספת משתמש נוסף גורמת לתוספת עבודה למחשב המרכזי. בתפישת שרת/לקוח הוספת משתמש נוסף מוסיפה מחשב חדש למערכת ולכן לא כל הנטל על המחשב המרכזי.

מכיון שמחשבים שונים משתפים פעולה ביניהם לביצוע משימה, מקובל לקרוא לשיטת עיבוד זו גם **עיבוד שיתופי** (Cooperative Processing). מודל המחשוב השיתופי הינו כללי יותר, מאחר והוא מניח שכל מחשב יכול לבצע משימות שונות, כלומר, גם לבקש שירותים ממחשבים אחרים וגם לתת שירותים למחשבים אחרים. לעומתו, מודל שרת/לקוח מבחין בין מחשב הלקוח המבקש שירותים ממחשבים אחרים, אולם אינו מספק בעצמו שירותים למחשבים אחרים.

נסקור בקצרה ארכיטקטורות שונות המגדירות את סגנון התקשורת בין הפרוצדורות. לארכיטקטורות שבהן פועלות פרוצדורות המבקשות שירות ופרוצדורות הנותנות שירות והנמצאות במחשבים שונים מקובל לקרוא בשם **ארכיטקטורות שרת/לקוח** (Client/Server Architectures).

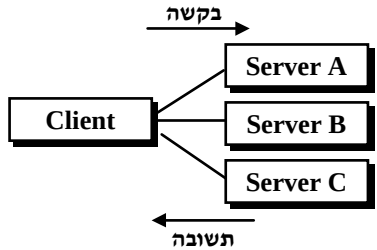
כפי שנראה קיימות מספר תצורות לקשרי גומלין בין לקוחות ושרתים. המשותף בין כולם הוא ההפרדה הברורה בין לקוח ושרת, כלומר אין מצב בו לקוח יכול לפנות ללקוח אלא רק לשרת. לעומת זאת, שרת יכול לפנות לשרת. בנוסף לארכיטקטורות אלו קיימות גם ארכיטקטורות שיתופיות (Peer to Peer) שבהן אין אבחנה בין שרת ללקוח.

❖ **שרת בודד ולקוחות מרובים (Single Server/Multiple Clients):** זוהי הארכיטקטורה הפשוטה ביותר ובה שרת אחד נותן שירותים למספר רב של לקוחות שונים. מקובל לקרוא לארכיטקטורה זו גם בשם Server Centric.



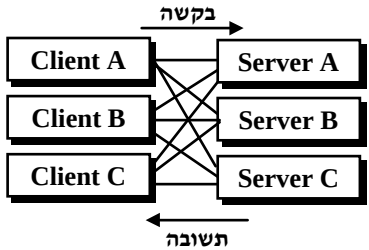
תרשים 16.9: שרת אחד המשרת מספר לקוחות.

❖ **לקוח בודד ושרתים מרובים (Single Client/Multiple Servers):** בארכיטקטורה זו לקוח אחד מקבל שירותים ממספר רב של שרתים. מקובל לקורא לארכיטקטורה זו גם בשם Client Centric.



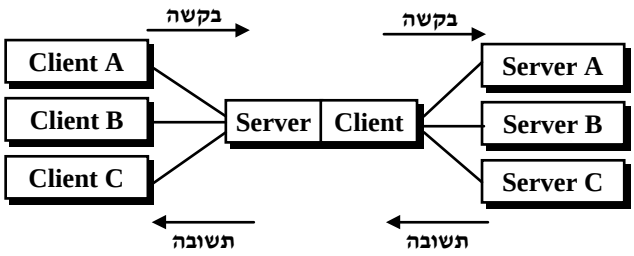
תרשים 16.10: לקוח אחד המקבל שירות ממספר שרתים.

❖ **ריבוי לקוחות וריבוי שרתים (Multiple Clients/Multiple Servers):** בארכיטקטורה זו מספר לקוחות יכולים לפנות למספר שרתים ואותו שרת יכול לשרת מספר רב של לקוחות.



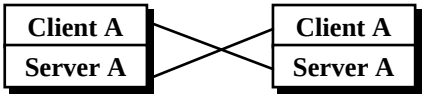
תרשים 16.11: ריבוי לקוחות וריבוי שרתים.

❖ **ארכיטקטורות רב שכבתיות (Multi Tier Client/Server):** ארכיטקטורה זו מרחיבה את המודל ומאפשרת לשרת להפוך ללקוח של שרת אחר.



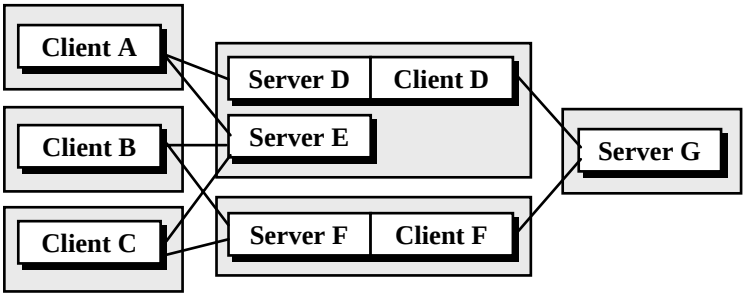
תרשים 16.12: ארכיטקטורה רב-שכבתית.

❖ **ארכיטקטורות שוויוניות (Peer to Peer):** בארכיטקטורה זו, כל צומת יכול לפנות לכל צומת כלומר אין כאן אבחנה בין לקוח לשרת, מאחר וכל אחד יכול לבצע את שתי המשימות, גם לבקש שירות וגם לתת שירות.



תרשים 16.13: ארכיטקטורה שוויונית.

נשים לב לכך שעד כאן לא התייחסנו למחשב שבו פועלת פרוצדורת הלקוח ופרוצדורת השרת. שתי הפרוצדורות יכולות לפעול באותו מחשב, ולקיים ביניהן משטר של לקוח ושרת ויכולות לפעול במחשבים שונים ולקיים את אותו משטר. מקובל לכן להבחין בין שכבות התוכנה (Software Tiers) לבין שכבות החומרה (Hardware Tiers).



תרשים 16.14: ריבוי שכבות תוכנה וחומרה.

תרשים 16.14 מדגים את האבחנה בין שכבות התוכנה לשכבות החומרה. למשל שרתים D ו-E פועלים באותו מחשב, ושרת F פועל על מחשב נפרד. שרת D ושרת F פונים לשרת G הפועל על מחשב נפרד.

מרכיבי תוכנית יישום

לאחר שהצגנו את הרעיון שניתן לבזר את משימות המחשוב בין משימות הפועלות במחשבים שונים והראינו שקיימות ארכיטקטורות שונות להגדרת יחסי הגומלין בין השרת ללקוח, נסתכל עכשיו על תוכנית יישום טיפוסית ונבדוק כיצד ניתן לבזר אותה.

נתחיל בהצגת המבנה העקרוני של תוכנית יישום מקוונת, תוכנית המציגה נתונים על מסך, קולטת נתונים מהמשתמש, בודקת את תקינות הנתונים, מעבדת אותם ומעדכנת את בסיס הנתונים. מתוך התבוננות על תוכנית מסוג זה, ניתן להבחין שהיא מורכבת ממספר מרכיבים, כלומר מספר קטעי תוכנה המטפלים במטלות מוגדרות וייעודיות:

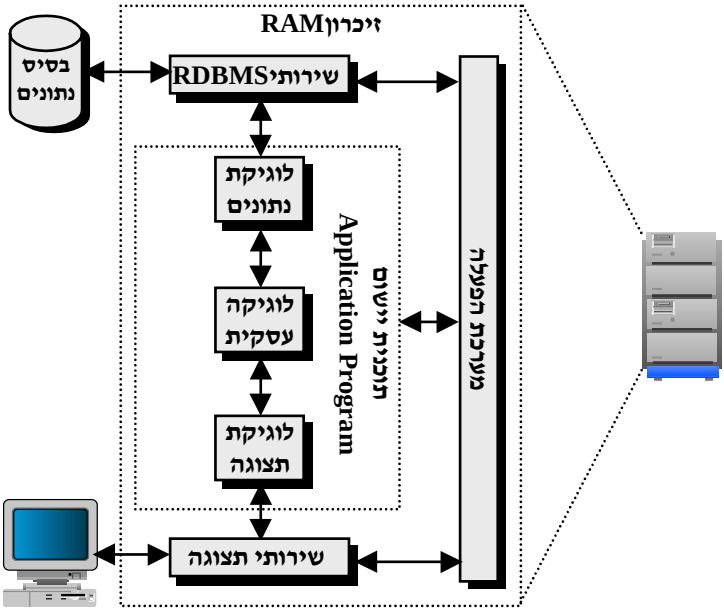
- **שירותי תצוגה (Presentation Services):** מרכיב זה עוסק בניהול התצוגה, כלומר ניהול המסך. רכיב זה מגדיר את המראה של המסך, את האובייקטים שניתן להציג (תפריטים מסוג Pull Down, לחצנים כגון Check Box או Radio Button, שדות להצגת נתונים או שדות ברי עדכון וכד'), מבצע את בקרת הסמן (Cursor Control), מנהל את החלונות, מציג אובייקטים גרפיים שונים, מנהל את הצבעים ועוד. במסופים טיפשים, כמו 3270 של יבמ או VT-100 של דיגיטל (כיום, קומפאק), שירותי התצוגה היו מוגבלים מאוד. בתחנות עבודה מבוססות Unix המשתמשות בפרוטוקול X-Windows או במחשבים אישיים הפועלים עם מערכת הפעלה Windows, שירותי התצוגה עשירים מאוד. רכיב זה איננו מהווה חלק מהיישום אלא חלק מהשירותים שמערכת ההפעלה של מחשב הלקוח מספקת.

- **לוגיקת תצוגה (Presentation Logic):** מרכיב זה משתמש בשירותי התצוגה וקובע את ההתנהגות של היישום בכל הקשור לתצוגת וקליטת נתונים. בסביבות מבוססות Windows, מרכיב זה מקשיב לאירועים שונים הנוצרים כתוצאה מפעולות שהמשתמש מבצע ומפעיל לוגיקה בהתאם. למשל, אם המשתמש מבצע פעולת Drag and Drop של אובייקט כלשהו על חלון מסוים, רכיב זה מפעיל לוגיקה לבדיקה אם מותר לגרור את האובייקט לחלון זה ואם כן מבצע בדיקות תקינות נוספות בהתאם לצורך.

- **לוגיקת היישום (Application Logic):** מרכיב זה מבצע את הלוגיקה העסקית בהתאם לדרישות היישום. נושאים אופייניים לקטע זה הם בדיקות תקינות של הנתונים, צבירת נתונים, ביצוע חישובים שונים במידה ומתקיים תנאי כלשהו וכד'. מקובל לקרוא ללוגיקה זו גם בשם **כללים עסקיים** (Business Rules). מרכיב זה פונה למרכיב הנתונים כדי לקבל את הנתונים הדרושים מבסיס הנתונים, ופונה למרכיב לוגיקת התצוגה כדי להפעיל שירותי תצוגה או לקלוט את התגובות של המשתמש.

- **לוגיקת נתונים (Data Logic):** מרכיב זה מטפל בלוגיקה של שליפת הנתונים ועדכון הנתונים. הוא גם מבצע בדיקות מיוחדות של תקינות הנתונים לפני כתיבתם לבסיס הנתונים או לפני עדכון הנתונים. מרכיב זה מבצע את פקודות SQL או מפעיל את פרוצדורות בסיס הנתונים.

בתוכנית יישום הכתובה בצורה מודולרית, ניתן להבחין בבירור במרכיבי לוגיקת התצוגה, הלוגיקה העסקית ולוגיקת הנתונים. בתוכניות הכתובות בצורה פחות מודולרית, קטעים אלה מתערבבים זה בזה ולעיתים קשה להבחין מי עושה מה.



תרשים 16.15: חלוקת תוכנית היישום למרכיבים שונים.

תרשים 16.15 מציג את המרכיבים השונים של תוכנית היישום המקוונת ואת האינטראקציה שלהם עם מערכות תוכנה נוספות, כגון מערכת RDBMS, מערכת ניהול התצוגה ומערכת ההפעלה. כפי שניתן לראות הקטע העוסק בלוגיקת התצוגה נמצא באינטראקציה בעיקר עם המערכת המספקת את שירותי התצוגה ואילו הקטע העוסק בניהול נתונים נמצא באינטראקציה עם מערכת RDBMS באמצעות פקודות SQL, על ידי הפעלת SQL API או על ידי הפעלת פרוצדורות בסיס נתונים.

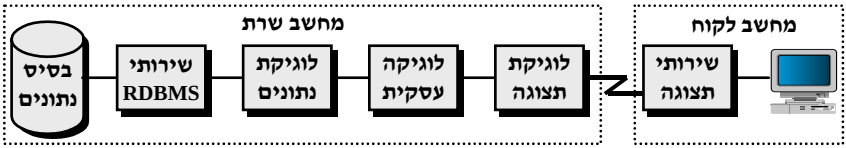
בסביבה לא מבוזרת, כל המרכיבים מבוצעים בתוך מחשב אחד על ידי CPU אחד או יותר ובאחריות מערכת הפעלה אחת. בסביבה מבוזרת ניתן לבזר את המרכיבים השונים בין מחשבים שונים.

מודל מסגרת לביזור מרכיבי תוכנית יישום

תהליך הביזור מאפשר ביצוע של אחד או יותר מהמרכיבים שהצגנו בסעיף הקודם, במחשבים שונים הקשורים ביניהם ברשת תקשורת כלשהי. עקרונית ניתן לבצע כל אחד מהמרכיבים על מחשב נפרד, כל אחד עם מערכת הפעלה משלו או לבצע יותר ממרכיב אחד על אותו מחשב. נראה מספר מודלים נפוצים של ביזור הרכיבים בין המחשבים השונים.

תצוגה מבוזרת (Distributed Presentation)

במודל זה כל מרכיבי התוכנית מבוצעים במחשב השרת, ורק שירותי התצוגה מופעלים על מחשב מרוחק.

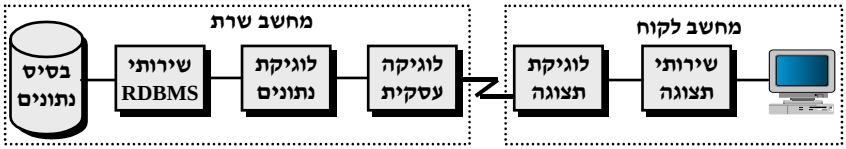


תרשים 16.16: תצוגה מבוזרת.

מחשב השרת אינו מודע כלל לעובדה שהתצוגה עוברת עיבוד נוסף במחשב הלקוח ומבחינתו הוא מתקשר עם מסופים "טיפשים" כמו מסופי 3270 או VT100. מחשב הלקוח, שהוא בדרך כלל מחשב PC הפועל עם מערכת הפעלה Windows, מפעיל תוכנה מיוחדת הכוללת בתוכה מרכיב של החקיית מסוף (Terminal Emulation) הגורם למחשב השרת לחשוב שהוא מתקשר עם מסוף רגיל וכן מרכיב של Screen Scrapping. מרכיב זה ממיר את התצוגה הרגילה של מסוף טיפש לתצוגה גרפית. מאחר ומחשב הלקוח מכיל מעבד משל עצמו ומסך גרפי, ניתן לבצע תחפושת למסכים הפשוטים ולהפכם למסכים גרפיים נאים לעין. מאחר ורוב התוכנות האלו מכיל גם שפת תכנות (כגון Visual Basic), ניתן להשתמש בתוכנות אלו גם לביצוע חלק מלוגיקת התצוגה ואפילו חלק מהלוגיקה העסקית. נציין ששימוש בתוכנות אלו לביצוע לוגיקה מביא לרובד נוסף של סיבוכיות בתחזוקה, מאחר ואותה לוגיקה נמצאת בשני מחשבים שונים – בשרת ובמחשב הלקוח. השימוש במודל זה הוא מוגבל יחסית ומשמש בעיקר כדי לספק ממשקים גרפיים לתוכנה הפועלת על מחשב מרכזי.

תצוגה מרוחקת (Remote Presentation)

במודל זה, שירותי התצוגה ולוגיקת התצוגה מבוצעים במחשב הלקוח, ואילו מרכיב הלוגיקה העסקית וניהול הנתונים מבוצעים במחשב השרת.

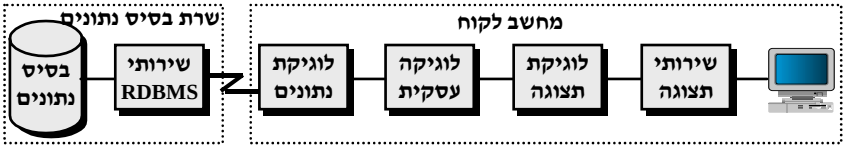


תרשים 16.17: תצוגה מרוחקת.

במודל זה שירותי התצוגה ולוגיקת התצוגה מתבצעים באופן מלא במחשב הלקוח. היישום העובד במחשב השרת מודע לכך שהוא מתקשר עם תחנת עבודה ולא עם מסוף טיפש. מקובל לקרוא למודל זה גם בשם Thin Client, כלומר מחשב לקוח העוסק רק בתצוגה ולא מבצע לוגיקה עסקית.

שרת בסיס נתונים (Database Server)

במודל זה כל מרכיבי היישום מתבצעים על מחשב הלקוח ורק מרכיב ניהול הנתונים מתבצע במחשב השרת. מאחר וכל הלוגיקה מבוצעת על מחשב הלקוח ולעיתים לוגיקה זו יכולה להיות מורכבת מאוד, מקובל לקרוא למודל זה גם בשם Fat Client ("לקוח שמן").

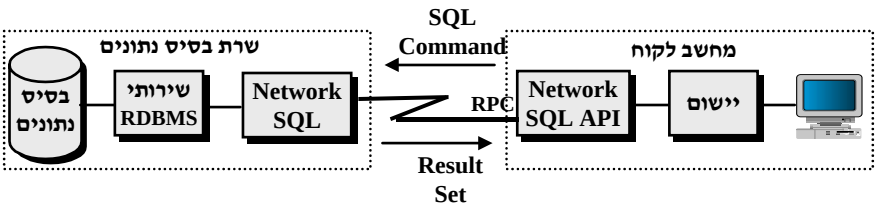


תרשים 16.18: שרת בסיס נתונים.

בשנים הראשונות להופעת מודל שרת/לקוח בשוק, זה היה המודל הנפוץ ביותר וחלק גדול מהיישומים הפועלים כיום, עובדים בצורה זו. יתרון שיטה זו הוא שכל היישום פועל על מחשב הלקוח ולכן קל יחסית לפתח ולנפות יישומים אלה משגיאות. רוב מחוללי היישומים המודרניים כמו PowerBuilder, Visual Basic, Magic, Delphi ואחרים תמכו בגרסאות הראשונות שלהם רק במודל עבודה זה.

במודל שרת בסיס הנתונים, מחשב הלקוח שולח פקודת SQL אל שרת בסיס הנתונים, המבצע את פקודת SQL: מבצע אופטימיזציה של הפקודה, מחפש בטבלאות, מחפש באינדקסים, מבצע פעולות JOIN וכד'. בסופו של דבר, שולח שרת בסיס הנתונים את הטבלה התוצאתית אל מחשב הלקוח. בשיטה זו תוכנית היישום המופעלת במחשב הלקוח נהנית מהעוצמה של שפת SQL. ניתן לשפר את הביצועים של שרת בסיס הנתונים על ידי ניהול מושכל של מאגרי קלט/פלט, ניהול זיכרון מטמון, טכניקות מהירות לאינדקסים וכד'. כלומר, שרת בסיס הנתונים מבצע את כל האופטימיזציה כדי לבצע את תפקידו ביעילות מרבית.

כדי לאפשר פנייה מרחוק אל שרת בסיס הנתונים, פיתחו היצרנים השונים את תפישת Networked SQL (SQL ברשת). כלומר, זו האפשרות לשגר פקודות SQL דרך רשת תקשורת אל השרת ולהחזיר את טבלת התוצאה אל מחשב הלקוח. היצרנים פיתחו מרכיב מיוחד הפועל על מחשב הלקוח ויוצר את הקישוריות עם שרת בסיס הנתונים. לדוגמה, ב-Oracle קוראים לרכיב זה בשם SQL*Net, ב-Sybase הוא נקרא DBLib.



תרשים 16.19: עבודה עם SQL ושרת בסיס נתונים.

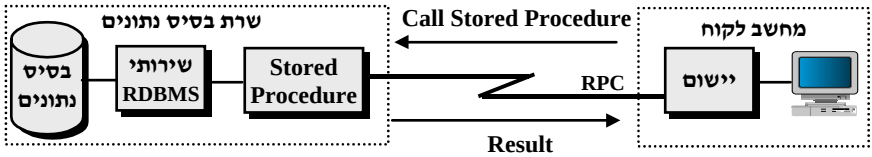
היישום הפועל על מחשב הלקוח פונה לבסיס הנתונים על ידי פקודות SQL או על ידי גישה לממשק התכנות של SQL. הרכיב של Networked SQL מקבל את הפנייה, ותוך שימוש במנגנון RPC פונה אל שרת בסיס הנתונים. מרכיב ה Networked SQL הנמצא על השרת, מקבל את הפקודה ומעביר אותה למערכת RDBMS. עם קבלת התוצאה מתחיל תהליך העברת הטבלה התוצאתית חזרה אל מחשב הלקוח. קיימות שיטות שונות להעברת התוצאה: העברת דף ראשון ליישום ולאחר מכן העברת יתר הדפים ברקע, העברת התוצאה בשלמותה בלבד ליישום וכד'. למרות ששיטה זו מעבירה יחסית מעט נתונים על הרשת, עדיין הכמות הגדולה של האינטראקציות בין מחשב הלקוח לשרת בסיס הנתונים היא גדולה. מכיון שכל אינטראקציה כזאת עוברת דרך רשת התקשורת, זמני התגובה של היישומים הפועלים בשיטה זו הם חלשים יחסית.

ניהול הנתונים מבוצע על ידי מחשב השרת. מחשב הלקוח מטפל בלוגיקה של היישום ובתצוגה. הפניות ממחשב הלקוח למחשב השרת הן לצורך קבלת הנתונים. מקובל לקרוא למחשב השרת גם **שרת בסיס הנתונים** (Database Server), כי הוא אחראי לניהול בסיס הנתונים. מקובל להבחין בין שרתי בסיסי נתונים בהתאם לרמת האינטראקציה בין מחשב הלקוח ומחשב השרת מבחינת סוג הנתונים המועברים ביניהם: רשומה בודדת או טבלה שלמה.

לאחר ההתלהבות הראשונית מקלות הפיתוח במחוללי היישומים החדשים ומהממשקים הגרפיים היפים של היישומים החדשים, החלו לצוץ גם הבעיות של מודל "הלקוח השמן". בעיות אלו התמקדו בעיקר בביצועים החלשים יחסית ובקושי בתחזוקה ובהפצת גרסאות חדשות בגלל הצורך להפיץ את התוכנה למספר גדול של מחשבי לקוח.

שרת בסיס נתונים עם פרוצדורות בסיס נתונים

ניתן לשפר את המודל הקודם על ידי שימוש בפרוצדורות בסיס נתונים (Stored Procedures). במקום לפנות אל שרת בסיס הנתונים על ידי פקודות SQL בודדות, ניתן לבנות פרוצדורת בסיס נתונים שתכיל מספר פקודות SQL וכן לוגיקה מסוימת.



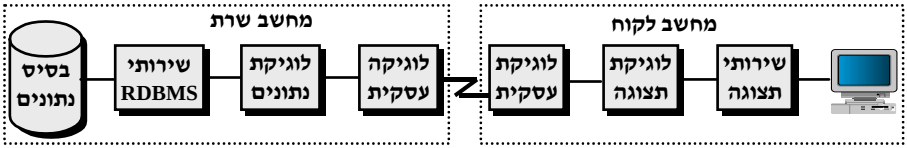
תרשים 16.20: עבודה עם פרוצדורות בסיס נתונים.

במודל זה, ההידודיות בין היישום לבין שרת בסיס הנתונים היא קצרה ביותר ומבוססת על מנגנון RPC המעביר את הפרמטרים הדרושים. פרוצדורת בסיס הנתונים, הכתובה בשפה קנינית כגון PL/SQL או Transact-SQL או בשפת דור שלישי פרוצדורלית כגון C++ או Java, מתבצעת על השרת ובאחריות המערכת.

מקובל לכנות את שרתי בסיס הנתונים האלה בשם **שרתים אקטיביים** (Active Database Servers), בזכות יכולתם להכיל לא רק נתונים אלא גם פרוצדורות המבצעות חלק מהיישום. העבודה עם פרוצדורות בסיס נתונים, מקטינה את העומס על רשת התקשורת, משום שיש לשגר רק את שם הפרוצדורה ואת הפרמטרים ולא את פקודות SQL, שיכולות לעיתים להיות ארוכות מאוד. הנקודה המרכזית היא היכולת לבצע הידור ומיטוב (אופטימיזציה) מראש של פקודות SQL ובכך להביא לשיפור משמעותי בזמני התגובה.

לוגיקה מבוזרת (Distributed Logic)

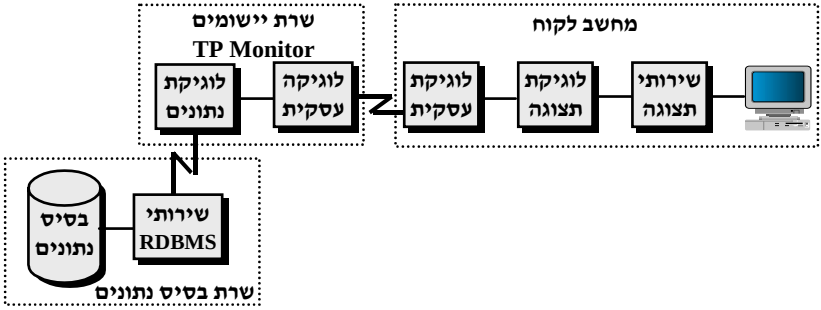
מודל זה מאפשר פיצול של הלוגיקה העסקית בין מחשב הלקוח לבין מחשב השרת. התצוגה מבוצעת באופן מלא על ידי מחשב הלקוח ולוגיקת הנתונים מבוצעת באופן מלא על ידי מחשב השרת.



תרשים 16.21: לוגיקה מבוזרת.

מודל זה הפך לנפוץ עם השנים, מאחר והוא מאפשר בניית פרוצדורות משותפות הפועלות בשרת והעמדת שירותים אלה לרשות כל הלקוחות. מאחר ומחשב השרת הינו בדרך כלל מחשב חזק יותר, הביצועים של הפרוצדורות בשרת הן טובות יותר. כדי להשתמש במודל זה, יש צורך להפעיל תוכנה מיוחדת על השרת, Transaction Processing Monitor או בקיצור TP Monitor, המספק אוסף שלם של שירותים לכל יישומי הלקוח. תוכנה זו מנהלת את התורים במחשב השרת, מסדרת את הפניות על פי עדיפויות, מפעילה את הפרוצדורות בשרת, מאפשרת צורות תקשורת שונות (אסינכרונית או סינכרונית), מספקת שירותי שרידות במקרה של נפילת שרת על ידי העברת המשימות לשרת גיבוי ועוד. את פרוצדורות השרת ניתן לכתוב בשפת תכנות מדור שלישי כגון C++, Cobol, Java וגם תוך שימוש ביכולות חדשות יחסית של מחוללי יישומים כגון PowerBuilder, Magic, Forte ואחרים. מחוללים אלה מסוגלים כיום לחולל קוד שיפעל במחשב השרת ולא רק במחשב הלקוח. מחוללים כגון Forte מסוגלים לייצר קוד הפועל במחשב השרת או במחשב הלקוח ומאפשרים העברה קלה של קוד ממחשב למחשב ללא צורך בשינוי ביישום.

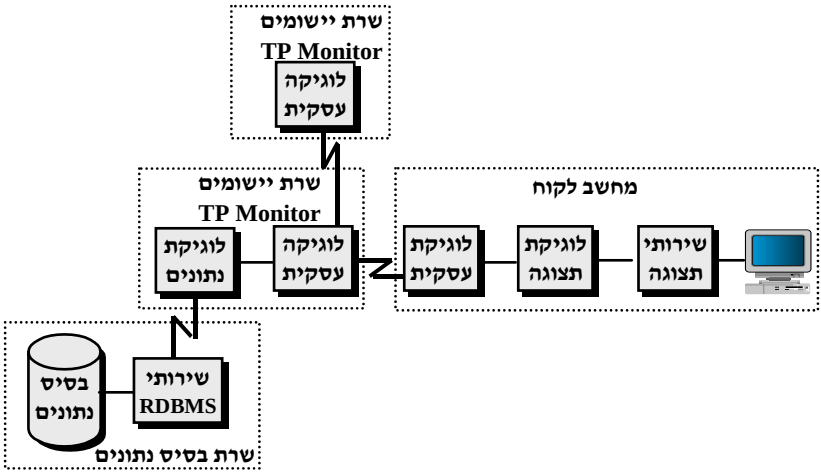
ניתן להרחיב מודל זה גם על ידי יצירת הפרדה בין שרת היישומים (Application Server) לבין שרת בסיס הנתונים (Database Server). הפרדה זו מאפשר חלוקת אחריות ברורה למרכיבי היישום השונים. אגב, ההפרדה בין שרת היישומים ושרת בסיס הנתונים היא הפרדה לוגית ולכן, שני רכיבים אלה יכולים לפעול על שרת אחד או על שרתים נפרדים.



תרשים 16.22: לוגיקה מבוזרת ושימוש בשרת יישומים.

מקובל לקרוא למודל זה בשם **שרת/לקוח תלת-שכבתי** (Three Tier Client/Server). מכיון שתוכנות TP Monitor כמו Tuxedo, CICS/6000, MTS (Microsoft Transaction Monitor) ואחרות, תומכות גם במודל עבודה עם מספר שרתי יישומים מבוזרים (Distributed TP Monitor), ניתן לבנות יישומים רב שכבתיים הפועלים בשרתים שונים. למודל זה מקובל לקרוא בשם **שרת/לקוח רב-שכבתי** (n-Tier Client/Server). הדבר מוצג בתרשים 16.23.

לסיכום, מודל העבודה עם לוגיקה מבוזרת מורכב יותר, כי הוא מחייב שימוש בתוכנה TP Monitor, שימוש במחולל יישומים עבור הלקוח ושפת תכנות מדור שלישי עבור השרת. עם זאת, הביצועים של מודל עבודה זה נחשבים לטובים, יכולת הגידול (Scalability) שלו נחשבת לטובה מאוד והוא מאפשר בניית ישומים שישרתו מאות ואלפי משתמשים בצורה אמינה, תוך שמירה על רמת זמינות ושרידות גבוהה.



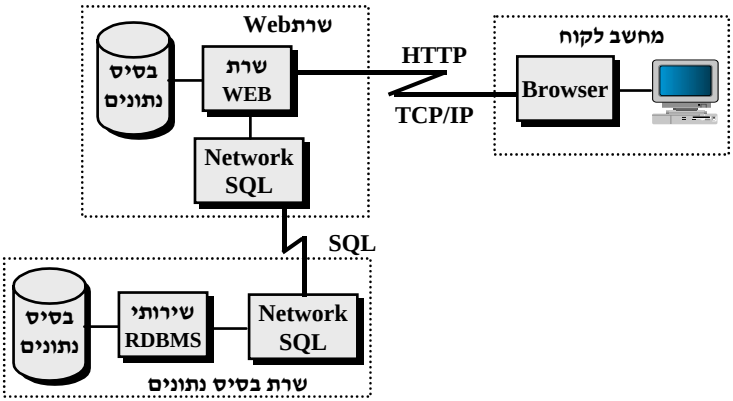
תרשים 16.23: מודל עבודה עם מספר שרתי יישומים.

מודל האינטרנט

בשנים האחרונות החל להופיע מודל שרת/לקוח חדש, מודל האינטרנט. מודל זה מבוסס על דפדפן (Browser) הפועל במחשב הלקוח, על פרוטוקול תקשורת מיוחד HTTP הפועל מעל פרוטוקול התקשורת TCP/IP ועל שרת Web. שרת זה מריץ תוכנה מיוחדת המקבלת את הפניות ממחשבי הלקוח, שולפת את דפי המידע מתוך בסיס נתונים ומשגרת אותם חזרה אל מחשב הלקוח. דפי המידע המנוהלים בבסיס הנתונים של שרת Web הם בדרך כלל עתירי טקסט, גרפיקה, קול וכד'. שפת Java הביאה לכך שניתן לשגר מהשרת אל מחשב הלקוח גם לוגיקה בשיטת ישומונים (Applets) הכתובים בשפה זו. סביבת העבודה Java מכילה רכיב תוכנה מיוחד Java Virtual Machine המאפשר ללוגיקה להתבצע על מיגוון רחב מאוד של מחשבי לקוח, ולכן היא מתאימה ביותר למשימה זו.

בתחילה היה מקובל להשתמש במערכות פשוטות לניהול קבצים (File system) כדי לנהל את דפי התוכן השונים. עם הזמן ובגלל הצורך לנהל אובייקטים מורכבים יותר, החלה מגמה של מעבר ממערכות פשוטות של קבצים למערכות RDBMS לניהול הנתונים של יישומי האינטרנט. בגלל הצורך לספק ביצועים מעולים, לשרת אלפי משתמשים בו-זמנית, לבצע גיבוי תוך כדי עבודה וכד', מפעילים היום רוב אתרי האינטרנט הגדולים מערכות RDBMS לניהול דפי HTML ולניהול האובייקטים המורכבים הדרושים ליישומים אלה. מודל האינטרנט מוצג בתרשים 16.24.

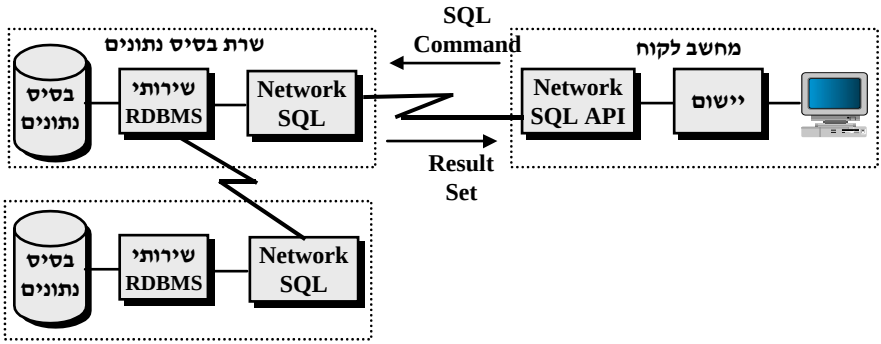
כאשר דרושים דפי מידע דינמיים המכילים נתונים מתוך בסיסי הנתונים הארגוניים, ניתן להשתמש ברכיב הקישוריות של מערכות RDBMS ובממשק מיוחד כדי לשגר פקודות SQL אל שרת בסיס הנתונים, לקבל את התוצאות ולשלב אותן בתוך דפי המידע המשוגרים חזרה אל מחשב הלקוח. ניתן גם לגשת לבסיס הנתונים ישירות באמצעות ממשק JDBC (Java DataBase Connectivity) מתוך ישומי Java המשוגרים אל מחשב הלקוח.



תרשים 16.24: מודל האינטרנט.

בסיס נתונים מבוזר

כל המודלים שהצגנו עד כה הניחו שהיישום עובד עם בסיס נתונים אחד בלבד. ניתן להגדיר מודל עבודה נוסף שבו הנתונים עצמם מבוזרים בין בסיסי נתונים שונים, כלומר פיזור הטבלאות השונות בין מחשבים שונים תוך כדי שמירה על העובדה שמדובר בבסיס נתונים לוגי אחד.

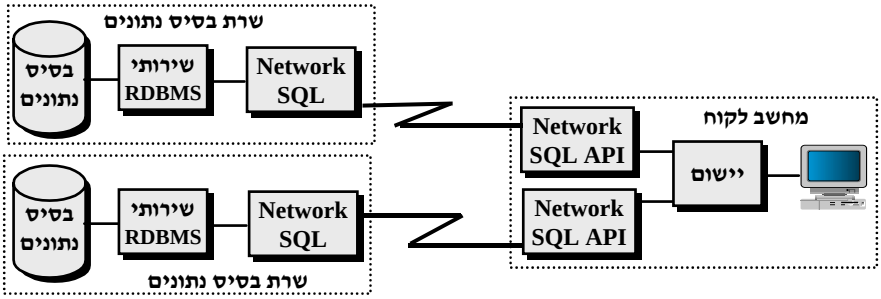


תרשים 16.25: בסיס נתונים מבוזר.

במודל זה, ביזור בסיס הנתונים הינו שקוף מבחינת היישום. היישום פונה למערכת ניהול בסיס הנתונים בצורה רגילה. תפקיד מערכת זו לגשת אל הנתונים המנוהלים בצמתים שונים ברשת ולהביאם אל היישום. מערכת RDBMS התומכת במודל עבודה זה נקראת מערכת Distributed RDBMS או בקיצור DRDBMS. אם בכל הצמתים פועל בסיס נתונים של אותו יצרן, מקובל לקרוא למצב זה בשם **בסיס נתונים מבוזר הומוגני**. אם בצמתים השונים פועלים בסיסי נתונים של יצרנים שונים מקובל לקרוא למצב זה בשם **בסיס נתונים מבוזר הטרוגני**.

למרות הנסיונות הרבים לפתח מודל עבודה זה, האתגרים בביזור הנתונים באופן שקוף ליישום גדולים ועדיין לא ניתן לומר שכל הבעיות נפתרו. יצרנים שונים תומכים ברמות שונות במודל עבודה זה, אולם הוא אינו נפוץ. למשל, רוב יצרני מערכות RDBMS תומכים כיום בביצוע תנועות מבוזרות (Distributed Transactions) תוך שימוש בפרוטוקול Two Phase Commit. למרות זאת, רוב מפתחי היישומים נמנעים משימוש בבסיסי נתונים מבוזרים ובתנועות מבוזרות בגלל המורכבות הרבה ובגלל הרגישות לרשת התקשורת. נושא מערכות DRDBMS יוסבר בפרק 17.

אפשרות נוספת היא שהיישום יפעל מול מספר בסיסי נתונים שונים. במודל זה היישום חשוף לעובדה שהנתונים מנוהלים במספר בסיסי נתונים שונים ופועל באופן ישיר עם מספר מערכות RDBMS שונות.



תרשים 16.26: עבודה עם מספר שרתי בסיס נתונים שונים.

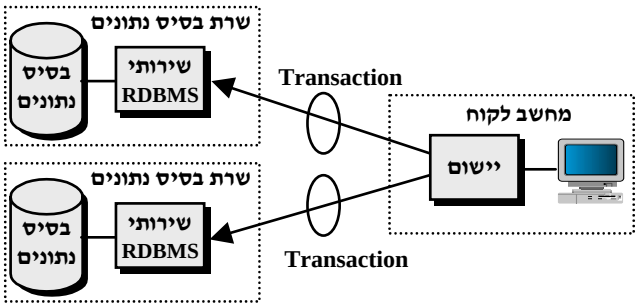
ניתן לראות שבמחשב הלקוח מותקן רכיב קישוריות לכל אחד מבסיסי הנתונים בנפרד. היישום מבצע Connect עם כל בסיס נתונים בנפרד, משגר אליו פקודות SQL ומקבל את התוצאות.

רמות התמיכה בתנועות הפועלות בסביבה מבוזרת

נסקור עכשיו את רמות התמיכה השונות של מערכות RDBMS בתנועות בסביבה מבוזרת (Distributed Transactions). רמות אלו מבוססות על מודל כללי שהוצג על ידי חברת יבמ בשנת 1981. המודל מבוסס על ארבע רמות שונות, מתמיכה בתנועה המכילה משפט SQL בודד ועד לתנועה הכוללת מספר פקודות SQL המטפלות בנתונים הנמצאים בבסיסי נתונים שונים.

בקשה מרוחקת (Remote Request)

זוהי הרמה הבסיסית ביותר וכאן התמיכה הנדרשת היא לביצוע פקודת SQL בודדת מול בסיס נתונים מרוחק. אם קיימת תמיכה ברמה זו, תוכנית יישום יכולה לבצע מספר פקודות SQL מול בסיס נתונים מרוחק אחד או מספר בסיסי נתונים מרוחקים. כל פקודת SQL מתייחסת לנתונים בבסיס נתונים אחד בלבד ומהווה תנועה בפני עצמה. משמעות הדבר היא שאחרי כל פקודה, תוכנית היישום מבצעת תחילה את הפקודה COMMIT.

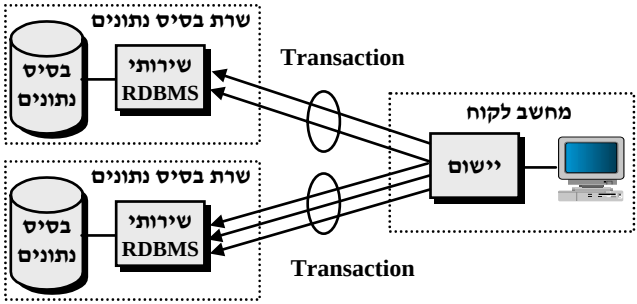


תרשים 16.27: בקשה מרוחקת.

תמיכה ברמה זו מאפשר לתוכנית יישום הפועלת במחשב לקוח לבצע שאילתות וגם עדכונים פשוטים מול מספר בסיסי נתונים מרוחקים שונים. האחריות על אמינות בסיסי הנתונים כתוצאה מפעולות העדכון היא על תוכנית היישום. אם בוצעה תנועה בבסיס נתונים מרוחק אחד והתנועה בבסיס הנתונים השני נכשלה, באחריות תוכנית היישום לבצע Rollback לתנועה הראשונה.

תנועה מרוחקת (Remote Transaction)

זוהי רמת תמיכה גבוהה יותר בביצוע תנועות בסביבה מבוזרת. במצב זה, תוכנית היישום יכולה לבצע תנועה המורכבת ממספר פקודות SQL מול בסיס נתונים מרוחק אחד או יותר.

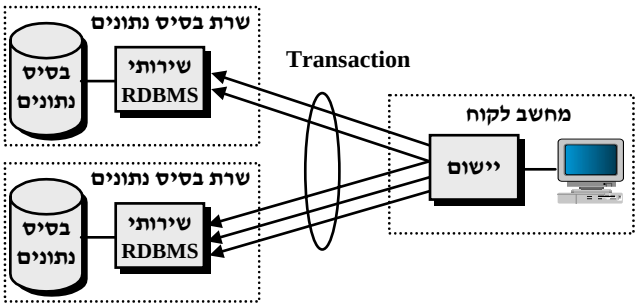


תרשים 16.28: תנועה מרוחקת.

ברמה זו, בסיס הנתונים תומך בתנועה המורכבת ממספר פקודות SQL, אולם כל פקודות SQL במסגרת התנועה מתייחסות לנתונים בבסיס נתונים אחד בלבד. האחריות על אמינות בסיסי הנתונים כתוצאה מפעולות העדכון היא על תוכנית היישום. אם בוצעה תנועה בבסיס נתונים מרוחק אחד והתנועה בבסיס הנתונים האחר נכשלה, באחריות תוכנית היישום לבצע גלילה לאחור (Rollback) כדי לבטל את התנועה הראשונה ולהחזיר את המצב לקדמותו.

תנועה מבוזרת (Distributed Transaction)

במצב זה תוכנית היישום יכולה לבצע תנועות הכוללות מספר פקודות SQL כאשר פקודות SQL השונות בתוך התנועה יכולות להתייחס לבסיסי נתונים שונים.



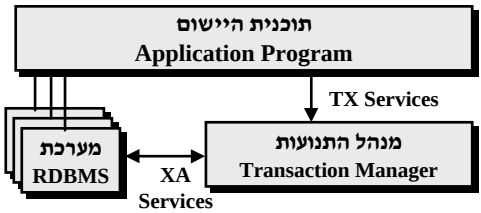
תרשים 16.29: תנועה מבוזרת.

ברמה זו, בסיס הנתונים תומך בביצוע תנועות המורכבות ממספר פקודות SQL שונות. כל פקודת SQL בתוך התנועה יכולה לפנות לבסיס נתונים אחד בלבד, אולם בתוך התנועה ניתן לבצע פקודות SQL הפונות לבסיסי נתונים שונים. כאן, זו אחריות מערכת RDBMS לשמור על אמינות הנתונים ולא אחריות תוכנית היישום. אם פקודת עדכון אחת בתוך התנועה בוצעה בהצלחה אולם פקודה אחרת נכשלת, המערכת צריכה לבצע גלילה לאחור בבסיסי נתונים שונים. כדי לתמוך ברמה זו, פיתחו יצרני מערכות RDBMS את הפרוטוקול Two Phase Commit, ובקיצור – 2PC, התומך בתנועות המעדכנות בסיסי נתונים שונים. כמעט כל היצרנים המובילים של מערכות RDBMS המסחריות תומכים בפרוטוקול זה, אם כי מימשו אותו בצורות שונות. פרוטוקול זה מוסבר בסעיף הבא.

בעיה מתעוררת כאשר בסיסי הנתונים השונים אינם מתוצרת יצרן אחד או מסוג אחד. במצב זה, לא ניתן להשתמש בפרוטוקול 2PC של יצרן מסוים. הפתרון הוא להשתמש בתוכנת TP Monitor התומכת בתנועות מבוזרות. ארגון X/Open הגדיר מודל מיוחד, מודל Distributed Transaction Processing או בקיצור מודל DTP, המגדיר כיצד TP Monitor אחד יכול לפקח על תנועות מבוזרות.

המודל מגדיר שלושה מרכיבים שונים ואת הממשק בין מרכיבים אלה. המרכיבים הם: תוכנית היישום, מנהל התנועות ומערכות RDBMS או מנהלי המשאבים (Resource Managers) כמו שהם נקראים במודל. המודל מגדיר שני ממשקים: ממשק TX Services בין תוכנית היישום ומנהל התנועות וממשק XA Services בין מנהל התנועות לבין מערכת RDBMS.

ממשק TX Services מאפשר לתוכנית היישום להודיע למנהל התנועות מתי תנועה מבוזרת מתחילה על ידי שימוש בפונקציה tx_begin, להודיע על סיום מוצלח על ידי הפונקציה tx_commit או על כשלון התנועה על ידי הפונקציה tx_rollback. ממשק XA Services מספק אוסף של פונקציות המאפשרות למנהל התנועות לשמור על אמינות התנועה המבוזרת ולממש את פרוטוקול 2PC.

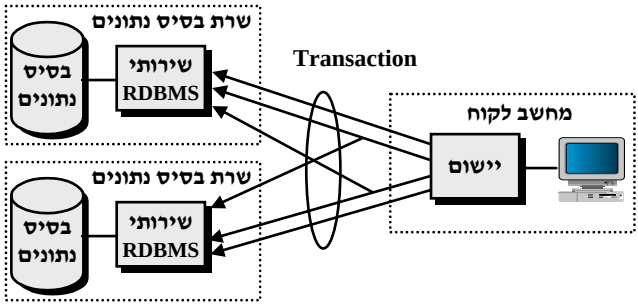


תרשים 16.30: מודל X/Open DTP.

מערכות RDBMS המסחריות המובילות תומכות בתקן XA ולכן מאפשרות למנהלי תנועות שונים לתמוך בתנועות מבוזרות בסביבה הטרוגנית. למרות תמיכה זו, רמת השימוש במנהלי תנועות לביצוע תנועות מבוזרות היא נמוכה יחסית, בגלל המורכבות והבעייתיות של התנועות המבוזרות.

בקשה מבוזרת (Distributed Request)

זוהי הרמה הגבוהה ביותר של תמיכה בתנועות מבוזרות. במצב זה, פקודת SQL בודדת יכולה לפנות לטבלאות הנמצאות בבסיסי נתונים שונים. תנועה אחת יכולה להכיל מספר רב של פקודות SQL כאלו.

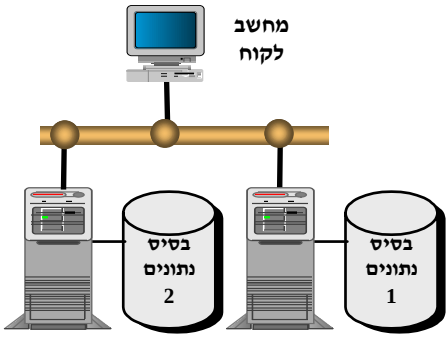


תרשים 16.31: בקשה מרוחקת.

האתגר העיקרי ברמה זו הוא כמובן ביכולת מערכת RDBMS להסתיר מהיישום את ביזור הנתונים, ולאפשר לו לבצע פקודות SQL הפונות לטבלאות שונות, מבלי לדעת היכן באמת טבלאות אלו נמצאות. כפי שכבר הוסבר בסעיף הקודם, זוהי מערכת DRDBMS. רוב היצרנים אינם תומכים באופן מלא ברמת ביזור זו. הבעיה העיקרית היא כמובן שמערכת DRDBMS צריכה לנהל מעקב היכן בדיוק נמצאות הטבלאות השונות ולבצע אופטימיזציה, הלוקחת בחשבון את הביזור ואת עלות העברת הנתונים ברשת התקשורת. לשיטת אופטימיזציה זו מקובל לקרוא **אופטימיזציה גלובלית** והיא קשה מאוד למימוש. כאמור, נושא זה יוסבר בפרק 17.

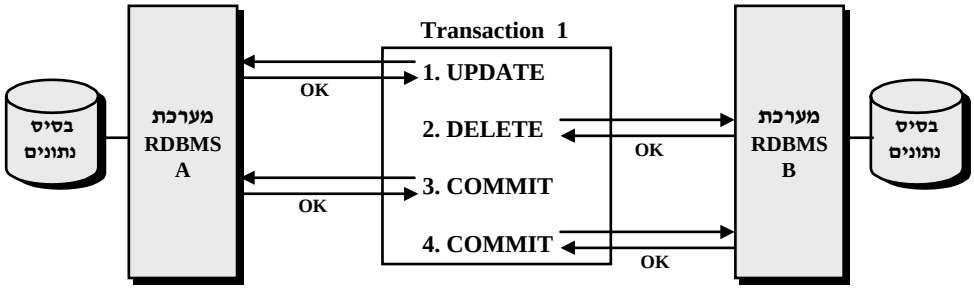
פרוטוקול Two Phase Commit לניהול תנועות מבוזרות

מערכת RDBMS מכילה מנגנון ניהול תנועות המבטיח שאמינות בסיס הנתונים לא תיפגם כתוצאה מכשלון בביצוע תנועה. בסעיף זה נרחיב את הדיון על שמירת אמינות בסיס הנתונים במצב בו באותה תנועה משתתף יותר מבסיס נתונים אחד. נתבונן לרגע על תוכנית יישום הפועלת במחשב לקוח כלשהו ומבצעת תנועה מבוזרת, כלומר מעדכנת שני בסיסי נתונים שונים, הפועלים על שרתים שונים, בתוך אותה תנועה.



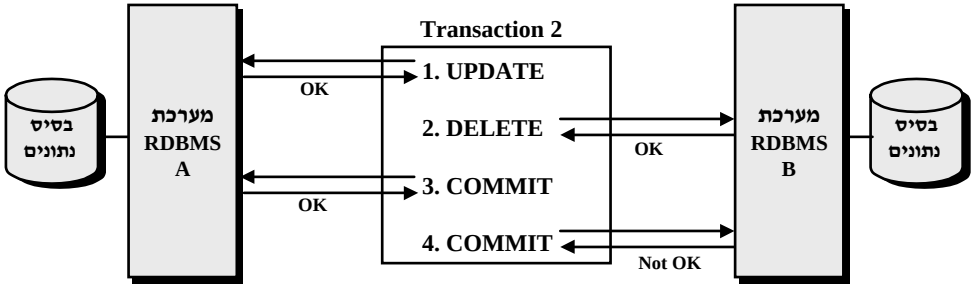
תרשים 16.32: סביבת העבודה של תנועה מבוזרת.

נניח שהתוכנית החלה את התנועה וביצעה את הפקודה UPDATE לטבלה הנמצאת בבסיס נתונים הראשון ופקודה DELETE לטבלה הנמצאת בבסיס נתונים שני. בשלב זה, תוכנית היישום משגרת את הפקודה COMMIT בנפרד לכל אחת משתי מערכות RDBMS. אם כל אחת משתי המערכות הצליחה לבצע COMMIT, אזי מבחינת תוכנית היישום התנועה המבוזרת הסתיימה בהצלחה. תרשים 16.33 מדגים תנועה מבוזרת שמסיימת בהצלחה.



תרשים 16.33: תנועה מבוזרת המסתיימת בהצלחה.

מה יקרה אם לאחר שתוכנית היישום שיגרה את הפקודה COMMIT למערכת RDBMS הראשונה וקיבלה תשובה שהפקודה הצליחה, היא משגרת את הפקודה COMMIT למערכת השנייה ומקבלת תשובה שהפקודה נכשלה. הכישלון יכול להיות למשל, בגלל בעיה בדיסק, תקלה במערכת ההפעלה או כל סיבה אחרת. במצב זה, תוכנית היישום תרצה לבצע גלילה לאחור בשני בסיסי הנתונים. הבעיה היא, שהמערכת הראשונה כבר ביצעה COMMIT ולכן לא יכולה לבצע ROLLBACK. יש לנו כאן מצב שבו נפגמה אמינות בסיסי הנתונים.

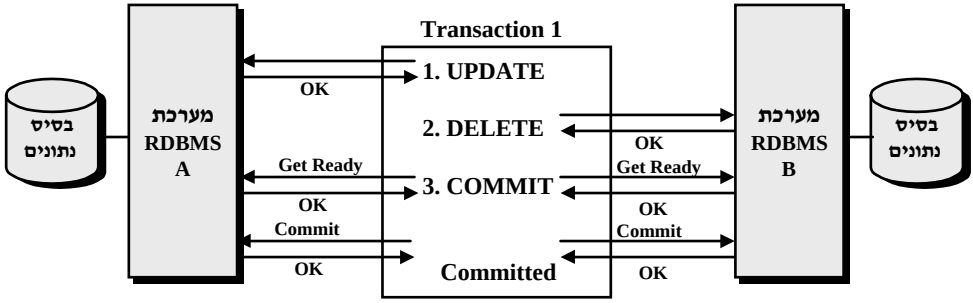


תרשים 16.34: תנועה מבוזרת המסתימת בכשלון של עדכון בסיס נתונים אחד.

כדי להתמודד עם מצב זה פותח פרוטוקול Two Phase Commit או בקיצור 2PC. פרוטוקול זה מתמודד עם הבעיה על ידי הגדרת שני שלבים לתהליך Commit. לצורך הפרוטוקול, אחת מבין מערכות RDBMS המשתתפות בתנועה המבוזרת תוגדר **כמערכת מתאמת** (Coordinator). בדרך כלל זו המערכת הפועלת באותו מחשב שבו פועלת תוכנית היישום. הפרוטוקול מגדיר שני שלבים:

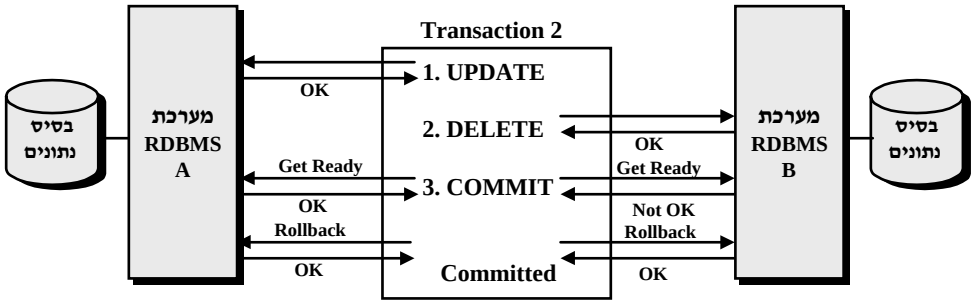
- **שלב ההכנה (Get Ready Phase)**: בשלב זה המערכת המתאמת שולחת את הפקודה Get Ready לכל מערכות RDBMS המשתתפות בתנועה. כל מערכת המקבלת את הפקודה GET READY חייבת להיכנס למצב שבו היא מוכנה לבצע COMMIT או ROLLBACK על פי ההוראות שתקבל. אם המערכת מוכנה להיכנס למצב Ready to Commit, היא רושמת ביומן האירועים שלה את העובדה הזאת ומחזירה תשובה חיובית למערכת המתאמת. אם המערכת אינה יכולה להיכנס למצב Ready to Commit, היא תבצע ROLLBACK ותחזיר תשובה שלילית למערכת המתאמת. אם המערכת המתאמת קיבלה תשובות חיוביות מכל המערכות היא תעבור לשלב הבא. אם המערכת המתאמת קיבלה תשובה שלילית אחת או יותר, או שחלף משך זמן מסוים (Timeout) שנקבע מראש ולא קיבלה תשובה, היא תודיע לכל המערכות לבצע ROLLBACK.
- **שלב Commit**: בשלב הזה, שמתחיל רק אם כל המערכות החזירו תשובה חיובית, המערכת המתאמת שולחת את הפקודה COMMIT לכל המערכות הקשורות. במצב זה כל המערכות מבצעות COMMIT ומחזירות תשובה למערכת המתאמת. אם כל המערכות החזירו תשובה חיובית, המערכת המתאמת רושמת ביומן האירועים שלה שהתנועה המבוזרת בוצעה בהצלחה. אם מערכת RDBMS כלשהי לא הצליחה לבצע COMMIT, או שמערכת מסוימת לא החזירה תשובה מעל משך זמן כלשהו, המערכת המתאמת תודיע לכל המערכות לבצע ROLLBACK.

תרשים 16.35 מציג את התנועה המבוזרת ואת שני שלבי הפרוטוקול העוברים בהצלחה, ולכן התנועה המבוזרת מסתיימת בהצלחה.



תרשים 16.35: תנועה מבוזרת עם פרוטוקול 2PC המסיימת בהצלחה.

תרשים 16.36 מציג מצב שבו בעת שלב ההכנה, מערכת RDBMS אחת מחזירה תשובה שלילית ולכן המערכת המתאמת מודיעה לכל המערכות לבצע ROLLBACK.



תרשים 16.36: תנועה מבוזרת עם פרוטוקול 2PC המסתיימת בכישלון.

פרוטוקול 2PC, שפותח ונתמך כיום על ידי רוב יצרני מערכות RDBMS, מבטיח שתנועה מבוזרת לא תפגום באמינות בסיס הנתונים. למרות שפרוטוקול זה נתמך על ידי רוב יצרני המערכות, הוא לוקה בחוסר פופולריות. הסיבות לכך הן :

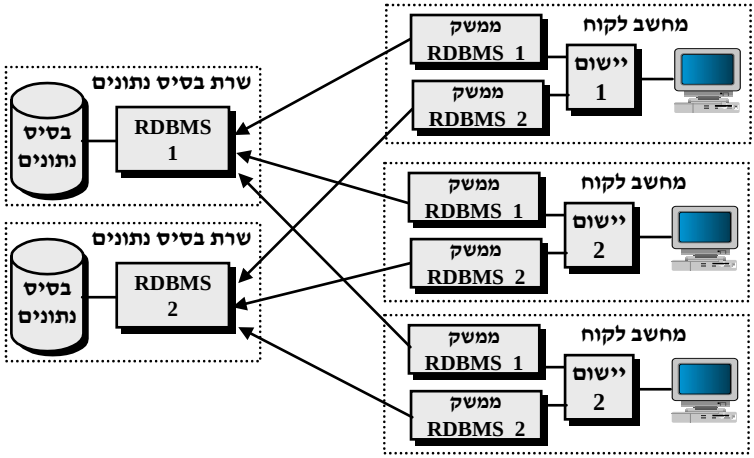
- **עומס נוסף על רשת התקשורת:** הפרוטוקול יוצר תעבורה נוספת ברשת התקשורת וככל שיש יותר מערכות RDBMS המשתתפות באותה תנועה, תעבורה זו גדלה.
- **זמן תגובה ארוך:** התנועה המבוזרת חייבת להמתין עד לסיום העדכון של כל בסיסי הנתונים לפני שהיא יכולה להמשיך בעיבוד. מכיון שמעורבת כאן גם רשת תקשורת, הזמן לסיום תנועה מבוזרת יכול להיות ארוך ובלתי נסבל מנקודת המשתמש במערכת המידע.
- **רגישות לרשת התקשורת**: תקלה ברשת התקשורת יכולה לגרום לביטול התנועה בגלל Timeout. מאחר ורשתות התקשורת סובלות מבעיות של עומס ולפעמים גם נפילות, פרוטוקול 2PC יכול להביא למספר רב מאוד של ביטול תנועות, כלומר ביצוע ROLLBACK למרות שלא היתה כל תקלה בבסיסי הנתונים שהשתתפו בתנועה.

שיטת עבודה זו, הנקראת גם השיטה הסינכרונית, נמצאת בשימוש מוגבל מאוד. רוב הארגונים בחרו להתמודד עם בעיית הביזור של תנועות מבוזרות בשיטות אחרות של שכפול והעתקת נתונים. שיטות אלו פחות אמינות מפרוטוקול 2PC, אולם הן הוכיחו את עצמן כמעשיות יותר. שיטות אלו מתוארות בסעיף האחרון של פרק זה.

ארכיטקטורות לקישוריות בסביבה הטרוגנית

אתגר הגישה לנתונים בסביבה מבוזרת והטרוגנית גדול, וגם חיוני מאוד. הסביבה ההטרוגנית המבוזרת היא **המציאות** בה פועלות רוב מערכות המידע המודרניות. נתבונן בארגון שמיישם מערכת ERP, כגון SAP R/3 או Oracle Application, ומשתמש בבסיס הנתונים Oracle. בנוסף, הארגון מפעיל מערכת מידע מרכזית המשרתת אלפי משתמשים במחשב המרכזי מסוג יבמ, תוך שימוש בבסיס הנתונים DB/2, ורכש לאחרונה מערכת לניהול קשרי לקוח הפועלת על שרתי NT עם בסיס נתונים SQL Server. סביר להניח שחלק מהיישומים צריכים לגשת לנתונים המנוהלים על ידי יישומים אחרים, כלומר לגשת לבסיסי נתונים שונים הפועלים על פלטפורמות חומרה שונות. כיצד מאפשרים גישה לנתונים בסביבה כזאת?

במודל העבודה של בסיס הנתונים המבוזר הצגנו את אחת האפשרויות לגישה מתוך יישום לקוח למספר בסיסי נתונים שונים. מודל עבודה זה מחייב התקנה במחשב הלקוח של רכיב Networked SQL של כל יצרן בסיס נתונים אליו היישום מבקש לגשת.



תרשים 16.37: עבודה ישירה עם ממשקי תכנות של בסיסי נתונים שונים.

בארכיטקטורה זו על תוכניות היישום השונות להתמודד עם הקושי וחוסר האחידות של בסיסי הנתונים השונים. בין אלה נוכל למנות: תחביר שונה של פקודות SQL, קודי שגיאה שונים, צורת ייצוג נתונים שונה, תמיכה שונה בטיפוסי נתונים זהים, בעיות בהגדרת סדר

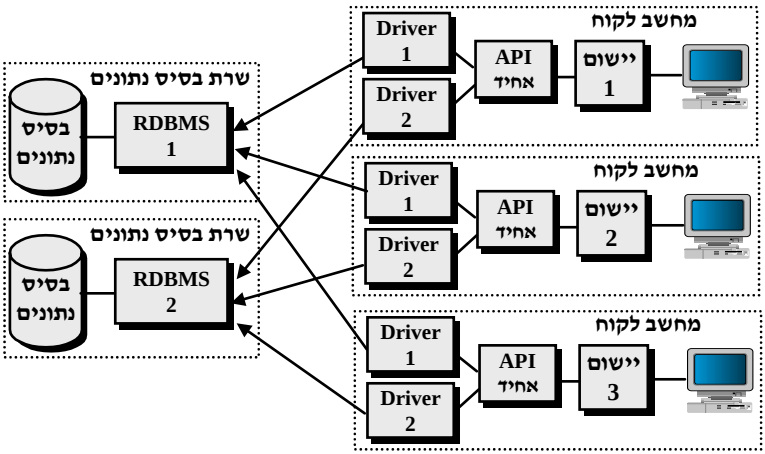
המיון, שיטות התקשרות (Connect) שונות, זיהוי משתמשים והרשאות שונות בכל בסיס נתונים. נדגיש שלמרות שרוב מערכות RDBMS המסחריות תומכות בתקן SQL2, מספר ההבדלים ביניהן גדול. חלק מההבדלים נובע מהעובדה שהתקן לא הגדיר נושאים מסוימים, וחלק מההבדלים נובע מצורת המימוש השונה של היצרנים.

במשך השנים, ככל שהצורך בתמיכה ביישומים הפועלים בסביבה מבוזרת הלך וגדל, החלו להתפתח מספר ארכיטקטורות המנסות להתמודד עם קושי זה. שלוש הארכיטקטורות הנפוצות הן הממשק האחיד, שער יציאה אחיד והפרוטוקול האחיד. נסקור בקצרה כל אחת משלושתן.

ארכיטקטורת הממשק האחיד (Common Interface)

ארכיטקטורה זו מבוססת על העקרון של ממשק גישה אחיד במחשב הלקוח. ממשק אחיד זה מתמודד עם כל מורכבות העבודה מול בסיסי נתונים שונים. תוכנית היישום משתמשת **בממשק תכנות אחיד** (Common API) לקבלת השירותים מבסיסי הנתונים השונים. לדוגמה, תוכנית היישום מתקשרת, כלומר יוצרת Connection, משתמשת באותה פונקציה כדי להתקשר לבסיס נתונים Oracle הפועל על שרת Unix, לבסיס הנתונים DB/2 הפועל על מחשב יבמ עם מערכת הפעלה OS/390 או לבסיס הנתונים SQL Server הפועל על שרת NT, וזאת למרות השוני הקיים בפונקציה המבקשת את יצירת ההתקשרות בכל אחת ממערכות RDBMS אלו.

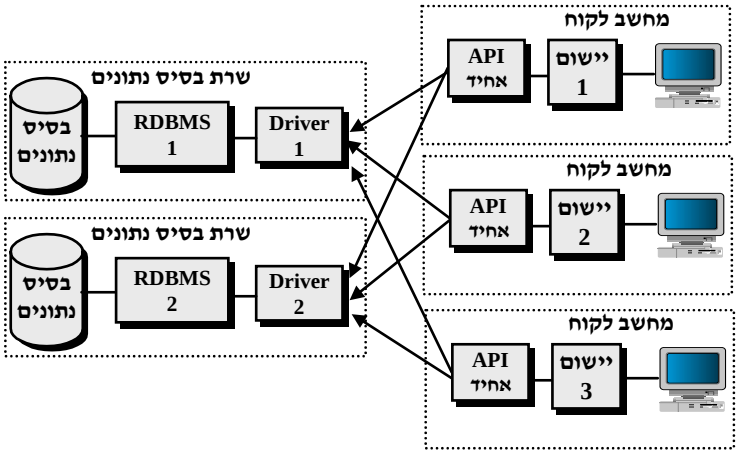
הרעיון המרכזי מאחורי ארכיטקטורה זו הוא לבנות ממשק אחיד ומאחוריו אוסף של **תוכניות שירות** הקרויות **Drivers**. לכל בסיס נתונים יש תוכנית שירות ייחודית המטפלת בהמרת הפונקציה מהמבנה האחיד, כפי שהיא נתמכת על ידי הממשק האחיד למבנה הייחודי של הפונקציה כפי שכל מערכת RDBMS תומכת בו. לאחר שמוגדרת הפונקציה בממשק האחיד, באחריות יצרן המערכת, או יצרן צד שלישי, לבנות תוכנית שירות המבצעת את ההמרות מהממשק האחיד אל הממשק הייחודי של המערכת המסוימת.



תרשים 16.38: ארכיטקטורת הממשק האחיד.

ארכיטקטורת הממשק האחיד מבוססת על המכנה המשותף הנמוך ביותר של השירותים השונים שכל מערכת RDBMS תומכת בהם ועל הגדרת פנייה אחידה להפעלת פונקציה זו.

ניתן לבנות ארכיטקטורה זו גם בצורה שבה תוכנית השירות המבצעת את ההמרות מהממשק האחיד לממשק הספציפי של המערכת יותקן בצד השרת, וכך יוריד מעומס ההמרות במחשב הלקוח.



תרשים 16.39: ארכיטקטורת הממשק האחיד עם Driver בצד השרת.

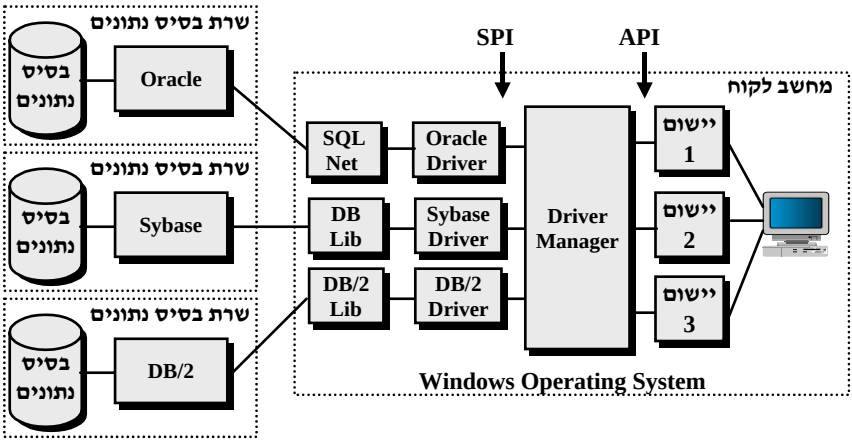
ארכיטקטורה זו פחות נפוצה, אם כי יש מספר מוצרים העושים שימוש בצורת עבודה זו ולפעמים בשילוב מסוים בין שתי השיטות.

ארכיטקטורת הממשק האחיד הופיעה לראשונה במוצר Data Access Language (DAL) של חברת Apple שמפעיל תוכניות שירות מסוג Driver הנמצאות בצד השרת. שפת הגישה האחידה שולבה אל תוך מערכת ההפעלה MAC/OS ואיפשרה למפתחי היישומים גישה אחידה למקורות נתונים שונים. בשנת 1991 הופיע המוצר EDA/SQL של חברת IBI. הממשק האחיד של EDA/SQL מתבסס על ממשק התומך בגירסה מצומצמת וחלקית של תקן SQL1. המיוחד במוצר זה שהוא מאפשר גישה באמצעות אותו ממשק גם למערכות שאינן מבוססות על המודל הטבלאי. משמעות הדבר היא שניתן לבצע פקודת SELECT על טבלה המנוהלת כקובץ בבסיס נתונים לא טבלאי כגון IMS. גם במוצר זה, רוב העיבוד מתבצע בצד השרת.

יצרני מערכות RDBMS החלו גם הם לפתח ממשקים למערכות RDBMS שלא מתוצרתם. לדוגמה חברת Oracle פיתחה ממשק גישה ל-DB/2, חברת Sybase פיתחה ממשק גישה ל-Oracle וכד'. שיטה זו, שבה כל יצרן מנסה להתמודד עם הממשקים של מערכות מתחרות, הוכיחה את עצמה כבעייתית בגלל הקושי בשיתוף הפעולה ביניהם וכן בגלל השדרוגים התכופים של כל מערכת. שדרוגים אלה חייבו את היצרנים, וכמובן לאחר מכן את הלקוחות, להתקין כל הזמן גרסאות חדשות של הממשק.

כדי להתמודד עם הנושא של קישוריות בין בסיסי נתונים הטרוגניים ומתן אפשרות לעבודה שיתופית (Interoperability) בסביבות פתוחות וזרות ובעקבות לחצים של לקוחות גדולים, הקימו מספר יצרני מערכות RDBMS את הארגון SQL Access Group, ובקיצור SAG. בארגון שהוקם בשנת 1989 היו חברים רוב היצרנים המובילים של מערכות RDBMS ויצרני תוכנה וחומרה מובילים, למעט יבמ שהחליטה לא להצטרף אליו. יעדו של הארגון היה להציע תקן אחיד לממשק הגישה לבסיסי נתונים שונים. קבוצות העבודה החלו בהגדרת ממשק אחיד, ממשק הנקרא SQL Call Level Interface, ובקיצור SQL CLI.

מיקרוסופט, החברה גם היא בארגון SAG, החליטה לקחת את אפיון הממשק ולפתח בעצמה ממשק כזה עבור מערכות ההפעלה שלה. ממשק זה נקרא Open DataBase Connectivity, ובקיצור **ODBC**. ממשק זה פועל בצד הלקוח ולא בצד השרת.



תרשים 16.40: תצורת ממשק ODBC הפועל מול שלושה בסיסי נתונים שונים.

כפי שניתן לראות מתרשים 16.40, ממשק ODBC מורכב משני רכיבים: מנהל תוכניות השירות (Driver Manager) המספק API אחיד לתוכניות היישום ומפעיל את תוכנית השירות (Driver) הנדרשת, ואוסף של תוכניות שירות (Drivers) שכל אחת מהן מותאמת לבסיס נתונים אחר. בין ה- Driver Manager לבין ה- Driver מוגדר ממשק אחיד בשם Service Provider Interface, ובקיצור - SPI. הגדרת הממשק SPI מאפשרת ליצרנים שונים לבנות תוכניות שירות אלו עבור המערכות שלהם, כך שניתן יהיה להפעיל אותם על ידי Driver Manager ולגשת אל בסיסי הנתונים שלהם מתוך היישום. טעינת תוכניות שירות אלו מתבצעת בצורה דינמית על פי הצורך מתוך ספריית DLL (Dynamic Load Library). תוכנית השירות הנטענת פונה אל רכיב Network SQL הרגיל של כל בסיס נתונים. לדוגמה, תוכנית השירות של Oracle פונה לרכיב SQL*Net שהוא הרכיב הסטנדרטי המאפשר גישה מרחוק לבסיס הנתונים.

ארכיטקטורת ODBC מאפשרת ליצרנים שונים, גם של מערכות שאינן מבוססות על המודל הטבלאי, לבנות את תוכנית השירות הייחודית, ה-Driver, שלה ולאפשר גישה אחידה אל הנתונים. לדוגמה, ניתן לגשת באמצעות ממשק ODBC אל גיליון אלקטרוני Excel או אל בסיס הנתונים של Lotus Notes.

מנקודת מבטו של מפתח היישום, ממשק ODBC נראה כאוסף של פונקציות חדשות, בנוסף לספריה הגדולה ממילא של פונקציות Windows API. להלן מספר דוגמאות של פונקציות בספריית ODBC:

- בשלב ראשון תוכנית היישום תקצה את סביבת ODBC על ידי הפונקציה SQLAllocEnv.
- פתיחת קשר עם בסיס נתונים כלשהו יבוצע על ידי הפונקציה SQLConnect.
- פקודת SQL תבוצע באופן מיידי על ידי הכנסת הפקודה למשתנה והפעלת הפונקציה SQLExecDirect. ניתן גם להכין מראש פקודת SQL בעזרת הפונקציה SQLPrepare, להכניס את הפרמטרים לפקודה לפני ביצועה על ידי הפונקציה SQLSetParam ולבצע אותה על ידי הפונקציה SQLExecute. בצורה זו האופטימיזציה של פקודת SQL מתבצעת פעם אחת בלבד, אך הפעלתה עם פרמטרים שונים יכולה להתבצע מספר רב של פעמים.
- עבור שליפת שורות מטבלה קיימות מספר פונקציות, כגון SQLNumResultCols הסופרת את מספר העמודות בטבלת התוצאה, SQLDescribeCol המחזירה את טיפוס הנתונים והאורך של כל עמודה, או SQLBindCol המבצעת קישור של עמודה עם משתנה של תוכנית היישום.
- שליפת שורות מהטבלה התוצאתית מתבצעת על ידי הפעלה בלולאה של הפונקציה SQLFetch.

ממשק ODBC תומך רק ב-SQL דינמי. כלומר, פקודות SQL עוברות תהליך מיטוב בכל הפעלה של תוכנית היישום. כפי שראינו, ניתן לבצע את המיטוב פעם אחת ולהפעיל את הפקודה מספר רב של פעמים, אולם לא ניתן לשמור את תוצאות המיטוב כמו ב- Embedded SQL.

מיקרוסופט פרסמה את התקן OLE DB המחליף בהדרגה את תקן ODBC שקדם לו. בתקן זה יש הרחבות למולטימדיה ולתמיכה בשירותי OLAP (ניתוח רב-מימדי).

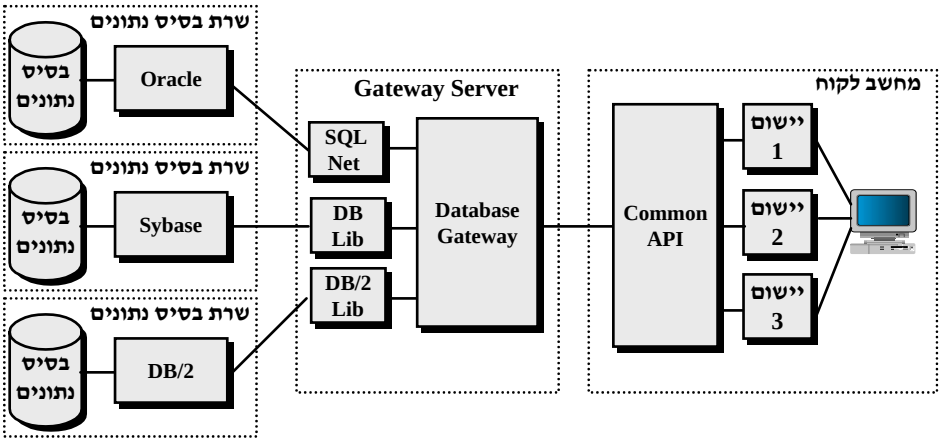
בשנים האחרונות, עם התפוצה הרחבה שזכתה שפת Java, פותח הממשק Java Database Connectivity, ובקיצור JDBC, הפועל גם הוא בארכיטקטורת הממשק האחיד. ממשק זה מאפשר גישה מתוך יישום הכתוב בשפת Java למיגוון רחב של בסיסי נתונים טבלאים.

ארכיטקטורת שער יציאה (Common Gateway)

ארכיטקטורה זו מבוססת על ממשק תכנות אחיד והמרת הממשק האחיד לממשק ספציפי במחשב נפרד ולא במחשב הלקוח. מחשב זה ייקרא Database Gateway Sever והוא מפעיל תוכנה ייעודית לטיפול בממשק העבודה עם בסיסי הנתונים השונים.

היתרון של ארכיטקטורה זו הוא כמובן בהורדת הנטל של המרת הממשקים ממחשב הלקוח, שהוא בדרך כלל תחנת עבודה שולחנית, והעברת נטל זה לשרת ייעודי רב עוצמה.

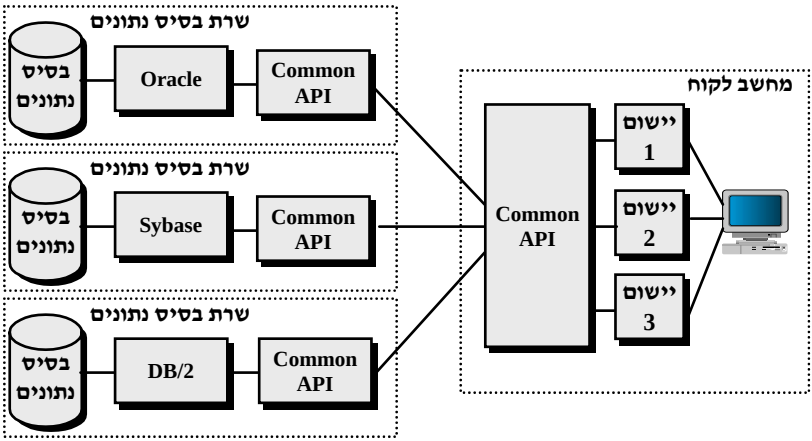
כמובן ששרת זה יכול לבצע מטלות נוספות כגון איזון עומסים, המרת פרוטוקולים לתקשורת בין LAN ל-WAN, טיפול בתנועות הפונות למספר בסיסי נתונים, בדיקות הרשאה, הפעלת פרוצדורות בסיס נתונים, קישוריות מול מחשבים מרכזיים, הפעלת שירותי CICS וכד'. בין המוצרים המובילים המבוססים על ארכיטקטורה זו ניתן למנות את Enterprise Connect של חברת Sybase.



תרשים 16.41: ארכיטקטורת שער יציאה.

ארכיטקטורת הפרוטוקול האחיד (Common Protocol)

ארכיטקטורה זו מעבירה את נטל המרת הממשקים אל שרת בסיס הנתונים, ומבוססת על פרוטוקול אחיד, המאפשר לכל יישום לקוח לפנות בצורה אחידה לכל בסיסי הנתונים השונים.



תרשים 16.42: ארכיטקטורת הפרוטוקול האחיד.

ניתן לראות מתרשים 16.42 שהיישום רואה ממשק אחיד. כלומר, הוא משתמש בקבוצה מוגדרת של פונקציות כדי לבקש שירותי בסיס נתונים, ללא תלות בבסיס הנתונים המסוים. פונקציות אחידות אלו מועברות אל שרת בסיס הנתונים המתאים. שרת בסיס הנתונים מבצע את המרת הפונקציות האחידות אל הפונקציות המסוימות ומטפל בכל הפעולות הדרושות.

חברת יבמ פרסמה תקן לפרוטוקול אחיד שיאפשר גישה מכל יישום לכל אחד מבסיסי הנתונים שלה: DB/2 או SQL/DS המצויים על מחשב מרכזי, DB/2 Universal Database הפועל על שרת RS/6000 ומערכת הפעלה AIX, ו-SQL/400 הפועל על מחשב AS/400. התקן נקרא Distributed Relational Database Architecture, ובקיצור DRDA. לתקן זה יש חשיבות בעיקר אצל לקוחות המתבססים על בסיסי הנתונים השונים של חברת יבמ ומבקשים להבטיח Interoprability פשוטה יחסית.

בנוסף ליבמ, גם ארגון התקינה הבינלאומי, ISO פרסם תקן המבוסס על ארכיטקטורת הפרוטוקול האחיד וקראה לו Remote Database Access או בקיצור RDA. בשלב זה עדיין יש הסכמה רק על קבוצה חלקית של פונקציות ורק ב-SQL דינמי. לא כל היצרנים הכריזו על תמיכה בתקן ISO/RDA.

ניהול העתקים (Copy Management)

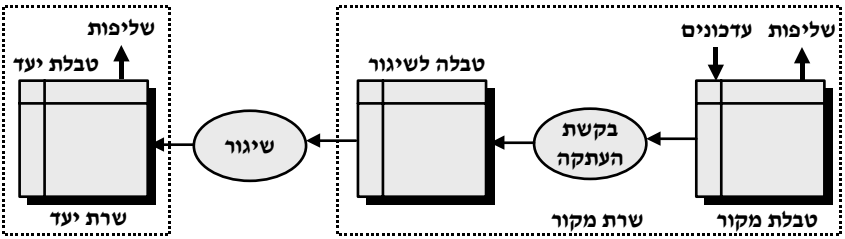
עד כאן הנחנו שכל נתון מנוהל בבסיס נתונים אחד בלבד, והבעיה העיקרית היתה כיצד להבטיח שתוכנית היישום תוכל להגיע אל בסיסי נתונים מרוחקים דרך רשתות תקשורת מקומיות ומרחביות. לפעמים, הפתרון הוא לאו דוקא לאפשר לתוכנית יישום לגשת לבסיסי נתונים מרוחקים אלה, אלא הפוך – להביא את הנתונים אל תוכנית היישום. פתרון זה מבוסס על ביזור הנתונים באמצעות טכניקות של **העתקת נתונים** (Data Copy) וניהולם במספר מחשבים שונים. ההנחה הבסיסית היא שהנתונים המועתקים אינם מתעדכנים אלא משמשים רק למטרות של שאילתות ודוחות. העתקת הנתונים ממקום אחד למשנהו יכולה לפתור בעיות רבות:

- **שיפור ביצועים:** במקום לפנות אל הנתונים דרך רשתות התקשורת, העתקתם למחשב מקומי יכולה להביא לשיפור משמעותי בביצועים של תוכנית היישום. תוכנית זו פונה אל נתונים הנמצאים באותו מחשב שבו היא פועלת ולכן הביצועים יהיו טובים יותר.
- **היעדר הפרעה למערכות התפעוליות:** המערכות התפעוליות של הארגון משרתות מספר רב מאוד של משתמשים ולכן הן פועלות במחשבים חזקים יחסית. מחשבים אלה עמוסים בפניות של תוכניות יישום מרוחקות המתחרות בפניות של תוכניות היישום המקומיות. דבר זה נכון במיוחד אם תוכניות הישום המרוחקות מבקשות בעיקר לבצע שאילתות, לעיתים מורכבות מאוד, ולא לעדכן את הנתונים.

הטיפול בנושא העתקת נתונים קיבל תאוצה רבה עם הופעת תפישת מחסן הנתונים, אשר מבוססת על גזירת הנתונים מהמערכות התפעוליות והעתקתם לפלטפורמת מחשוב ייעודית. נסקור בקצרה מספר טכניקות לניהול העתקים.

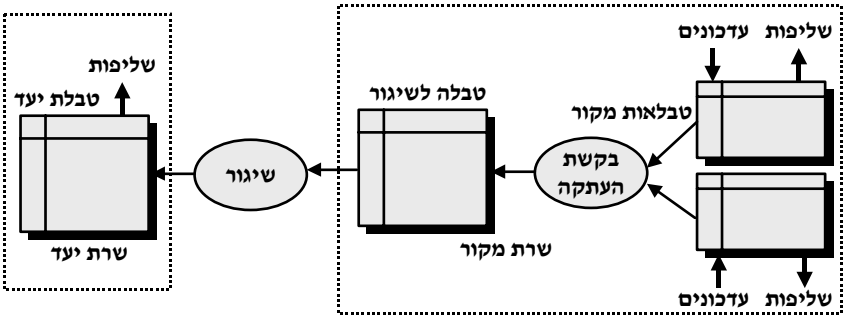
העתקה פשוטה ליעד אחד

המקרה הפשוט ביותר הוא **העתקה פשוטה** (Simple Extract) של טבלה אחת או יותר בשלמות ממקור כלשהו אל יעד כלשהו.



תרשים 16.43: העתקה פשוטה של טבלה.

כפי שניתן לראות, ההעתקה מורכבת משני שלבים: שלב יצירת הטבלה לשיגור בשרת המקור ושלב שיגור הטבלה אל שרת היעד. השיגור יכול להתבצע דרך הרשת על ידי שימוש בפרוטוקול העברת קבצים כמו FTP. בקשת ההעתקה יכולה להיות מורכבת יותר ולהתבצע על ידי פקודת SQL הפונה למספר טבלאות ומייצרת טבלה לשיגור שהיא במבנה שונה מטבלת המקור.



תרשים 16.44: העתקת טבלת יעד במבנה שונה מטבלת המקור.

טבלת המקור מתעדכנת באופן שוטף ואילו העותק של הטבלה מהווה רק **צילום מצב רגעי** (Snapshot) של טבלת המקור ולכן התוכן שלו נכון לנקודת זמן מסוימת. עותקי הטבלה משמשים **לשליפות בלבד** (Read Only Copy). לעותק של הטבלה המקורית נקרא **טבלה מועתקת** (Duplicated Table) או בקיצור עותק.

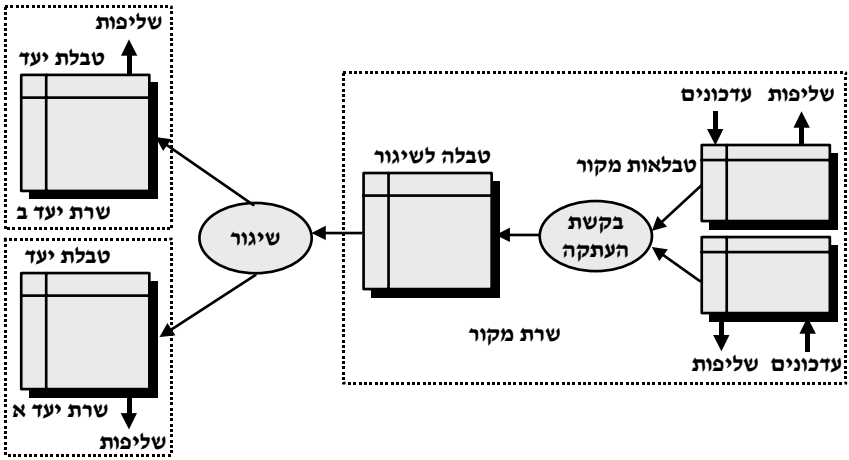
בקשת ההעתקה יכולה להתבצע על ידי פקודת SQL, אוסף של פקודות SQL המשובצות בתוך תוכנית יישום כלשהי או פרוצדורת בסיס נתונים המופעלת ומייצרת את הטבלה לשיגור. לעיתים יש צורך לבצע עיבודים מורכבים של הנתונים לקראת העתקתם. לדוגמה, בסביבה של מחסן נתונים, תהליך גזירת הנתונים מהמערכות התפעוליות הוא מורכב ודורש בחלק גדול מהמקרים גם ביצוע של המרות קודים, שינוי מונחים וכד'.

אחת הבעיות של המשתמשים בשרתים שונים, הפונים לעותקים של טבלה, היא לדעת לאיזו נקודת זמן העותק איתו הם עובדים הוא נכון. מכיון שהטבלה המקורית מתעדכנת באופן שוטף, למועד האחרון של יצירת העותק יש חשיבות. אחת השיטות המקובלות היא להוסיף טבלה מיוחדת המנוהלת בשרת היעד ומכילה את תאריך הנכונות של העותק. לתאריך זה מקובל לקרוא **חותמת הזמן** (Time Stamp) של העותק. תדירות החידוש (Refresh Frequency) של העותק נקבעת על פי הצרכים וכמובן על פי כמות העדכונים של טבלת המקור. אם טבלת המקור כמעט ואינה מתעדכנת, תדירות החידוש תהיה נמוכה יחסית.

ניתן להגדיר **חידוש אוטומטי** (Automatic Refresh), שלפיו באופן תקופתי ואוטומטי מופעלת פרוצדורה המבצעת את השליפה והשיגור של העותק אל היעד. אפשרות אחרת היא **חידוש ידני** (Manual Refresh) כלומר פרוצדורת השליפה והשיגור מופעלת בצורה יזומה ולא באופן אוטומטי. לדוגמה, תוכנית היישום בשרת היעד יכולה לבדוק את נכונות הטבלה ובמידת הצורך ליזום מהלך של קבלת עותק חדש, אם מתברר שהעותק התיישן.

העתקה פשוטה למספר יעדים

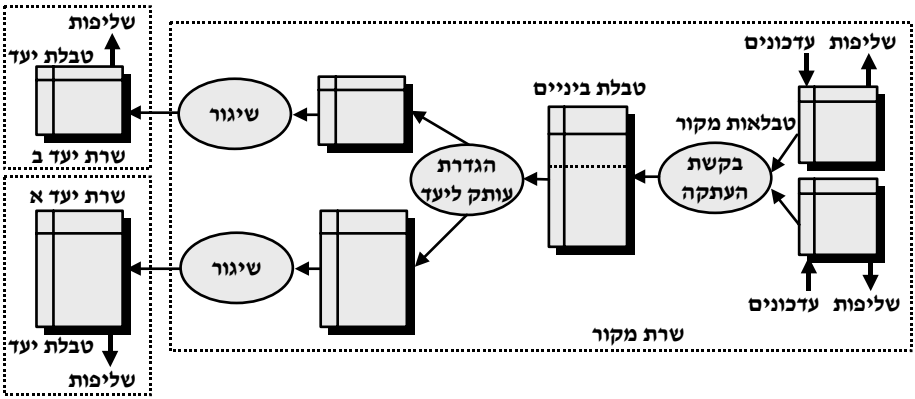
עד כאן הראינו מצב של שיגור הטבלה ליעד אחד בלבד. במציאות ייתכן שנרצה להעתיק את הטבלה למספר שרתים שונים. בדוגמה של המכללה, אם נניח שיש לה מספר קמפוסים ברחבי העיר ובכל קמפוס יש שרת נפרד, ייתכן שנרצה להעתיק את טבלת הקורסים לכל אחד מהשרתים האלה.



תרשים 16.45: העתקת טבלה אחת למספר יעדים שונים.

העתקת מקטעים של טבלאות (Table Partitioning)

המצב הולך ומסתבך כשרוצים להעתיק את הטבלה למספר יעדים שונים, אבל עם תוכן שונה לכל יעד. לדוגמה, נרצה להעתיק לכל קמפוס של המכללה רק את הקורסים הניתנים באותו קמפוס. לשיטת חלוקה זו של טבלה אחת למספר טבלאות קוראים **פיצול טבלאות** (Table Partitioning). לטבלה המתקבלת כתוצאה מתהליך הפיצול קוראים **מקטע** (Partition), מאחר והיא מהווה מקטע מסוים של הטבלה המקורית. ניתן לפצל טבלה בצורה אופקית לקבלת **מקטע אופקי** (Horizontal Partition) המכיל שורות מסוימות בלבד של הטבלה, או בצורה אנכית לקבלת **מקטע אנכי** (Vertical Partition) המכיל עמודות מסוימות של טבלה. ניתן גם לבצע פיצול משולב לקבלת **מקטע מעורב** (Hybrid Partition) המכיל שורות ועמודות מסוימות של טבלת המקור.



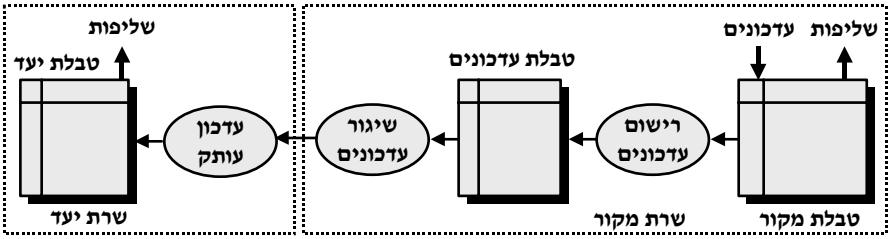
תרשים 16.46: העתקת מקטעים שונים ליעדים שונים.

כפי שניתן לראות, אם נרצה לבנות מקטעים שונים ולשגר כל מקטע ליעדים שונים, נושא ניהול ההעתקים מתחיל להיות מורכב. עלינו לדעת איזה עותק נמצא היכן, למתי הוא מעודכן, האם השיגור עבר בהצלחה, מתי לשגר אותו מחדש וכד'.

הפצת עדכונים בלבד (Incremental Updates)

אחת השאלות שעולות מייד היא מה קורה כאשר טבלאות המקור מתעדכנות. ניתן אז לייצר מחדש את העותק של טבלת המקור המבוקשת ולשגר אותה אל היעד, תוך כדי דריסת הגירסה הקודמת של הטבלה. כפי שנאמר, לשיטה זו נקרא **חידוש העותק**. חידוש עותק הינו תהליך פשוט יחסית, אבל מה קורה אם הטבלה לשיגור היא גדולה מאוד, למשל טבלה המכילה מספר עשירות מיליוני שורות, או שמספר העדכונים שבוצעו בטבלת המקור קטן מאוד. תהליך החידוש והחלפת העותק הוא תהליך ארוך ויכול להעמיס על רשת התקשורת בצורה משמעותית. אם מספר העדכונים שבוצעו בטבלת המקור הוא קטן יחסית, יש בזבוז משאבים גדול בתהליך החידוש של העותק.

במקום לשלוח מחדש כל פעם את כל הטבלה, ניתן להעתיק את הטבלה פעם אחת בלבד ומכאן ואילך לבצע **עדכונים תוספתיים** (Incremental Updates) על ידי העברת קובץ המכיל את השינויים בלבד (Delta File) שבוצעו בטבלת המקור מאז העדכון התוספתי האחרון ועדכון העותק בשינויים בלבד.



תרשים 16.47: הפצת שינויים בלבד.

כפי שניתן לראות מתרשים 16.47, שיטה זו מורכבת יותר, מאחר והיא מחייבת זיהוי השינויים בטבלת המקור מאז השיגור האחרון ועדכון העותקים השונים בכל השינויים. איתור הנתונים שעודכנו (Changed Data Capturing) הינו תהליך מורכב, מאחר ולפעמים קשה מאוד לזהות את השינויים בטבלאות מקור. למשל איך נדע איזה שורות בוטלו איזה שורות חדשות התווספו ואיזה שורות עודכנו.

מנגנונים לשכפול טבלאות (Replication Servers)

הצורך בניהול נתונים בסביבה מבוזרת והעובדה שהתקוות שהיו לגבי מערכות ניהול בסיסי נתונים מבוזרות לא התגשמו, הביא את הארגונים לאמץ שיטות של העתקת טבלאות כפתרון חלקי לבעיית הביזור. כפי שהראינו, העתקת הטבלאות יכול להפוך לנושא מורכב מאוד. יצרנים שונים החלו לפתח פתרונות של ניהול העתקות כמענה חלקי לביזור הנתונים. חברת Sybase היתה אחת החלוצות בפיתוח שיטת העתקת טבלאות לשרתים מרוחקים עם המוצר Replication Server. עם הזמן, הציעו רוב יצרני מערכות RDBMS ויצרני צד שלישי, פתרונות לנושא ניהול העתקות, כמו למשל המוצר Data Propagator של חברת יבמ ואחרים. פתרונות אלה נבדלים ביניהם בשירותים השונים שהם מציעים, בתמיכה במיגוון מקורות נתונים, בתמיכה בפלטפורמות חומרה שונות וכד'.

נסקור בקצרה את המאפיינים העיקריים של מנגנוני שכפול טבלאות, מבלי להתייחס באופן ספציפי למוצר כלשהו, אם כי חלק גדול מהתכונות המתוארות מקורם ב- Replication Server של חברת Sybase. מנגנון השכפול מורכב ממספר מרכיבים:

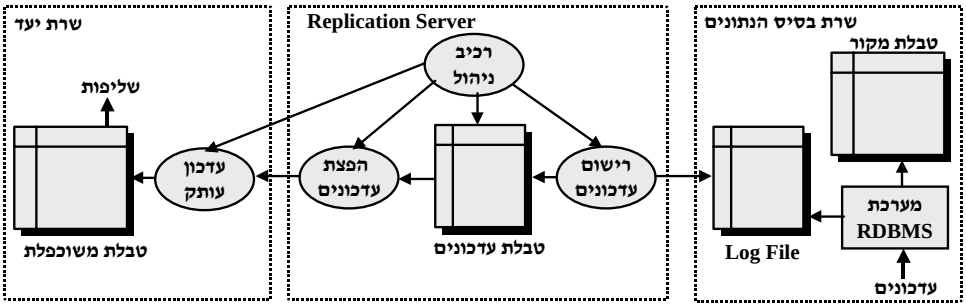
- **רכיב הניהול:** רכיב זה מאפשר להגדיר איזה טבלאות יש לשכפל, באיזה תדירות יש לשכפל כל טבלה, מהם היעדים אליהם יש להעביר את הטבלאות המשוכפלות, מה סטטוס השכפול של כל טבלה, איזה עיבודים יש לבצע על הנתונים לפני הפצתם וכד'.

רכיב זה מיועד למנהל המערכת האחראי על נושא שכפול הנתונים בארגון. רכיב זה יכול לרוץ על שרת נפרד או על אותו שרת שבו פועל בסיס הנתונים.

- **רכיב זיהוי השינויים:** רכיב זה מזהה את השינויים שבוצעו בטבלאות המקור. השיטה הנפוצה ביותר היא על ידי בדיקת קובץ יומן האירועים (Log File) של בסיס הנתונים ואיחזור השורות שעודכנו על ידי תנועות שהסתיימו בהצלחה, כלומר ביצעו Commit. כל השינויים בנתוני הטבלאות שיש לשכפל נשלפים מתוך יומן האירועים ונכתבים לטבלה נפרדת, טבלת העדכונים. טבלה זו מכילה את כל השינויים שבוצעו בטבלאות המקור ושיש להפיצם.

- **רכיב הפצת השינויים**: רכיב זה מטפל בהפצת העדכונים ליעדים השונים. הרכיב מופעל על פי אירועים: אם עבר זמן מסוים מאז ההפצה האחרונה, אם הצטברו מעל מספר מסוים של עדכונים עבור טבלה מסוימת, הפעלה רציפה (As soon as possible) שמשמעותה שמייד לאחר עדכון טבלת המקור יש להתחיל בהפצת השינוי וכד'. רכיב זה שולף את השינויים מטבלת העדכונים, יוצר קשר עם השרת המרוחק ומעביר את כל השינויים אל השרת המרוחק. אם השרת המרוחק אינו זמין או שיש בעיית תקשורת, ניתן להגדיר את הכללים מתי לנסות להפיץ פעם נוספת את השינויים.

- **רכיב עדכון העותקים המרוחקים:** רכיב זה פועל על השרת שבו נמצאת טבלת היעד. הוא מקבל את כל העדכונים ומבצע את עדכון הטבלה המועתקת.



תרשים 16.48: עיקרון הפעולה של שרת שכפולים.

עד כאן הנחנו שהעותקים אינם משמשים לעדכונים אלא לאיחזור נתונים בלבד. עקרונית ניתן לאפשר ביצוע שינויים לעותקים, אולם אז כמובן עולה מייד הסוגיה כיצד שומרים על סינכרון טבלת המקור עם העותק. אם מותר לעדכן גם את הטבלה המקורית וגם את הטבלה המשוכפלת, מנגנון השכפול צריך לשמור על סינכרון בין הטבלאות. עליו להפיץ את השינויים שבוצעו בטבלת המקור אל הטבלה המשוכפלת וממנה, ולפתור סתירות שעלולות להתעורר כתוצאה מכך שמותר לעדכן שני עותקים של אותה טבלה.

סיכום

אנו נמצאים בעיצומו של תהליך ביזור הארגונים וביזור מערכות המידע שלו. סביבות מבוזרות והטרוגניות הן המציאות שבה פועלים כיום ויפעלו בעתיד רוב הארגונים. הצורך בביזור תהליכים ונתונים הולך וגובר כל הזמן ולכן טכנולוגיית בסיסי הנתונים חייבת להתמודד עם מציאות זו ולספק פתרונות למצב זה.

בפרק זה סקרנו את האתגרים שביזור התהליכים והנתונים מציב בפני טכנולוגיית בסיסי הנתונים ומה התשובות שלה כיום לאתגרים אלה. תפישת שיתוף הפעולה – Interoperability – והיכולת של יישומים לגשת למיגוון מקורות נתונים הוא חשוב. למרות המורכבות, קיימות כבר כיום מספר ארכיטקטורות המאפשרות גישה למקורות שונים של נתונים. יצירת תקנים שיבטיחו יכולת זו בעתיד היא חשובה ולכן העבודה עדיין רבה. השוני בין המערכות הוא גדול ולכן גם האתגר.

אין כל ספק שהסביבה המבוזרת מציבה מספר אתגרים רציניים ליצרני מערכות ה- RDBMS וחלק מהתשובות שמערכות אלו מספקות כיום עדיין לא מספקות. היצרנים וארגוני התקינה השונים ימשיכו להשקיע בנושא חשוב זה וסביר להניח שעם השנים נהיה עדים לשיפורים שיאפשרו ביזור קל ונוח יותר וגישה קלה יותר לנתונים המבוזרים בין פלטפורמות חומרה שונות.

שאלות חזרה ותרגילים

שאלות חזרה

1. מה היתרון של העברת פקודות SQL על ידי RPC לעומת העברת כל הפקודות ממחשב הלקוח למחשב השרת?
2. הסבר כיצד פועל מנגנון IPC ומה חשיבותו.
3. הסבר את המרכיבים העיקריים של ארכיטקטורת שרת/לקוח. מדוע ארכיטקטורה זו מורכבת יותר מארכיטקטורת המחשב המרכזי?
4. הסבר כיצד פועל פרוטוקול 2PC ותאר את שלביו השונים.
5. מה ההבדל בין תנועה מבוזרת לבין בקשה מבוזרת?
6. הסבר את העקרונות של שלוש הארכיטקטורות לקישוריות בסביבה הטרוגנית מבוזרת. הצבע על היתרונות ועל החסרונות של כל אחת מהן.
7. מה ההבדל בין בסיס נתונים מבוזר לבין ניהול מבוזר של עותקי טבלאות?
8. הסבר את עקרונות הפעולה של מנגנון שכפול טבלאות.

תרגילים

כפי שראינו בפרק זה, קיימים הבדלים רבים בין בסיסי הנתונים הטבלאיים השונים ולכן קיים צורך בפיתוח Gateways. קחו שני בסיסי נתונים טבלאיים המוכרים לכם (לדוגמה Oracle ו- Sybase או SQL Server) ונסו למצוא את ההבדלים ביניהם. אם אין לכם נתונים, פנו אל הספרות של מערכות RDBMS.

פרק 17

בסיסי נתונים מבוזרים (Distributed Databases)

1. מבוא
2. מהו בסיס נתונים מבוזר (Distributed Database)
3. יתרונות וחסרונות של בסיס נתונים מבוזר
4. מערכות מבוזרות הומוגניות והטרוגניות
5. ארכיטקטורה של מערכת מבוזרת
6. רמות השקיפות בבסיס נתונים מבוזר
7. פיצול טבלאות (Table Fragmentation)
8. הקצאת מקטעים בבסיס נתונים מבוזר
9. תהליך בניית בסיס נתונים מבוזר
10. שימוש בבסיס נתונים מבוזר
11. עדכון בסיס נתונים מבוזר עם שכפול
12. קטלוג המערכת בסביבה מבוזרת
13. אמינות בסיס נתונים מבוזר (Distributed Data Integrity)
14. כללי Date להגדרת בסיס נתונים מבוזר
15. סיכום

מבוא

בפרק זה נציג את המערכות לניהול בסיסי נתונים טבלאים מבוזרים ונבחן מספר סוגיות הקשורות לניהול בסיסי נתונים מבוזרים. הביזור של הארגונים המודרניים והופעת סביבות המחשוב המבוססות על שרתים זולים יחסית, רשתות תקשורת מקומיות מהירות ומחשבים אישיים על שולחנם של רוב העובדים, יצר לחץ על יצרני מערכות RDBMS לאפשר ניהול מבוזר של נתונים. המערכות לניהול בסיסי נתונים שקדמו למערכות הטבלאיות ומבוססות על מודלים כגון המודל ההיררכי או הרשתי, התגלו כלא מתאימות לעבודה בסביבה מבוזרת בגלל שתי סיבות עיקריות:

- ממשק העבודה בין מערכת DBMS לתוכנית היישום מבוסס על רשומה בודדת ולכן שימוש במודל זה בסביבה מבוזרת עלול לגרום לעומס בלתי סביר ברשת התקשורת.
- מערכות אלו מתבססות על מצביעים למימוש קשרים בין נתונים, ואלה התגלו כבעיתיים מאוד כאשר יש צורך ליצור קשרים בין נתונים המנוהלים בשרתים שונים.

לעומת המערכות הקודמות, מערכות RDBMS מבוססות על שפת SQL המטפלת בו-זמנית במספר רב של שורות וטבלאות, מקטינות באופן משמעותי את תקורת התקשורת ומנהלות את הקשרים בין הנתונים על פי הערכים בטבלאות ולא על ידי מצביעים. זהו הרקע שהביא לכך שההתפתחות במערכות בסיסי נתונים מבוזרים החלה רק לאחר התבססותה של הטכנולוגיה הטבלאית.

במשך השנים ניסו יצרני מערכות RDBMS לשכלל את בסיסי הנתונים שלהם ולהפוך אותם למערכות מבוזרות: Distributed RDBMS, ובקיצור DRDBMS. המטרה היתה לבנות מערכת שתאפשר ביזור **שקוף לחלוטין** של הנתונים ברחבי הארגון. התברר שהאתגר הטכנולוגי בהכנסת מימד הביזור בניהול הנתונים גדול מאוד. מה שלא פחות חשוב, התברר שהוא אינו עונה על המציאות שהלכה והתהוותה – סביבת המחשוב המודרנית אינה הומוגנית ומעטים הם הארגונים שמבססים את כל ניהול הנתונים שלהם על מערכת מתוצרת יצרן אחד בלבד. בפרק הקודם הסברנו את הפתרונות לביזור נתונים, פתרונות שאינם מבוססים על מערכת DRDBMS, אלא על קישוריות בין מערכות RDBMS שונות ועל יכולת שיתוף הפעולה ביניהן (Interoperability). למרות שפתרונות הקישוריות אינם מהווים פתרון מלא לנושא ביזור הנתונים ואינם עומדים ביעד של שקיפות מלאה לביזור הנתונים, התברר שהם מספקים תשובות סבירות למציאות.

בפרק זה נסקור את הרעיונות העיקריים עליהם מבוססים מערכות DRDBMS למרות שאין כיום אף מערכת מסחרית התומכת באופן מלא בכל הרעיונות האלה. סביר להניח שעם השנים נהיה עדים לשיפורים במערכות המסחריות ולמימוש חלק מרעיונות אלה. המטרה העיקרית בהצגת נושא זה היא להציג בפני הקורא את תפישת ביזור הנתונים ואת הסיבות לאתגר הגדול במימושה.

בפרק זה

- נגדיר מהו בסיס נתונים מבוזר ומהי מערכת לניהול בסיס נתונים מבוזר.
- נסקור את היתרונות והחסרונות של מערכות לניהול בסיסי נתונים מבוזרים.
- נסביר את ההבדל בין מערכת מבוזרת הטרוגנית לבין מערכת מבוזרת הומוגנית.
- נציג ארכיטקטורה כללית של מערכת לניהול בסיס נתונים מבוזר ונבהיר את העקרונות עליהם מבוססת ארכיטקטורה זו: פיצול טבלה למקטעים לא חופפים, הקצאת המקטעים באתרים (שרתים) שונים, ניהול עותקים שונים של אותו מקטע במספר שרתים שונים, ניהול קטלוג מערכת בסביבה מבוזרת ועוד.
- נתאר את תפישת המקטע כבסיס לביזור, ונסקור את האפשרויות השונות לפיצול טבלה אחת למספר מקטעים שונים.
- נסביר את רמות השקיפות השונות שמערכת מבוזרת יכולה לספק למשתמש ונראה כיצד שאילה מסוימת נראית כתוצאה מעבודה ברמות שקיפות שונות.
- נסקור את 12 הכללים להגדרת מערכת מבוזרת אידיאלית, כללים שנכון להיום אף מערכת מסחרית אינה מקיימת בשלמותם.

קוראים המעוניינים להעמיק את ידיעותיהם בנושא זה מופנים לספרם המעולה של Osuzu ו-Valduiez [OV91].

מהו בסיס נתונים מבוזר (Distributed Database)

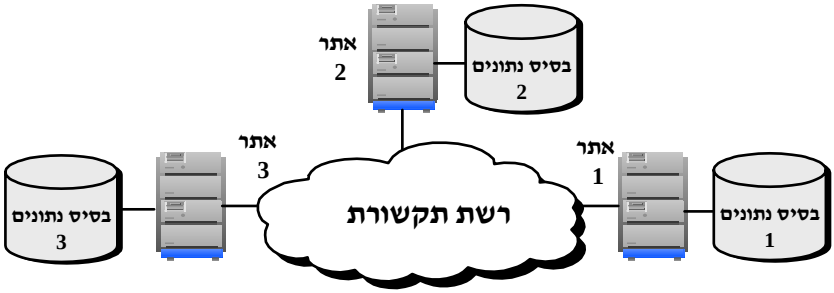
נתחיל בהגדרה של בסיס נתונים מבוזר.

בסיס נתונים מבוזר **Distributed Database**	***בסיס נתונים מבוזר*** **מוגדר כאוסף של נתונים הקשורים ביניהם בקשרים לוגיים כלשהם. הנתונים מנוהלים במספר שרתים שונים המחוברים ביניהם ברשת תקשורת. קיים לפחות יישום אחד המשתמש בנתונים המנוהלים בשרתים השונים.**

נדגיש מספר היבטים הנובעים מהגדרה זו של בסיס נתונים מבוזר:

- **ביזור הנתונים בין מספר מחשבים:** באופן פיסי, בסיס הנתונים המבוזר מנוהל ביותר מאשר שרת אחד. השרתים השונים מחוברים ביניהם ברשת תקשורת כלשהי. השרתים השונים הם עצמאיים ומסוגלים לבצע יישומים מקומיים. זהו כמובן מצב שונה לעומת המצב שבו בסיס הנתונים מאוחסן ומנוהל בשרת אחד בלבד.
- **קשר לוגי בין הנתונים:** הנתונים מנוהלים בשרתים השונים וקשורים ביניהם בקשר לוגי כלשהו. הקשר הלוגי הזה הופך את בסיסי הנתונים למבוזרים, ולא לאוסף של בסיסי נתונים שונים המנוהלים בשרתים שונים.

❖ **יישום גלובלי לעומת יישום מקומי:** קיימת לפחות תוכנית יישום אחת הזקוקה לנתונים המנוהלים בשרתים השונים ולשם כך עליה לפנות דרך רשת התקשורת לשרתים אחרים. ליישומים הזקוקים לנתונים המנוהלים במספר בסיסי נתונים שונים נקרא בשם **יישומים גלובלי**. בנוסף ליישומים הגלובליים, קיימים גם **יישומים מקומיים** המשתמשים בנתונים מקומיים בלבד ואינם זקוקים לנתונים המנוהלים בשרתים מרוחקים. במצב הכללי ביותר, היישום הגלובלי מעדכן את הנתונים ולא רק שולף אותם.



תרשים 17.1: בסיס נתונים מבוזר בין שרתים שונים.

כפי שנראה בדיון שבהמשך הפרק, ביזור בסיס הנתונים בין שרתים שונים אינו פשוט. הוא מציב מספר אתגרים משמעותיים בפני היצרנים.

מערכת לניהול בסיס נתונים טבלאי מבוזר Database Distributed Relational Management System	מערכת תוכנה המאפשרת ניהול בסיס נתונים טבלאי מבוזר.

מערכת DRDBMS מנהלת בסיס נתונים טבלאי שבו חלק מהטבלאות ולפעמים אפילו קטעים שונים של אותה טבלה מבוזרים במספר שרתים שונים. בנוסף ליכולת של מערכות אלו לתת שירות ליישומים מקומיים, הם מסוגלים לשרת גם יישומים גלובליים. זוהי רמה גבוהה יותר של ניהול לעומת מערכת RDBMS מקומית המנהלת את כל הטבלאות בשרת אחד בלבד. בסביבה מבוזרת נדרש מהמערכת יותר מאשר נדרש ממערכת מקומית. נושאים כגון איתור הנתונים ברשת, אופטימיזציה הלוקחת בחשבון גם את העומס על רשת התקשורת, טיפול בתנועות המתבצעות בו-זמנית במספר רב של שרתים ועוד, הן מטלות נוספות הקיימות במערכת DRDBMS לעומת מערכת RDBMS.

כדי להבהיר את הנושא נשתמש בדוגמה פשוטה שתלווה אותנו לאורך פרק זה. נניח שהמכללה מנהלת בסיס נתונים המכיל רק שלוש טבלאות: סטודנטים, קורסים וציונים. למכללה שלושה קמפוסים ברחבי העיר המרוחקים זה מזה גיאוגרפית, ולכן הוחלט לבנות בסיס נתונים מבוזר, כך שלכל קמפוס יהיה את בסיס הנתונים שלו, כלומר טבלאות סטודנטים, קורסים וציונים הרלוונטיות לאותו קמפוס ינוהלו בשרת הממוקם פיסית בקמפוס. יחד עם ביזור הנתונים בשלושת הקמפוסים, קיימות מספר יחידות מרכזיות של

המכללה, כמו למשל המזכירות האקדמית, יחידות רישום סטודנטים ואחרות הזקוקות למידע משולב מכל בסיסי הנתונים. נרצה לקבל מידע למשל, כמה סטודנטים למדו בסמסטר מסוים במכללה, בכמה קורסים למדו יותר מ-100 סטודנטים, מיהם הסטודנטים שקיבלו ממוצע ציונים מעל 95 ועוד.

אילו בסיס הנתונים היה מרכזי בלבד, ניתן היה לענות על דרישות אלו בצורה פשוטה. כאשר בסיס הנתונים מבוזר, הבעיה הופכת למורכבת יותר. הבה נראה כיצד מערכת DRDBMS תטפל בשאילתה אחת החוקרת כמה סטודנטים למדו בסמסטר מסוים במכללה. נניח שהשאילתה מופנית לאחד משלושת השרתים. להלן שלבי הפעולה לקבלת הנתונים:

א. מערכת DRDBMS שמקבלת את השאילתה צריכה לפענח אותה ולהבין שהנתונים הדרושים מנוהלים בשרתים שונים. כדי להבין זאת, על המערכת לנהל קטלוג נתונים גלובלי המתאר היכן מנוהלים כל הנתונים.

ב. השאילתה תפורק למספר תת-שאילתות אותן ניתן לבצע בשרתים אחרים. תת-השאילתות, תשוגרנה לשני השרתים המרוחקים.

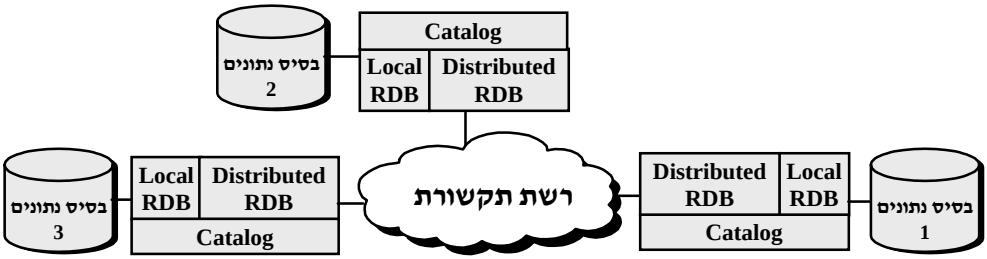
ג. במקביל, תת-השאילתה תבוצע על בסיס הנתונים בשרת המקומי.

ד. המערכת תמתין לקבלת התשובות משני השרתים המרוחקים.

ה. עם קבלת התשובות, המערכת תתחיל בתהליך בניית הטבלה המסכמת. טבלה זו תבנה על ידי מיזוג שלוש הטבלאות, כך שכל פעם שאותו סמסטר מופיע ביותר מטבלה אחת, יבוצע סיכום של מספר הסטודנטים.

כפי שניתן לראות מדוגמה פשוטה זו, מהלך ביצוע השאילתה מורכב יותר מאשר הביצוע בבסיס נתונים מקומי. על המערכת לדעת באיזה שרתים קיימים הנתונים המבוקשים, לשגר ברשת התקשורת את הבקשות המתאימות, להמתין לקבלת התשובות מכל המחשבים ולמזג את התוצאות.

בכדי לנהל בסיס נתונים מבוזר יש צורך בהוספת מרכיב תוכנה חדש, Distributed RDB, שיפעל ליד המרכיב של המערכת המקומית, Local RDB. רכיב זה מזהה את העובדה שהיישום פונה לנתונים המנוהלים בשרת אחר, מאתר את השרת המתאים, שולח ברשת התקשורת את ההודעה המתאימה, קולט ומעבד את התשובה.



תרשים 17.2: מרכיב נוסף לניהול המערכת המבוזרת.

כדי שרכיב בסיס הנתונים המבוזר ידע היכן הנתונים מנוהלים, יש להוסיף לקטלוג המערכת מידע נוסף. מידע זה מתאר את הביזור, כלומר היכן נמצאת איזו טבלה וכד'.

יתרונות וחסרונות בסיס הנתונים המבוזר

נסקור בקצרה את היתרונות ואת החסרונות של בסיס נתונים מבוזר.

יתרונות

- **התאמה למבנה ארגוני מבוזר:** כיום רוב הארגונים מבוזרים בצורה זו או אחרת. ביזור בסיס הנתונים מאפשר התאמה נוחה יותר של מבנה הנתונים עם מבנה הארגון. כל יחידה מנהלת את הנתונים שלה.
- **עצמאות מקומית (Local Autonomy):** הנתונים השייכים ליחידה מקומית מנוהלים ביחידה המקומית מבלי לפגום ביכולת לקבל את התמונה הכוללת. ניתן לקבוע לכל אתר את המדיניות המקומית (מבחינת הרשאות, מדיניות גיבוי, אילוצים עסקיים וכד'). מנהל בסיס הנתונים, ה-DBA המקומי, עוסק בבסיס הנתונים המקומי בשעה שנהל בסיס הנתונים הגלובלי אחראי להגדיר את המידע של הביזור.
- **ביצועים משופרים (Improved Performance):** רוב היישומים הם מקומיים, ולכן הנתונים מנוהלים במחשב המקומי ונמנע הצורך להעביר נתונים ברשת תקשורת מרחבית (WAN). הדבר יכול לשפר מאוד את זמני התגובה של היישומים המקומיים.
- **זמינות מערכת משופרת (Improved System Availability):** בסיס הנתונים מנוהל במספר רב של שרתים, ולכן תקלה בשרת מקומי אינה משביתה את כלל הפעילות, אלא רק את הפעילות המקומית. במערכת מרכזית שבה בסיס הנתונים של כל הארגון מנוהל בשרת מרכזי אחד, תקלה יכולה להשבית את כל הארגון.
- **זמינות נתונים משופרת (Improved Data Availability):** בסיס נתונים מבוזר מאפשר לנהל את אותם הנתונים במספר אתרים שונים, ולכן נוכל לשפר את זמינות הנתונים. אם הנתונים משתבשים באתר מסוים, ניתן לשחזר אותם מאתרים אחרים. למשל אם טבלת סטודנטים מנוהלת בכל המחשבים, תקלה באחת הטבלאות מאפשרת שחזור נוח, מאחר והטבלה מנוהלת גם באתרים אחרים.
- **התרחבות מודולרית (Modular Extendibility):** הוספת אתר היא פעולה נוחה ואינה מחייבת הגדלת כוח המחשוב באתר המרכזי. לכל אתר השרת המקומי שלו.

חסרונות

- **מורכבות (Complexity):** ניהול בסיס נתונים מבוזר היא פעילות מורכבת בהרבה מניהול בסיס נתונים מרכזי. הנושאים המוסיפים למורכבות הם: ניהול קטלוג נתונים גלובלי כדי לדעת היכן הנתונים נמצאים, ניהול נעילות בסביבה מבוזרת, טיפול בבעיות של עדכון בו-זמני בסביבה מבוזרת, טיפול בבעיות של סינכרון בין עותקים שונים של אותם נתונים, אופטימיזציה של שאילתות מבוזרות, טיפול בשלמות ואמינות נתונים בסביבה מבוזרת ועוד.

- **עלות (Cost):** ניהול נתונים מרכזי מאפשר ניצול של יתרונות לגודל. בסיס נתונים מבוזר מחייב שימוש בשרתים מקומיים ושימוש במערכות תוכנה מורכבות יותר. למרות שהעלות של שרתים נמצאת במגמת ירידה מתמדת, עדיין יש לדאוג לכל אתר את אמצעי הגיבוי שלו, את מערך הדיסקים שלו וכל אלה מייקרים את הפתרון הכולל. בדרך כלל יש גם תוספת לעלויות הנובעת מהצורך באנשים רבים יותר. לדוגמה, מנהל בסיס נתונים (DBA) מקומי בכל אתר, לעומת מספר קטן יותר של מנהלים כאשר בסיס הנתונים הוא מרכזי.
- **רגישות לרשת התקשורת (Network Sensitivity):** במקרה של בסיס נתונים מבוזר, תקלה ברשת התקשורת יכולה לגרום לקשיים בהפעלת היישומים הגלובליים וכן בסינכרון בסיסי הנתונים השונים.
- **אבטחת מידע (Security):** אבטחת המידע בסביבה מבוזרת היא מורכבת יותר מאשר במערכת מרכזית. יש צורך באבטחת המידע במספר רב של אתרים ולא רק באתר מרכזי אחד.
- **עיצוב מורכב (Complex Design):** עיצוב בסיס נתונים מבוזר הוא מורכב יותר מאשר עיצוב בסיס נתונים מרכזי. יש לקחת בחשבון שיקולים נוספים כגון באיזה אתר כדאי לשים איזה נתונים, כמה עותקים של טבלה מסוימת יש לנהל וכו'.

מערכות מבוזרות הומוגניות והטרוגניות

מערכת מבוזרת כלשהי לניהול בסיסי נתונים יכולה לפעול על שרתים מסוג אחד, או על שרתים מסוגים שונים ובמערכות הפעלה שונות. לדוגמה, מערכת Oracle הפועלת בשרת עם מערכת הפעלה Unix ובשרת אחר עם מערכת הפעלה Windows NT. מקובל להבחין בין מערכות מבוזרות הומוגניות והטרוגניות לניהול בסיסי נתונים.

מערכת מבוזרת הומוגנית **Homogeneous DRDBMS**	***מערכת מבוזרת הומוגנית*** **מוגדרת כמערכת** DRDBMS **שבה כל המערכות המקומיות הן של אותו יצרן.**

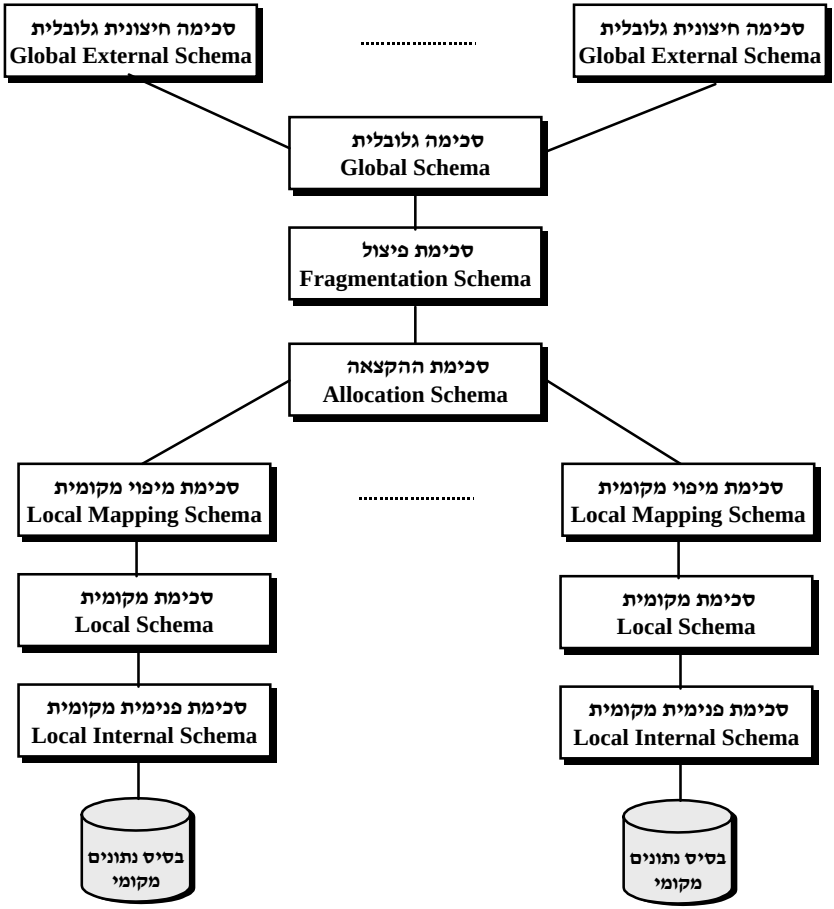
מערכת מבוזרת לניהול בסיסי נתונים נחשבת למערכת הומוגנית כאשר כל מערכות ה- RDBMS המקומיות הן של אותו יצרן. מערכות מסוג זה הן קלות יותר לניהול ותחזוקה, מאחר והן מתוצרת אותו יצרן.

מערכת מבוזרת הטרוגנית **Hetrogeneous DRDBMS**	***מערכת מבוזרת הטרוגנית*** **מוגדרת כמערכת** DRDBMS **שבה המערכות המקומיות הן של יצרנים שונים.**

במערכות מבוזרות הטרוגניות נמצא מערכות RDBMS שונות בשרתים שונים. לדוגמה, בסיס נתונים המבוזר בין מערכת DB2 הפועלת על מחשב יבמ מרכזי, מערכת Informix הפועלת על שרת עם מערכת הפעלה Unix, מערכת SQL Server הפועלת על שרת עם מערכת הפעלה Windows NT וכד'. זהו מצב מורכב ומסובך יותר, כי מערכות של יצרנים שונים חייבות להיות מסוגלות "להבין" זו את זו. בפרק הקודם סקרנו את הסוגיות העיקריות של קישוריות ונגישות בסביבה הטרוגנית מסוג זה.

ארכיטקטורה של מערכת מבוזרת

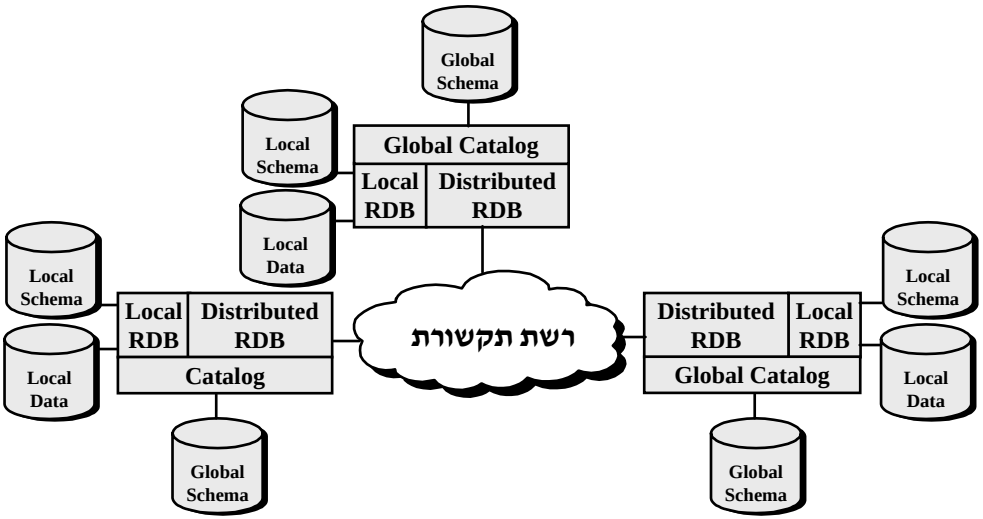
כדי לממש מערכת מבוזרת לניהול בסיסי נתונים יש צורך במספר רמות נוספות בהגדרות המנוהלות על ידי בסיס הנתונים. תרשים 17.3 מציג תיאור של מבנה עקרוני של מרכיבי מערכת מבוזרת.



תרשים 17.3: ארכיטקטורה עקרונית של מערכת מבוזרת.

עתה נסקור את מרכיבי הארכיטקטורה:

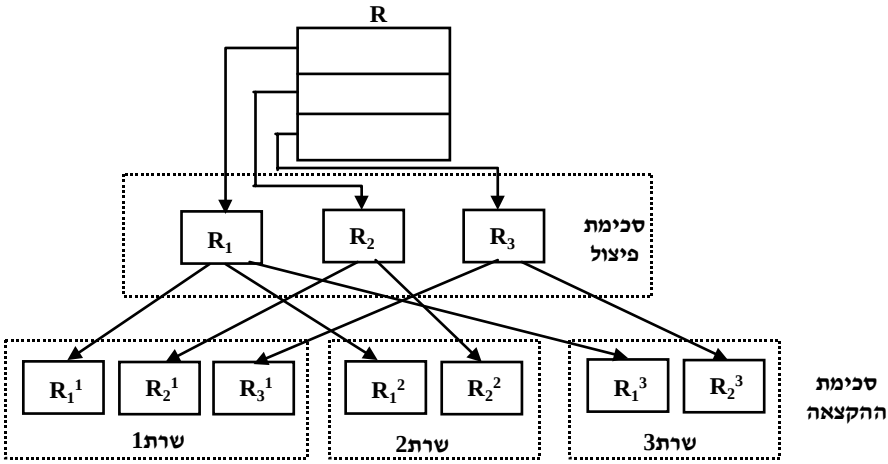
- **סכימה חיצונית גלובלית (Global External Schema)** מגדירה את נקודת המבט של תוכנית יישום גלובלית כלשהי. מנקודת המבט של תוכנית יישום גלובלית, בסיס הנתונים המבוזר נראה בדיוק כמו בסיס נתונים מרכזי.
- **סכימה גלובלית (Global Schema)** מגדירה באופן לוגי את כל הנתונים המנוהלים בבסיס הנתונים באופן רגיל וללא כל התייחסות לנושא הביזור. למעשה, ברמה הזו אין כל הבדל בין בסיס נתונים מרכזי לבסיס נתונים מבוזר.
- **סכימת הפיצול (Fragmentation Schema)** מתארת את הביזור הלוגי של הנתונים. כפי שנראה, ניתן לפצל טבלה אחת למספר **מקטעים** (Fragments) שונים. סכימה זו מתארת כיצד פיצלנו את הטבלה ברמה הלוגית. ניתן לפצל טבלה למקטעים זרים או חופפים, למקטעים אופקיים או אנכים וכד'.
- **סכימת ההקצאה (Allocation Schema)** מגדירה את המיקום של כל מקטע. סכימה זו מבצעת את המיפוי בין המקטע הלוגי לבין השרת הפיסי שבו המקטע מאוחסן. ניתן להכניס כפילות אל תוך בסיס הנתונים המבוזר על ידי כך שנאחסן אותו מקטע במספר שרתים שונים. ל**שכפול הנתונים** (Replication) חשיבות רבה, כי בצורה זו ניתן לשמור את הנתונים בשרת שבו הם נדרשים תכופות (ייתכן יותר משרת אחד). שיטה זו טובה גם לגיבוי במקרה ששרת מסוים ברשת מושבת.
- **סכימת המיפוי המקומי(Local Mapping Schema)** מתארת את המיפוי בין מקטע של טבלה גלובלית לטבלה המנוהלת על ידי מערכת RDBMS המקומית.
- **סכימה פנימית מקומית (Local Internal Schema)** מתארת את צורת האחסון של הטבלאות המקומיות במערכת RDBMS המקומית.



תרשים 17.4: מבנה מערכת מבוזרת.

תרשים 17.4 מדגים כיצד נראית מערכת מבוזרת. בכל שרת מנוהלים הסכימה המקומית והנתונים המקומיים. בנוסף מנוהל בשרת המקומי גם הרכיב של המערכת המבוזרת. רכיב זה מנהל את הסכימה הגלובלית, את סכימת הפיצול, את סכימת ההקצאה ואת סכימת המיפוי המקומית. בצורה זו, כל צומת ברשת מכיר את ביזור הנתונים. כמובן שניתן לנהל גם את הסכימה הגלובלית באתר אחד בלבד, אולם אז כל גישה מחייבת שימוש ברשת התקשורת ואתר זה הופך להיות לנקודת הכשל המרכזית (Single Point of Failure) של כל המערכת.

להמחשה נראה כיצד ניתן לפצל טבלה R למספר מקטעים ולקבוע היכן יאוחסן כל מקטע.



תרשים 17.5: דוגמה לפיצול טבלה למקטעים באתרים שונים.

התרשים מציג טבלה R שפוצלה לשלושה מקטעים לא חופפים: R_1, R_2 ו-R_3. פיצול זה מוגדר באמצעות סכימת הפיצול. סכימת ההקצאה קובעת באיזה שרתים נאחסן איזה מקטעים. לדוגמה:

מקטע R_1 מאוחסן בשרת 1 (יסומן R_1^1), בשרת 2 (יסומן R_1^2) ובשרת 3 (יסומן R_1^3).

מקטע R_2 מאוחסן בשרת 1 (יסומן R_2^1) ובשרת 2 (יסומן R_2^2).

מקטע R_3 מאוחסן בשרת 1 (יסומן R_3^1) ובשרת 3 (יסומן R_3^3).

יש להבחין כאן בין שני מושגים שונים: **מיקום** (Location) **ושכפול** (Replication).

המיקום קובע באיזה שרת המקטע מאוחסן. לדוגמה, מקטע R_2^1 מאוחסן באתר 1. **השכפול** קובע האם מותר לאחסן את אותו מקטע במספר שרתים. בהחלט ייתכן מצב, שמערכת מבוזרת אינה תומכת בשכפול. תרשים 17.5 מציג דוגמה הכוללת שכפול. לדוגמה, מקטע R_1 משוכפל ומאוחסן בשלושה שרתים שונים.

רמת השקיפות בבסיס נתונים מבוזר

שלוש הסכימות – הסכימה הגלובלית, סכימת הפיצול וסכימת ההקצאה – מאפשרות להגדיר את רמת השקיפות של המערכת, כלומר עד כמה המשתמש חשוף לקיומו של בסיס הנתונים המבוזר. כך מציינים את רמות השקיפות המקובלות:

א. **שקיפות לפיצול (Fragmentation transparency)**: זוהי רמת השקיפות הגבוהה ביותר. המשתמש אינו מודע לקיום המקטעים ומבצע את פעולותיו על הטבלה הגלובלית. בהתייחס לדוגמה הקודמת, המשתמש פועל על טבלה R. המערכת תשתמש במידע המנוהל בקטלוג על הפיצול ועל ההקצאה ותבצע את העבודה על המקטע ובשרת האופטימלי. במידת הצורך, המערכת תעביר מקטע משרת לשרת כדי לבצע פעולות Join בשרת האופטימלי.

ב. **שקיפות למיקום (Location transparency)**: זוהי רמת שקיפות נמוכה יותר. המשתמש מודע לקיום המקטעים ומבצע את פעולותיו עליהם, ולא על הטבלה הגלובלית. עם זאת, המשתמש אינו מודע לשרת הפיסי שבו מאוחסן המקטע. בהתייחס לדוגמה הקודמת, המשתמש פועל על טבלאות R_1, R_2 ו-R_3 מבלי לדעת באיזה שרתים המקטעים מאוחסנים. המערכת משתמשת במידע שבקטלוג לאיתור השרת שבו מאוחסנים המקטעים.

ג. **שקיפות לשכפול (Replication transparency)**: רמת שקיפות זו דומה לשקיפות המיקום, אולם אינה זהה לה. המשתמש מודע לקיום המקטעים, אולם אינו מודע לשכפול המקטע במספר שרתים שונים. ייתכן שמערכת תספק שקיפות מיקום, שבה המשתמש אינו יודע באיזה שרתים המקטע מאוחסן, אך היא לא תספק שקיפות לשכפול. במקרה זה המשתמש מודע לכך שמקטע מסוים מאוחסן בשלושה שרתים שונים ולכן הוא חייב לבצע את העדכון בכל שלושת השרתים מבלי לדעת מי הם שלושת השרתים. בהתייחס לדוגמה הקודמת, המשתמש מודע לך שמקטע R_1 שוכפל שלוש פעמים, אולם אינו יודע באיזה שרתים הוא מאוחסן.

פיצול טבלאות (Table Fragmentation)

תהליך הפיצול עוסק בפיצול של טבלה גלובלית אחת למספר מקטעים לא חופפים שמאוחסנים בשרת כלשהו במערכת המבוזרת. תהליך הפיצול אינו קובע היכן יאוחסנו המקטעים, אלא רק איזה מקטעים קיימים. נגדיר תחילה את המושג מקטע של טבלה.

מקטע של טבלה **Table Fragment**	***מקטע של טבלה* מוגדר כחלק של טבלה (תת-טבלה) כלשהי הנבנה בתהליך המבוסס על שלושה כללים: שלמות הנתונים, יכולת שחזור הטבלה מתוך תת-הטבלאות ואי חפיפה בין תת-הטבלאות.**

שאלה ראשונה שעולה מייד היא, בכלל מדוע לפצל טבלה למספר תת-טבלאות (או מקטעים, כפי שנקרא להם). מדוע לא להתייחס אל הטבלה כאל יחידת הביזור, אשר תנוהל במספר שרתים שונים. התשובה לכך היא שהטבלה איננה יחידת ביזור טבעית. רוב היישומים אינם פועלים על כל הטבלה אלא רק על חלקים ממנה. לעקרון זה מקובל לקרוא **עקרון הגישה מקומית** או **Locality of Access**. עקרון הגישה המקומית נובע מהמסקנה שרוב תוכניות היישום הפועלות בשרת מסוים נוטות לטפל באוסף חלקי בלבד של שורות מתוך הטבלה הגלובלית.

למשל, אם נבזר את טבלת **קורסים** לכל שלושת הקמפוסים, סביר להניח שברוב הגדול של היישומים המופעלים בקמפוס מסוים, הפניות הן לחלק מסוים בלבד של הטבלה, אל השורות המתייחסות **לקורסים** הניתנים באותו קמפוס. רק במיעוט המקרים נראה יישומים הפונים **לקורסים** הניתנים בקמפוסים אחרים. אם נקבל את עקרון הגישה המקומי, ביזור טבלה לכל השרתים מהווה בזבוז משאבים, הן מבחינת אחסון והן מבחינת הניהול שלהם. ממילא רוב התוכניות ניגשות רק לחלק מסוים של הטבלה.

בגלל עקרון הגישה המקומית של תוכניות היישום, פיצול טבלה למספר מקטעים ישפר את הביצועים ויגדיל את התפוקה (Througput) של כלל המערכת. ניהול מקטעים בשרתים רלונטיים בלבד, יאפשר לתנועות המקומיות לפעול מבלי להפריע אחת לשנייה ויביא לשיפור בביצועים. מאחר וכל תנועה מקומית פועלת על מקטע, שסביר להניח שהוא קטן יותר מהטבלה הגלובלית, הביצועים של התנועות המקומיות יהיו טובים יותר.

יחד עם היתרונות של פיצול טבלה למספר מקטעים, חשוב להדגיש גם את החסרונות של הפיצול. אם עקרון הגישה המקומית לא מתקיים, כלומר יש מספר רב של תוכניות שמטפלות ברוב חלקי הטבלה, ייתכן שהפיצול יגרום אפילו לירידה בביצועים והעמסה מיותרת של רשת התקשורת. הפיצול עלול גם לגרום לקושי בשמירה על האמינות והסמנטיקה של הטבלה, מאחר וחלקים שונים שלה נמצאים בשרתים שונים. בדיקות האמינות עלולות להיות מורכבות יותר אם עמודות או שורות שונות נמצאות בשרתים שונים. למרות חסרונות אלה, קיימת הסכמה בקרב החוקרים ויצרני מערכות RDBMS המנסים לקדם את נושא ביזור הנתונים, שהמקטע כיחידת הביזור מהווה בסיס טוב יותר לביזור.

כללים לבניית מקטעים

כפי שראינו מתוך ההגדרה של מקטע, המקטע חייב לקיים שלושה כללים. נרחיב במקצת את הדיון בכללים אלה. שלושת הכללים לבניית מקטעים הם:

- **שלמות (Completeness):** אם טבלה R מפוצלת למקטעים R_1, R_2, ...R_n, כל נתון המופיע בטבלה R חייב להופיע בלפחות מקטע אחד משמעות הכלל הזה היא שלא ייתכן שתוך כדי תהליך הפיצול נאבד נתונים מתוך הטבלה הגלובלית.
- **שחזור (Reconstruction):** ניתן לשחזר את טבלה R מתוך המקטעים R_1, R_2, ...R_n תוך שימוש בפקודות SQL רגילות. כלל זה מבטיח שכל התלויות הפונקציונליות

נשמרות גם בזמן הפיצול למקטעים. כלומר, הפיצול חייב להבטיח שלמות ויכולת שחזור.

- **חוסר חפיפה (Disjointness):** המקטעים צריכים להיות נפרדים ולא לחפוף זה את זה. אם עמודה C_i מופיעה במקטע R_i, אסור שעמודה זו תופיע במקטע אחר כלשהו. כלומר, נתונים המופיעים במקטע אחד לא יופיעו גם במקטע אחר. יש רק יוצא מן הכלל אחד והוא מתייחס לגבי עמודות השייכות למפתח העיקרי. לעמודות אלו מותר להופיע במספר מקטעים.

לדוגמה, נניח שנתונה הטבלה הבאה:

מרצים (<u>מספר מרצה</u>, שם מרצה, התמחות, קוד מחלקה אקדמית)

ניתן לפצל אותה במספר צורות שונות.

פיצול אפשרי אחד יהיה:

מרצים_1 (<u>מספר מרצה</u>)

מרצים_2 (<u>שם מרצה</u>, התמחות, קוד מחלקה אקדמית)

פיצול זה שומר על השלמות אולם אינו מאפשר שחזור, מכיון שמפתח הטבלה אינו מופיע בשני המקטעים שלה. פיצול אחר, המקיים את כל שלושת הכללים יהיה:

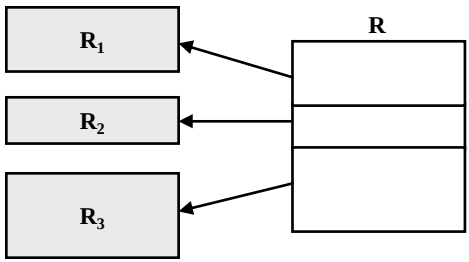
מרצים_1 (<u>מספר מרצה</u>, שם מרצה)

מרצים_2 (<u>מספר מרצה</u>, התמחות, קוד מחלקה אקדמית)

לאחר שסקרנו את הכללים שכל פיצול חייב לקיים, נסקור את השיטות השונות לפיצול טבלאות למקטעים.

פיצול אופקי (Horizontal Fragmentation)

הפיצול האופקי בונה את המקטעים על ידי ביצוע פקודות SELECT על **שורות** בלבד. כך, חלק מהשורות מועברות למקטע אחד, וחלק אחר של השורות, שאינו חופף, מועבר למקטע אחר.



תרשים 17.6: פיצול אופקי.

לצורך המחשת הנושא נתייחס למכללה, ונניח שהיא פועלת בשלושה קמפוסים שבכל אחד מהם יש מחלקה אקדמית אחת בלבד. נתבונן רק בטבלת מרצים, המכילה את הנתונים האלה: מספר מרצה, שם מרצה, התמחות וקוד המחלקה האקדמית שאליה הוא שייך. הטבלה הגלובלית של המרצים מוצגת בתרשים 17.7.

Faculty ***מרצים***

TEACHER_ID	NAME	SPECIALIZATION	DEPARTMENT
מס .מרצה	שם מרצה	התמחות	קוד מחלקה
210	Dr. Israel	Data Bases	CS
100	Prof. Ron	Compilers	CS
420	Prof. Eyal	Accounting	BS
58	Dr. Dan	Marketing	BS
95	Dr. Eli	Communication	CS
215	Dr. Jeff	Accounting	BS
340	Prof. Levy	Optimization	MT
99	Prof. Sharon	Queing Theory	MT
348	Prof. Avi	Information Systems	BS

תרשים 17.7: טבלת מרצים גלובלית.

פיצול אופקי של טבלה זו יכול להיעשות באמצעות שלוש פקודות SQL אלו:

```
1. DEFINE FRAGMENT FACULTY_BS AS
2. SELECT  *
3. FROM    FACULTY
4. WHERE DEPARTMENT = 'BS'

1. DEFINE FRAGMENT FACULTY_MT AS
2. SELECT  *
3. FROM    FACULTY
4. WHERE DEPARTMENT = 'MT'

1. DEFINE FRAGMENT FACULTY_CS AS
2. SELECT  *
3. FROM    FACULTY
4. WHERE DEPARTMENT = 'CS'
```

נדגיש שהמשפט DEFINE FRAGMENT איננו חלק מהתקן של שפת SQL ומופיע כאן רק כדי להדגים את הרעיון. רוב הדוגמאות המופיעות בהמשך נלקחו מתוך מערכת CA-Ingres, שהיתה אחת המערכות הראשונות והמובילות בכל הקשור לביזור בסיס הנתונים. יצרנים שונים ממשים את הרעיון בצורות שונות.

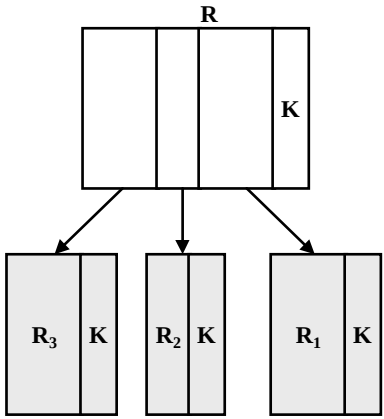
כתוצאה מפיצול אופקי זה נקבל שלושה מקטעים לא חופפים, מקטע אחד עבור כל מחלקה.

נשים לב, שלא איבדנו נתונים במהלך הפיצול ונוכל לחזור ולבנות את הטבלה הגלובלית על ידי איחוד שלוש הטבלאות:

```
1. SELECT *
2. FROM   FACULTY_BS
3. UNION
4. SELECT *
5. FROM   FACULTY_MT
6. UNION
7. SELECT *
8. FROM   FACULTY_CS
```

פיצול אנכי (Vertical Fragmentation)

זהו תהליך בניית מקטעים על ידי ביצוע פקודות SELECT הבוחרות עמודות מסוימות. חלק מהעמודות מועבר למקטע אחד וחלק עובר למקטע אחר. כדי לאפשר שחזור הטבלה הגלובלית מתוך המקטעים על ידי פעולות Join, המפתח העיקרי צריך להיות בכל מקטע.



תרשים 17.8: פיצול אנכי.

בדוגמה של טבלת המרצים נבצע פיצול מאונך, ואז נקבל מקטע עם שם המרצה וקוד המחלקה האקדמית ומקטע נפרד של התמחויות. נבצע את שתי פקודות SELECT האלו:

```
1. DEFINE FRAGMENT FACULTY_DATA AS
2. SELECT  TEACHER_ID, NAME, DEPARTMENT_ID
3. FROM    FACULTY

1. DEFINE FRAGMENT FACULTY_SPECIALIZATION AS
2. SELECT  TEACHER_ID, SPECIALIZATION
3. FROM    FACULTY
```

נוכל לשחזר חזרה את הטבלה הגלובלית על ידי פעולת Join בין שני המקטעים שהתקבלו.

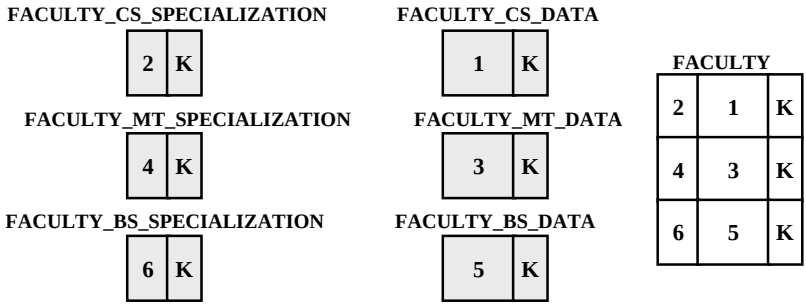
פיצול מעורב (Mixed Fragmentation)

תהליך פיצול זה משלב פיצול אופקי ופיצול אנכי. לצורך ההמחשה, נניח שיש לפצל את טבלת מרצים למקטעים לפי מחלקה, ולפצל כל מקטע לנתוני מרצים ולנתוני התמחויות. לקבלת שני המקטעים של המחלקה למדעי המחשב נבצע שתי פקודות SELECT אלו:

```
1. DEFINE FRAGMENT FACULTY_CS_DATA AS
2. SELECT  TEACHER_ID, NAME, DEPARTMENT_ID
3. FROM    FACULTY
4. WHERE DEPARTMENT = 'CS'
```

```
1. DEFINE FRAGMENT FACULTY_CS_SPECIALIZATION AS
2. SELECT  TEACHER_ID, SPECIALIZATION
3. FROM    FACULTY
4. WHERE DEPARTMENT = 'CS'
```

אם נבצע פעולות אלו עבור כל המחלקות נקבל ששה מקטעים – שניים לכל מחלקה.



תרשים 17.9: פיצול מעורב של טבלת מרצים.

הקצאת מקטעים בבסיס נתונים מבוזר

בגמר תהליך הפיצול למקטעים יש להחליט איזה מקטע יאוחסן באיזה שרת. בשלב הזה ניתן להכניס כפילות לבסיס הנתונים על ידי אחסון אותו מקטע ביותר משרת אחד. תהליך קביעת הקצאת המקטעים לשרתים הינו בעל חשיבות רבה בבניית בסיס נתונים מבוזר. ההחלטה על הקצאת המקטעים קובעת למעשה את היקף העיבוד המקומי ואמנם, המגמה היא להקצות את המקטעים לשרת שבו הם נדרשים. בדוגמה שלנו, ברור שנקצה לשרת של כל מחלקה את שני המקטעים של המחלקה. נניח גם שנקצה לשרת המזכירות האקדמית של המכללה את כל ששת המקטעים, כדי לאפשר למזכירות עיבוד מהיר של הנתונים הדרושים. לאחר ניתוח נוכל לקבל את סכימת ההקצאה כמוצג בתרשים 17.10.

שרת	מקטע
מדעי המחשב	FACULTY_CS_DATA FACULTY_CS_SPECIALIZATION
מתמטיקה	FACULTY_MT_DATA FACULTY_MT_SPECIALIZATION
מנהל עסקים	FACULTY_BS_DATA FACULTY_BS_SPECIALIZATION
מזכירות אקדמית	FACULTY_CS_DATA FACULTY_CS_SPECIALIZATION FACULTY_MT_DATA FACULTY_MTSPECIALIZATION FACULTY_BS_DATA FACULTY_BS_SPECIALIZATION

תרשים 17.10: סכימת הקצאה.

בסכימת הקצאה זו הוכנסה כפילות, כי כל מקטע מאוחסן פעמיים: פעם במחשב של המחלקה ופעם שנייה במחשב של המזכירות האקדמית. בשל כפילות זו, המזכירות האקדמית יכולה לקבל את המידע הדרוש לה במהירות ויעילות, כי הוא מושג במהלך עיבוד של טבלאות מקומיות. עם זאת, כל עדכון של טבלה במחשב המחלקה מחייב גם עדכון של הטבלה התואמת במחשב המזכירות האקדמית. ייתכן שכדי להעמיס פחות את רשת התקשורת ולייעל את העדכונים המקומיים, נחליט לא לבצע עדכון הטבלאות במזכירות בזמן אמיתי. טבלאות אלו יעודכנו על ידי עיבוד אצווה שיופעל בלילה ויעדכן את הטבלאות של המזכירות עם כל השינויים שהיו במהלך היום.

כפי שניתן לראות מדיון קצר זה, עיצוב בסיס נתונים מבוזר הינו מורכב למדי ומחייב לקחת בחשבון גם את מבנה רשת התקשורת והעומס הצפוי בה, העיבודים המקומיים לעומת העיבודים הגלובליים, עוצמת השרתים באתרים השונים, רמת הגיבוי הנדרשת ועוד. שיקולים אלה צריכים להנחות את המעצב כאשר הוא מחליט על רמת הכפילות בבסיס הנתונים המבוזר.

תהליך הבנייה של בסיס נתונים מבוזר

בגמר תהליך עיצוב בסיס הנתונים שבו הוחלט איזה מקטעים ינוהלו והיכן, מתחיל התהליך של בניית בסיס הנתונים עצמו. בכל שרת ברשת יש לבנות את בסיס הנתונים המקומי באמצעות הכלים הרגילים של שפת SQL, עם פקודות CREATE TABLE ו- CREATE INDEX. כל צומת חייב "להירשם", כדי שהטבלאות המקומיות יהפכו לחלק מבסיס הנתונים המבוזר. רישום זה יכול להתבצע על ידי פקודה מיוחדת כגון REGISTER.

לדוגמה, כדי שהטבלאות של המחלקה למדעי המחשב יירשמו כחלק מבסיס הנתונים המבוזר, נבצע את הפקודות האלו:

```
1. REGISTER TABLE FACULTY_CS_DATA
2. AS LINK FROM FACULTY_DATA
3. WITH NODE = CS

1. REGISTER INDEX FACULTY_CS_DATA_IX
2. ON FACULTY_CS_DATA
3. AS LINK FROM FACULTY_DATA_IX
```

נניח עכשיו, שבמחשב המזכירות האקדמית רוצים לבנות טבלה המכילה את כל נתוני המרצים מכל המחלקות. נגדיר את הטבלה הזאת באמצעות משפט CREATE מיוחד.

```
1. CREATE TABLE FACULTY_DATA AS
2. SELECT *
3. FROM FACULTY_CS_DATA
4. UNION
5. SELECT *
6. FROM FACULTY_MT_DATA
7. UNION
8. SELECT *
9. FROM FACULTY_BS_DATA
10. WITH NODE = 'Secretary'
```

פקודות אלו יטענו את הטבלה לשרת המזכירות האקדמית, כך שהיא תכיל את כל נתוני המרצים מכל המחלקות. מרגע זה ואילך ניתן לבנות שאילתות עבור טבלה זו. הטבלה FACULTY_DATA לא מתעדכנת באופן אוטומטי עם כל עדכון המבוצע בשרת של מחלקה כלשהי. מזמן לזמן, למשל כל לילה, ניתן לבנות מחדש את הטבלה FACULTY_DATA, כך שהמזכירות תשתמש בעותק המעודכן ללילה הקודם. לפני כל בנייה יש לבטל תחילה את העותק הנוכחי של הטבלה על ידי הפקודה DROP TABLE.

אילו רצינו שהעותק במזכירות האקדמית יעודכן אוטומטית, אזי היה צורך להגדיר בסכימת ההקצאה שכל המקטעים הרלוונטיים יאוחסנו בשרת המחלקה וגם בשרת המזכירות. מערכת DRDBMS היתה צריכה לוודא שכל עדכון יבוצע על שני המקטעים. במקרה זה נשיג עדכניות על חשבון עומס רשת התקשורת והאטה מסוימת בביצוע העדכון בשרת המקומי.

שימוש בבסיס נתונים מבוזר

השימוש בבסיס הנתונים המבוזר מותנה ברמת השקיפות שמערכת DRDBMS מספקת. ננסה להשתמש במספר שאילתות כדי להדגים את רמות השקיפות השונות שמערכת DRDBMS יכולה לספק.

שקיפות למקטעים
(Fragmentation Transparency)

השקיפות למקטעים היא רמת השקיפות הגבוהה ביותר, שבה המשתמש אינו מודע כלל לעובדה שבסיס הנתונים מבוזר. הוא כותב את השאילתות תוך פנייה אל הטבלאות הגלובליות בלבד. בדוגמה שלנו, משתמש המחובר לשרת של המזכירות האקדמית יפנה אל טבלת FACULTY מבלי להתייחס לעובדה שטבלה זו מורכבת למעשה משישה מקטעים שונים המאוחסנים בשלושה שרתים שונים, אחד בכל קמפוס. למשל, כדי לקבל את רשימת כל המרצים בעלי התמחות חשבונאות, הוא יכתוב את השאילתה הבאה:

```
1. SELECT *
2. FROM FACULTY
3. WHERE SPECIALIZATION = 'Accounting'
```

במקרה זה, רכיב האופטימיזציה של מערכת DRDBMS יגדיר את האסטרטגיה של ביצוע השאילתה. אסטרטגיה אפשרית אחת תהיה:

- פירוק השאילתה לשלוש תת-שאילתות שתשוגרנה לכל שרת בקמפוס. נראה את הדוגמה של השאילתה המיועדת לשרת של המחלקה למדעי המחשב. כפי שניתן לראות, השאילתה תבצע Join בין שני המקטעים כדי להציג את כל נתוני המרצים.

```
1. SELECT *
2. FROM    FACULTY_CS_DATA, FACULTY_CS_SPECIALIZATION
3. WHERE SPECIALIZATION = 'Accounting' AND
4.              FACULTY_CS_DATA.TEACHER_ID =
5.              FACULTY_CS_SPECIALIZATION.TEACHER_ID
```

- שיגור תת-השאילתה המתאימה לכל שרת. השרת המקומי יבצע את תת-השאילתה המתאימה.
- ביצוע תת-השאילתה בשרת המקומי, במקביל לביצוע תת-השאילתות בשרתים האחרים.
- עם קבלת כל התשובות, ביצוע פעולת Union על כל שלוש טבלאות התוצאה שהתקבלו.

בדוגמה זו, מערכת DRDBMS צריכה לבצע את כל ההחלטות הקשורות לבניית הטבלאות מתוך המקטעים, להחליט איזה תת-שאילתות לשגר לאיזה שרת, לאתר היכן המקטעים מאוחסנים וכד'. מערכת DRDBMS המספקת רמת שקיפות זו מאפשרת ביצוע שינויים בסכימת הפיצול ובסכימת ההקצאה, וכל זאת בלי להשפיע על תוכניות היישום.

שקיפות למיקום (Location Transparency)

השקיפות למיקום היא רמת שקיפות נמוכה יותר. אם מערכת DRDBMS מספקת שקיפות למיקום, משמעות הדבר היא שהמשתמש חייב להיות מודע לקיום המקטעים, אולם הוא אינו מודע באיזה שרת הם מנוהלים. בדוגמה שלנו, המשתמש יבצע Join בין שני המקטעים של הטבלה ויבצע את האיחוד בין התוצאות.

```
1.  SELECT *
2.  FROM    FACULTY_CS_DATA, FACULTY_CS_SPECIALIZATION
3.  WHERE SPECIALIZATION = 'Accounting' AND
4.              FACULTY_CS_DATA.TEACHER_ID =
5.              FACULTY_CS_SPECIALIZATION.TEACHER_ID
6.  UNION
7.  SELECT *
8.  FROM    FACULTY_MT_DATA, FACULTY_MT_SPECIALIZATION
10. WHERE SPECIALIZATION = 'Accounting' AND
11.             FACULTY_MT_DATA.TEACHER_ID =
12.             FACULTY_MT_SPECIALIZATION.TEACHER_ID
13. UNION
14. SELECT *
15. FROM    FACULTY_BS_DATA, FACULTY_BS_SPECIALIZATION
16. WHERE SPECIALIZATION = 'Accounting' AND
17.             FACULTY_BS_DATA.TEACHER_ID =
18.             FACULTY_BS_SPECIALIZATION.TEACHER_ID
```

כפי שניתן לראות, הכתיבה כאן יותר מייגעת ומחייבת הכרת כל המקטעים של הטבלה. אגב, ייתכן שבמקרה זה המשתמש היה מחליט לפנות רק אל המקטעים של המחלקה למנהל עסקים, משום שרק בה יש מרצים עם ההתמחות המבוקשת ולא היה פונה לשאר המקטעים.

עם זאת, המשתמש אינו מודע למיקום שבו מנוהלים המקטעים וגם אינו מודע לשכפול, כלומר אם מקטע מסוים מנוהל ביותר משרת אחד. רמת שקיפות זו אמנם נמוכה יותר מהרמה הקודמת, אולם עדיין מאפשרת העברת מקטע משרת לשרת, שכפול מקטע מספר פעמים, וכל זאת בלי לשנות את תוכניות היישום.

ללא שקיפות

בסיס נתונים מבוזר שאינו מספק רמת שקיפות כלשהי, מחייב את המשתמש לפנות אל המקטעים ולציין את השרת שבו הם מנוהלים. אם נתייחס לדוגמה הקודמת, המשתמש יכתוב את אותן הפקודות, אך יוסיף את שם השרת שבו מנוהלת הטבלה (ראה בעמוד הבא).

```
1.  SELECT *
2.  FROM    FACULTY_CS_DATA AT SITE CS,
3.          FACULTY_CS_SPECIALIZATION AT SITE CS
4.  WHERE SPECIALIZATION = 'Accounting' AND
5.          FACULTY_CS_DATA.TEACHER_ID =
6.          FACULTY_CS_SPECIALIZATION.TEACHER_ID
7.  UNION
8.  SELECT *
9.  FROM    FACULTY_MT_DATA AT SITE MT,
10.         FACULTY_MT_SPECIALIZATION AT SITE MT
11. WHERE SPECIALIZATION = 'Accounting' AND
12.         FACULTY_MT_DATA.TEACHER_ID =
13.         FACULTY_MT_SPECIALIZATION.TEACHER_ID
14. UNION
15. SELECT *
16. FROM    FACULTY_BS_DATA AT SITE BS,
17.         FACULTY_BS_SPECIALIZATION AT SITE BS
18. WHERE SPECIALIZATION = 'Accounting' AND
19.         FACULTY_BS_DATA.TEACHER_ID =
20.         FACULTY_BS_SPECIALIZATION.TEACHER_ID
```

במצב זה, שינוי השרת שבו מנוהל המקטע מחייב שינוי מתאים בתוכנית היישום. אחד הפתרונות שחלק ממערכות DRDBMS מספקות הוא שימוש בשמות נרדפים (Aliases) ולא לאפשר למשתמש לפנות לשם האמיתי של הטבלה. נניח שאנו רוצים לקבוע שם חד ערכי למקטעים של טבלת המרצים, FACULTY. נקבע את הכלל הבא:

$$S_i.FACULTY.F_j$$

כאשר S הוא שם השרת שבו המקטע מנוהל, F הוא מספר המקטע של הטבלה. לדוגמה השם S2.FACULTY.F1 יהיה השם הפיסי של המקטע הראשון של טבלת FACULTY המנוהל בשרת S2 ו- S2.FACULTY.F2 יהיה השם של המקטע השני המנוהל באותו שרת. ברור ששיטה זו, המבטיחה חד-ערכיות בשמות המקטעים, אינה נוחה למשתמשים. במקום לאפשר למשתמשים לפנות באופן ישיר לשמות פיסיים אלה, נגדיר שם נרדף לכל מקטע. לדוגמה למקטע S2.FACULTY.F1 נקרא בשם FACULTY_CS_DATA. מערכת DRDBMS תבצע את המיפוי בין השם הלוגי לשם הפיסי. אם נרצה להעביר את המקטע לשרת אחר, נצטרך לשנות פעם אחת את ההגדרה של השם הנרדף ולא נצטרך לשנות את תוכניות היישום.

עדכון בסיס נתונים מבוזר עם שכפול

אם בסיס הנתונים המבוזר אינו תומך בשכפול מקטעים, בעיית העדכון פשוטה יחסית, מאחר וכל מקטע מנוהל פעם אחת בשרת אחד בלבד. על מערכת DRDBMS להשתמש בקטלוג כדי למצוא היכן המקטע מנוהל ולבצע את העדכון המתאים. הבעיה הופכת לקשה יותר ברגע שאותו מקטע משוכפל מספר פעמים, כלומר מנוהל במספר שרתים שונים או אולי אפילו מספר פעמים באותו שרת (משיקולי שרידות או בצועים). יש מספר אפשרויות לטיפול בעדכון עותקים של מקטעים משוכפלים (Replicated Tables Update Policy).

להלן כמה מהאפשרויות:

- עדכון מיידי של כל העותקים של המקטע בכל השרתים. הבעיה בשיטה זו היא, שאם שרת מסוים לא פעיל מסיבה כלשהי, לא ניתן לבצע את העדכון. ככל שרמת השכפול הולכת וגדלה, ההסתברות לבצע את העדכון הולכת וקטנה בגלל האפשרות ששרת כלשהו לא יהיה זמין. יש כאן סתירה בין מטרת השכפול האמורה לספק רמה גבוהה של זמינות הנתונים בשרת המקומי לעומת האפשרות של שמירה על רמת עדכניות גבוהה של כל העותקים.
- עדכון מיידי של כל העותקים הזמינים ועדכון מאוחר יותר של יתר העותקים. למרות ששיטה זו מציגה שיפור מסוים לעומת האפשרות הקודמת, דחיית העדכונים, של העותקים המנוהלים בשרתים לא זמינים לשלב מאוחר יותר, מורכבת. הדבר מחייב פיתוח פרוטוקול מתאים, כך שכל אחד יוכל לוודא אם ניתן להפעיל את העדכונים על העותקים השונים. בפרק הקודם הצגנו את האפשרות לבצע שיכפול עותקים תוך שימוש במנגנון Replication Server.
- הגדרת עותק אחד כעותק ראשי ועותקים אחרים כעותקים משניים. העדכון יבוצע מול העותק הראשי ועם סיומו התנועה תיחשב כמבוצעת. באחריות השרת המנהל את העותק הראשי לבצע את העדכונים של העותקים המשניים. הבעיה בשיטה זו היא, שאם העותק הראשי לא זמין לא ניתן לבצע את העדכון.
- הגדרת עותק אחד כעותק ראשי וכל יתר העותקים משמשים לקריאה בלבד ואין צורך לעדכן אותם מייד. על המערכת לאפשר רענון העותקים המשניים מזמן לזמן באופן אוטומטי. שיטה זו אמנם מפשטת את העדכון, מאחר ויש לעדכן רק עותק אחד, אולם יש לזכור שכל יתר העותקים מעודכנים רק פעם אחת לתקופה.

קטלוג המערכת בסביבה מבוזרת

אחד הנושאים המרכזיים בסביבה מבוזרת הינו **מיקום קטלוג המערכת**. למשל, כיצד ידע שרת באתר אחד באיזה שרתים אחרים מנוהלות הטבלאות והמקטעים האחרים?

על הקטלוג במערכת המבוזרת להכיל מידע היכן נמצא כל מקטע, האם יש לאותו מקטע מספר עותקים באתרים שונים ומה מדיניות העדכון שלהם. עכשיו נשאלת השאלה היכן הקטלוג עצמו ינוהל. לנושא זה מספר פתרונות אפשריים. נמנה כמה מהם:

- **קטלוג המערכת המלא נמצא בכל השרתים:** במצב זה, כל שרת ברשת מנהל עותק מלא של קטלוג המערכת. יתרונה של שיטה זו, שכל שרת מכיל את כל המידע הדרוש לו כדי לקבוע היכן נמצאות הטבלאות. חיסרון השיטה הוא בכך שיש צורך לוודא שכל שינוי בקטלוג מקומי יועבר מיידית לכל יתר השרתים.
- **כל מערכת מנהלת את הקטלוג המקומי בלבד:** ברגע שיש פנייה לטבלה שאינה מוגדרת בקטלוג המקומי, המערכת פונה בשאילתה לכל אחד משאר השרתים ברשת כדי לאתר את הטבלה המבוקשת. שיטה זו תהיה בעלת ביצועים טובים פחות מקודמתה, משום שעל השרת המקומי לבצע מספר רב של בדיקות לפני ביצוע

הפקודה. שיטה זו גם מחייבת לפתח פרוטוקול שיאפשר למערכות "לשוחח" ביניהן. יתרונה של שיטה זו, שכל קטלוג מנוהל רק בשרת המקומי ושינוי בו אינו חייב להיות מועבר לשרתים אחרים ברשת.

- **ניהול הקטלוג בשרת מרכזי אחד בלבד:** כל השרתים האחרים פונים לשרת זה לפני שהם מבצעים פעולה כלשהי בבסיס הנתונים, הן המקומי והן המרוחק. יתרון שיטה זו, הוא בניהול מרכזי של הקטלוג ללא צורך בהפצתו לשרתים אחרים. החיסרון הבולט, הוא בעומס נוסף על רשת התקשורת והאטה בביצועים. חיסרון נוסף הוא למשל, כאשר מסיבה כלשהי השרת המרכזי לא פועל, כל המערכת מושבתת, כי בהיעדר הקטלוג לא ניתן לבצע אפילו עבודות מקומיות.

אמינות בסיס נתונים מבוזר (Distributed Data Integrity)

עד עתה עסקנו בעיקר באיחזור הנתונים מתוך בסיס נתונים מבוזר. הבעיה הופכת למורכבת יותר כאשר רוצים לאפשר גם עדכון נתונים. תנועה מבוזרת מבצעת עדכונים במספר טבלאות מבוזרות ואם, מסיבה כלשהי, העדכון בשרת אחד או יותר לא מצליח, על מערכת DRDBMS לבטל את העדכון בכל השרתים המעודכנים, כלומר לבצע גלילה לאחור (Rollback).

לדוגמה, כאשר מרצה עובר ממחלקה למחלקה, יש לבטל תחילה את השורה של המרצה בטבלה של המחלקה שבה הוא הועסק, ולהוסיף שורה חדשה לטבלה של המחלקה החדשה. נתאר לעצמנו מצב, שבו הביטול כבר בוצע ומסיבה כלשהי השרת של המחלקה החדשה אינו זמין ולכן לא ניתן להוסיף את השורה החדשה. במצב זה על המערכת לבטל את הביטול שכבר בוצע בשרת הראשון.

כדי להתמודד עם בעיה זו נבנה **פרוטוקול 2PC** (Two Phase Commit). פרוטוקול זה בא להבטיח, שכאשר תנועה אינה מסתיימת בהצלחה בשרת כלשהו, המערכת תבטל באופן אוטומטי את העדכונים שבוצעו בכל השרתים ותתן הודעה מתאימה למשתמש. פרוטוקול זה הוסבר בפרק הקודם.

כללי Date להגדרת בסיס נתונים מבוזר

בשנת 1987 פרסם Codd מספר כללים המאפשרים להגדיר את רמת התמיכה בביזור בסיס הנתונים של מערכת מסוימת. כללים אלה שופרו ועודכנו אחר כך על ידי C.Date [DC90] והם מוצגים בהמשך. עקרון היסוד של כל הכללים הוא שמנקודת מבטו של המשתמש, מערכת מבוזרת צריכה להראות כמו מערכת מרכזית. כפי שנראה, כללים אלה מחמירים מאוד ואין כיום אף מערכת מסחרית העומדת בכל הכללים האלה. ארבעת הכללים האחרונים מנסים להבטיח את אי-תלות המערכת בחומרה, במערכת הפעלה, ברשת התקשורת ואפילו בבסיס הנתונים המקומי. אלו דרישות של מערכת אידיאלית ולכן לא מפליא שלא קיימת אף מערכת מסחרית העומדת בכל הכללים האלה.

עצמאות מקומית (Local Autonomy)

כלל זה קובע שכל השרתים במערכת מבוזרת צריכים לפעול בצורה עצמאית. כלומר, הנתונים המקומיים מנוהלים באחריות השרת המקומי וכל הפעולות המקומיות מבוצעות ומבוקרות בצורה מקומית. אם פעולה מסוימת ממוקדת בנתונים מקומיים, השרת המקומי צריך לבצע את הפעולה ללא כל פניות לשרתים אחרים. כלל זה מבטיח את העצמאות של כל שרת במערכת המבוזרת.

אי הסתמכות על שרת מרכזי

על מערכת מבוזרת לפעול ללא הסתמכות על שרת אחד מרכזי, כלומר אין שרת אחד היכול לגרום להשבתת כל המערכת. משמעות כלל זה היא שמערכת מבוזרת אמיתית אינה יכולה להתבסס על קטלוג המנוהל במחשב אחד בלבד ואינה יכולה להסתמך על שרת אחד כדי לטפל בסוגיות כגון בקרת תנועות, בקרת עדכונים בו-זמניים וניהול נעילות.

פעילות רציפה (Continuous Operation)

מערכת מבוזרת צריכה להבטיח את הפעולה הרציפה ללא צורך בהשבתת המערכת כתוצאה מאירועים שונים. למשל הוספת שרת חדש לרשת, סילוק שרת פעיל מהרשת, הזזה של מקטעים משרת לשרת וכד׳.

אי-תלות במיקום (Location Independency)

על המערכת לספק למשתמש שקיפות מלאה למיקום המקטעים. הוא צריך להיות מסוגל לגשת באותה צורה למקטע מסוים מכל שרת ברשת. מנקודת המבט שלו, הנתונים בהם הוא מטפל נמצאים תמיד בשרת המקומי.

אי-תלות למקטעים (Fragmentation Independency)

על המערכת לספק שקיפות מלאה למקטעים. צורת החלוקה של טבלה למקטעים צריכה להיות שקופה באופן מלא מנקודת מבט המשתמש. כמובן ששינויים בפיצול הטבלה למקטעים אינם צריכים להשפיע על תוכניות היישום.

אי-תלות בשכפול (Replication Independency)

על המערכת לספק שקיפות מלאה לעובדה שמקטע מסוים משוכפל ומנוהל ביותר מאשר שרת אחד. מנקודת מבטו של המשתמש, הוא אינו יכול לפנות בצורה ישירה לעותק מסוים, הוא אינו יכול להחליט איזה עותק הוא מבקש לעדכן וכד׳.

שאילתות מבוזרות (Distributed Query)

על המערכת לבצע אופטימיזציה של שאילתות מבוזרות, כלומר שאילתות המטפלות בנתונים המנוהלים בטבלאות הנמצאות בשרתים שונים.

תנועות מבוזרות (Distributed Transactions)

על המערכת לתמוך בביצוע תנועות מבוזרות, כלומר תנועות המעדכנות טבלאות המנוהלות במספר שרתים שונים. על התנועות המבוזרות והמקומיות לקיים את כללי ACID, כאילו הן היו תנועות מקומיות בלבד.

אי-תלות בחומרה (Hardware Independency)

על מערכת DRDBMS לפעול בצורה בלתי תלויה על התקני חומרה שונים.

אי-תלות במערכת הפעלה (Operating System Independency)

מערכת DRDBMS צריכה לפעול בצורה בלתי תלויה במערכת הפעלה מסוימת.

אי-תלות ברשת התקשורת (Network Independency)

מערכת DRDBMS צריכה לפעול בצורה בלתי תלויה בפרוטוקולים השונים לתקשורת ובתצורות השונות של רשתות התקשורת.

אי-תלות בבסיס הנתונים (Database Independency)

על המערכת לאפשר הפעלת מספר מערכות RDBMS שונות בשרתים המקומיים.

סיכום

המערכות לניהול בסיסי נתונים מבוזרים פותחו כדי לענות על צורך אמיתי: ביזור היחידות העסקיות של הארגון תוך קירוב הנתונים ככל שניתן אל המשתמשים. פוטנציאל משמעותי טמון ביכולת לבצע עיבוד מקומי בשילוב היכולת להסתכל על בסיס הנתונים המבוזר כעל ישות לוגית אחת.

יש להניח שנראה במהלך השנים הקרובות מערכות מבוזרות הטרוגניות, המבוססות על פלטפורמות חומרה שונות, מערכות הפעלה שונות ומערכות RDBMS מקומיות שונות. דווקא היכולת של מערכת מסוג DRDBMS להתמודד עם סביבות מחשוב אלו, היא זו המבטיחה את המשך המאמצים בשיפור ושכלול טכנולוגיה זו. יחד עם זאת, האתגרים העומדים בפני יצרני מערכות DRDBMS הם גדולים מאוד ועדיין המרחק בין הרצוי למצוי הוא גדול. יחד עם זאת, רוב המערכות המסחריות המובילות התקדמו כברת דרך ניכרת בנושא הביזור ומציגות יכולת לא מבוטלת בכל הקשור לביזור.

מן הראוי להתייחס גם לחסרונות של טכנולוגיית ביזור בסיסי הנתונים. החיסרון העיקרי נובע מהאטה בביצוע העבודה המקומית, בגלל הצורך לעדכן גם נתונים במחשבים אחרים היכולים להיות מחוברים דרך רשתות תקשורת מקומיות מהירות (LAN), או רשתות תקשורת רחבות איטיות יותר (WAN). כאשר מערכת DRDBMS אינה מספקת רמה גבוהה של שקיפות, בניית היישום מורכבת יותר ורגישה לעיצוב בסיס הנתונים המבוזר.

העובדה שכל אתר מנהל בסיס הנתונים משלו, מחייבת שיהיו בו גם אנשי מקצוע מקומיים, בניגוד למצב שבו בסיס הנתונים מנוהל בשרת המרכזי בלבד. עיצוב בסיס נתונים מבוזר מורכב יותר מעיצוב בסיס נתונים מרכזי ומחייב שיקולים של עומס על רשת התקשורת, עוצמת השרתים המקומיים, רמת השכפול הנדרשת, פרוצדורות שחזור מורכבות ועוד.

שאלות חזרה ותרגילים

שאלות חזרה

1. הסבר מהו בסיס נתונים מבוזר.
2. מה ההבדלים העיקריים בין מערכת לניהול בסיס נתונים מרכזי למערכת לניהול בסיס נתונים מבוזר?
3. הגדר את המושג של מקטע (Fragment). לאיזה מטרה הוא משמש? איזה סוגי מקטעים קיימים? מדוע המקטעים חייבים להיות לא חופפים? מדוע המקטעים מהווים רעיון חשוב בביזור בסיס הנתונים?
4. הגדר את מושג שיכפול הנתונים. איזה יתרונות ניתן להשיג כתוצאה מהשיכפול ואיזה חסרונות יש לשיכפול?

5. הסבר את מושג השקיפות בהקשר של בסיסי נתונים מבוזרים. מהי שקיפות המיקום ושקיפות לשכפול?

6. מדוע אופטימיזציה של שאילתה בסביבה מבוזרת היא מורכבת יותר מאשר בסביבה מרכזית?

7. מהן התכונות ההופכות את בסיסי הנתונים הטבלאיים למועמדים טבעיים לביזור?

8. האם בסיס נתונים המנוהל במחשב מרובה מעבדים, כלומר מכיל מספר CPU, הוא בסיס נתונים מבוזר. איזה תנאי חייב להתקיים כדי שנוכל לקרוא לו בסיס נתונים מבוזר?

9. מה ההבדל בין מערכת מבוזרת הומוגנית והטרוגנית? מדוע קיים קושי לטפל בבסיס נתונים מבוזר הטרוגני?

10. מדוע יש צורך בפרוטוקול 2PC בניהול בסיס נתונים מבוזר? הסבר כיצד פרוטוקול זה פועל.

תרגילים

נתונה הסכימה הבאה של בסיס נתונים:

- **עובדים** (מספר זהות, שם עובד, עיר מגורים, כתובת, טלפון, מספר מחלקה בה עובד)
- **מחלקות** (מספר מחלקה, שם מחלקה, מספר זהות מנהל המחלקה)
- **פרויקטים** (מספר פרויקט, שם פרויקט, סוג פרויקט, מספר מחלקה אחראית)
- **שעות מצטברות** (מספר עובד, מספר פרויקט, מספר שעות מצטברות)

נניח שהחברה היא חברה מבוזרת הפועלת בשלוש ערים: חיפה, תל אביב ובאר שבע. הצג איזה מקטעים יש לבנות עבור הטבלאות השונות כדי לבזר את בסיס הנתונים. הסבר שתי חלופות שונות לביצוע השאילתה הבאה: מהו סה"כ השעות שהושקעו בכל הפרויקטים מסוג מסוים.

פרק 18

בסיסי נתונים מוכווני אובייקטים (Object Oriented Databases)

1. מבוא
2. סקירת החסרונות העיקריים של המודל הטבלאי
3. הרקע להתפתחות סביבות מוכוונות אובייקטים
4. מושגי יסוד במודל האובייקטים
5. מערכות ODBMS
6. סקירת ההבדלים בין מערכות RDBMS לבין מערכות ODBMS
7. מערכות Object-Relational DBMS
8. סיכום

מבוא

במשך השנים נוצרה הפרדה בתחום טכנולוגיית המידע. הנדסת התוכנה מטפלת בהיבט **הדינמי** של היישומים וטכנולוגיית בסיסי הנתונים מטפלת בהיבט **הסטטי** של היישומים, כלומר בניהול הנתונים. למעט מספר נושאים יוצאים מן הכלל (כמו פרוצדורות בסיסי נתונים, ומזניקים [Triggers]), בסיס הנתונים משמש כמאגר פסיבי לאחסון וניהול הנתונים לאורך זמן. היישומים שולפים ומעדכנים את הנתונים באמצעות פקודות SQL המשובצות בתוכנית היישום (Embedded SQL) או על ידי שימוש בממשק התכנות (Call Level Interface) של בסיס הנתונים. שפת SQL נבנתה והותאמה במיוחד לטיפול נוח במבני נתונים פשוטים, טבלאות, ובמספר טיפוסי נתונים בסיסיים, מספרים שלמים, מספרים עשרוניים, תווים וכד'. הפרדה זו בין ההיבט הדינמי וההיבט הסטטי מתאימה לרוב היישומים הרגילים שעוסקים בעיבוד נתונים ותנועות אולם אינה מתאימה למיגוון גדול של יישומים חדשים שהחלו להופיע.

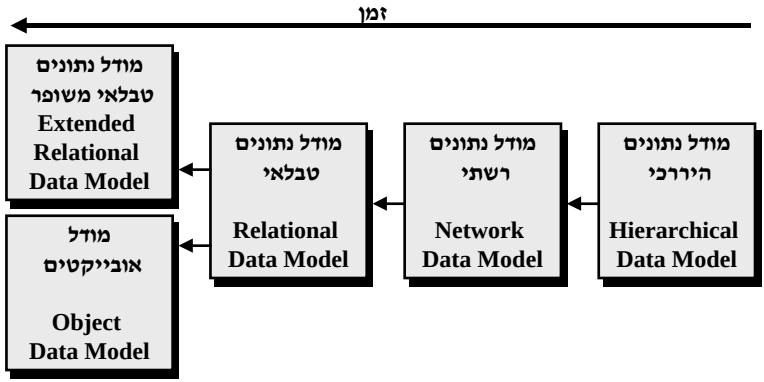
ההתפתחות הבלתי פוסקת בטכנולוגיית המידע, בחומרה ובתוכנה הביאה להופעת מיגוון רחב של יישומים חדשים ומורכבים בהרבה מהיישומים המקובלים של עיבוד נתונים. דור חדש זה כולל יישומים כגון:

- תיב"מ (CAD/CAM - Computer Aided Design and Manufacturing)
- הנדסת תוכנה (CASE - Computer Aided Software Engineering)
- מערכות מומחה (Expert Systems)
- מערכות מידע גיאוגרפיות (GIS - Geographical Information Systems)
- אוטומציה משרדית (Office Automation)
- הוצאה לאור (Computer Aided Publishing)
- קבוצות עבודה (GroupWare)
- תזרימי עבודה (Workflow)
- אינטרנט (Internet)
- מסחר אלקטרוני (Electronic-Commerce)
- ניהול רשתות תקשורת (Network Management)

התפוצה הרבה של מערכות RDBMS הביאה לכך שמפתחי היישומים ביקשו לנהל בבסיס הנתונים הטבלאי נתונים בעלי מבנה לא שיגרתי: שרטוטים, מפות, תמונות שנוצרו כתוצאה מסריקה, סרטי וידאו, גליונות אלקטרוניים, מסמכים שנוצרו על ידי מעבדי תמלילים וכד'. מה שהתברר הוא שמערכות RDBMS אינן מתאימות לטיפול בסוגי הנתונים והאובייקטים שהיישומים האלה צריכים לנהל. אחד המאפיינים העיקריים של יישומים אלה הוא שהם מנהלים נתונים הקשורים ביניהם במיגוון רחב של קשרים מורכבים, שלא ניתן לבטא אותם באופן טבעי על ידי ערכים בטבלאות. מערכות RDBMS מתאימות לטפל באובייקטים פשוטים שניתן לייצג על ידי שורות ועמודות המכילות ספרות או תווים, אולם אינן מתאימות לטפל באובייקטים מורכבים.

הקשיים בשימוש במערכות RDBMS עבור היישומים החדשים במקביל לתפוצה הרחבה של שפות תכנות מוכוונות אובייקטים (Object Oriented Programming Languages), הביאו את החוקרים באוניברסיטאות ובחברות מסחריות לחיפוש אחר מערכות לניהול בסיסי נתונים המתאימות יותר ליישומים וסביבות התכנות החדשות. עם השנים התפתחו שתי מגמות עיקריות, אחת מהפכנית יותר והשנייה התפתחותית יותר:

- החלפת המודל הטבלאי במודל חדש, מודל האובייקטים. מגמה זו הביאה להופעת **מערכות לניהול בסיסי נתונים מוכווני אובייקטים** ODBMS (Object Oriented DataBase Management Systems), מערכות הפועלות בהרמוניה עם סביבות התכנות מוכוונות האובייקטים כגון C++, Smalltalk, Java ואחרות.
- הרחבה ושיפור המודל הטבלאי תוך שילוב רעיונות הבאים ממודל האובייקטים בתוך המודל הטבלאי. למודל משופר זה מקובל לקרוא ERDM (Extended Relational Data Model). מגמה זו הביאה להופעת **מערכות משולבות אובייקטים – טבלאות** ORDBMS (Object Relational DataBase Management Systems), שהן למעשה גרסאות משופרות של מערכות RDBMS. חלק מהרעיונות להרחבה ושיפור המודל הטבלאי עם מושגים הלקוחים ממודל האובייקטים כבר מופיע בטיוטה לתקן SQL3.



תרשים 18.1: התפתחות המודלים לניהול בסיסי נתונים על ציר הזמן.

נכון לזמן כתיבת שורות אלו נראה שמערכות ODBMS לא הצליחו להפוך לדור הבא של מערכות ניהול בסיסי נתונים ולהחליף את מערכות RDBMS. למערכות אלו יש כיום קהל מצומצם יחסית של מפתחי יישומים המעדיף להשתמש בהם לפיתוח יישומים מיוחדים, אולם הן כמעט ואינן משמשות את יישומי עיבוד הנתונים והתנועות. לעומת זאת, התפוצה הרחבה של מערכות RDBMS והשיפורים שלהן בכיוון ORDBMS חודרות יותר ויותר אל תחום היישומים החדשים, במקביל לשליטתן הבלתי מעורערת בתחום יישומי עיבוד הנתונים והתנועות הרגילים. לדוגמה מערכות RDBMS מהוות תשתית בחלק מאתרי אינטרנט הגדולים והעמוסים ביותר בעולם, אתרים המנהלים נתוני מולטימדיה מגוונים, ומשמשות בסיס לחלק גדול מאוד מיישומי המסחר האלקטרוני ההולך ומתפתח באינטרנט.

בפרק זה נסקור את התוצר של שתי מגמות אלו, ונציג את מודל האובייקטים ומערכות ODBMS. גם נציג את השיפורים במודל הטבלאי ובמערכות ORDBMS.

בפרק זה

- נסקור את החסרונות של המודל הטבלאי בעיקר בהקשר לצורך לתמוך ביישומים חדשים המנהלים מיגוון סוגי נתונים – תמונות, מפות, קול, וידאו, טקסט וכד'.
- נציג בקצרה את הרקע להתפתחות מודל האוביקטים מתוך שפות התכנות מוכוונות האובייקטים והארגונים העוסקים כיום בהגדרת המונחים בתחום האובייקטים.
- נסקור את מושגי היסוד של מודל האובייקטים: מהו אובייקט, כיצד ניתן לזהותו באופן חד-משמעי, תכונות האובייקט, פרוצדורות, אריזת התכונות והפרוצדורות, מחלקות של אובייקטים, היררכיה של אובייקטים, עיקרון הירושה ועוד. מושגים אלה לקוחים משפות תכנות מוכוונות אובייקטים שהפכו לנפוצות יותר ויותר, במיוחד עם הצורך לבנות יישומים מורכבים וגדולים.
- נציג את מערכות ODBMS המובילות כיום. מערכות אלו מתמודדות עם אחד האתגרים המורכבים ששפות תכנות מוכוונות אובייקטים צריכות ענות עליהם: כיצד לנהל את האובייקטים הזמניים הנוצרים תוך כדי ריצת התוכנית, ולהפוך אותם לאובייקטים קבועים הנשמרים לאורך זמן.
- נסקור את ההבדלים העיקריים בין מערכות RDBMS לבין מערכות לניהול בסיסי נתונים מוכווני אובייקטים.
- נסקור את מערכות ORDBMS המהוות שיפור והרחבה של המודל הטבלאי לכיוון מודל האובייקטים. נציג את הרעיון המרכזי והוא היכולת להגדיר טיפוסי נתונים מופשטים הכוללים את הנתונים וגם פונקציות הפועלות על הנתונים.

סקירת החסרונות העיקריים של המודל הטבלאי

בספר זה הדגשנו את היתרונות העיקריים של המודל הטבלאי והצגנו את מערכות ניהול בסיסי נתונים טבלאיים – RDBMS – המבוססות עליו. נחזור בקצרה על היתרונות העיקריים של המודל:

- **יסודות תיאורטיים חזקים:** המודל הטבלאי מבוסס על עקרונות פורמליים הלקוחים מתורת המחלקות.
- **שפת SQL:** SQL היא שפה לא פרוצדורלית רבת עוצמה הנתמכת על ידי תקן בינלאומי. השפה מאפשרת להגדיר את בסיס הנתונים, לטפל בנתונים הן בצורה אינטראקטיבית והן מתוך שפת תכנות מארחת, להגדיר את כללי אבטחת המידע ולפקח על ביצוע התנועות. מאחר והשפה אינה פרוצדורלית, היא מאפשרת הגדרה נוחה של מה לעשות עם הנתונים ולא איך לגשת לנתונים.

- **שירותים מתקדמים:** מערכות RDBMS מודרניות מספקות מיגוון רחב של שירותים וביניהם ניהול קטלוג נתונים מרכזי, שרותי אבטחת מידע, תמיכה בתנועות, ניהול נעילות למניעת עדכונים בו-זמניים, גיבוי תוך כדי פעולה, התאוששות מהירה ועוד.
- **תהליך עיצוב פורמלי:** המודל הטבלאי מספק אוסף של כללי עיצוב פורמליים באמצעותם ניתן לקבוע אם העיצוב של בסיס הנתונים הוא תקין ואינו סובל מתופעות לא רצויות. תהליך זה, הנקרא תהליך נירמול הנתונים, הביא לשיפור ההבנה של עקרונות עיצוב בסיסי הנתונים.

יתרונות אלה הפכו את תעשיית מערכות RDBMS לתעשייה המגלגלת סכומי עתק (ההערכה היא שבשנת 2000 המחזור של תעשייה זו יהיה בהיקף של כ-11 מיליארד דולר, מתוך מחזור כולל של 13 מיליארד דולר בתחום בסיסי נתונים). יחד עם היתרונות שלו, יש למודל הטבלאי גם לא מעט חסרונות. נסקור את העיקריים שבהם:

- **ייצוג ישויות על ידי טבלאות בלבד:** המודל הטבלאי מבוסס על מבנה אחד בלבד, הטבלה. ייצוג המציאות על ידי המודל הטבלאי מתבצע על ידי מיפוי הישויות לאוסף של טבלאות. לפעמים ישות אחת במציאות, כגון טופס רישום לקורס או טופס הזמנת ציוד, מפורקת למספר רב של טבלאות שונות, דבר המקשה על הבנת המציאות מתוך התבוננות במודל. השחזור של ישות במציאות מחייב ביצוע אוסף של פעולות Join, פעולות שהן יקרות במונחים של משאבי מחשב ולא יעילות מבחינת הביצועים.
- **ייצוג קשרים על ידי ערכים בלבד:** הקשרים בין הישויות מיוצגים על ידי ערכים בעמודות. לדוגמה הקשר בין סטודנטים לקורסים, שהוא קשר רב-רב-ערכי במציאות מיוצג במודל על ידי שלוש טבלאות שונות: סטודנטים, קורסים וציונים. שיטת ייצוג זו נוחה ליישומי עיבוד נתונים רגילים, אולם מציבה קושי בייצוג קשרים בין אובייקטים שאינם מבוססים על ערכים.

לדוגמה, מערכת תיב"מ אלקטרונית מטפלת בכמות גדולה של רכיבים אלקטרוניים הקשורים ביניהם בקשרים מסועפים הנובעים מהשתתפותם במעגלים חשמליים, במספר רב של סכימות חשמליות, או בסכימות עריכה פיסית של המעגל האלקטרוני וכד'. קיים קושי לבטא קשרים אלה על ידי טבלאות וערכים בלבד. דוגמאות נוספות יכולות להיות שרטוטים של חדרים בבניין בהם הקשר בין האובייקטים (כסא, שולחן, מנורה וכד') יכול לנבוע מהקרבה הפיסית בין האובייקטים; מפות גיאוגרפיות בהן הקשר בין נקודות יישוב יכול לנבוע מהעובדה שקיים כביש העובר ביניהן; מערכות המנהלות קלסתרונים של חשודים בהן הקשר בין התמונות יכול לנבוע מהדמיון החזותי בין החשודים ועוד. בכל הדוגמאות האלו, ייצוג הקשרים בין האובייקטים על ידי טבלאות וערכים בעמודות אינו טבעי.

- **הומוגניות:** מודל הנתונים הטבלאי הינו מודל הומוגני, כלומר מודל המניח הומוגניות הן בין השורות והן בין העמודות. הומוגניות בין השורות, משמעותה היא שכל השורות בטבלה הן בעלות מבנה זהה. הומוגניות בין העמודות, משמעותה היא שכל עמודה היא בעלת טיפוס נתונים מסוים, כלומר הערכים של העמודה נובעים ממרחב ערכים אחד בלבד. בהצטלבות שבין שורה ועמודה חייב להופיע ערך אחד

בלבד. הנחות יסוד אלו של המודל הטבלאי מתאימות לרוב היישומים הרגילים של עיבוד נתונים. הבעיה היא שהומוגניות זו אינה מתאימה ליישומים המנהלים נתונים מורכבים – שרטוטים, מפות, תמונות, מידע מרחבי וכד'.

- **מיגוון מצומצם של טיפוסי נתונים נתמכים:** המודל הטבלאי תומך במיגוון מצומצם של טיפוסי נתונים וביניהם מספרים שלמים, מספרים עשרוניים, מחרוזת של תווים ומשתנים בוליאניים. בהמשך, הוסיפו למודל גם תמיכה בטיפוסי נתונים כגון תאריכים, זמן וכסף. נדגיש ונזכיר שמאחורי כל טיפוס נתונים עומדים גם האופרטורים הנתמכים. למשל, על טיפוס נתונים נומרי השפה תומכת באופרטורים כגון חיבור, חיסור, כפל, חילוק וכד'. על טיפוס נתונים של תאריך השפה תומכת באופרטורים של חיבור תאריכים, חיסור תאריכים, הפרש בין תאריכים וכד'. המודל אינו מאפשר הרחבת טיפוסי הנתונים על פי צרכי היישומים והגדרת האופרטורים שניתן לבצע על טיפוסי הנתונים החדשים. במובן הזה ניתן לומר שהמודל הטבלאי תומך במספר סופי ומוגדר מראש של טיפוסי נתונים ואינו מאפשר הרחבה דינמית של טיפוסי הנתונים והאופרטורים שלהם. המיגוון המצומצם של טיפוסי הנתונים הנתמכים על ידי המודל הטבלאי מקשה מאוד על ניהול נתונים מורכבים יותר כגון סדרות עתיות, מידע מרחבי, מידע חזותי וכד'.

- **מיגוון מצומצם של אילוצים:** המודל הטבלאי תומך במספר קטן של סוגי אילוצים וביניהם חד-ערכיות של מפתחות, חובת קיום ערכים בעמודה ושלמות הקשרים בין טבלאות. בשנים האחרונות הרחיבו את המודל לתמוך באילוצים מורכבים יותר המבוטאים על ידי Assertions, אולם זוהי תמיכה מוגבלת בלבד. המודל אינו תומך באילוצים מורכבים כגון שמוליך נחושת במעגל חשמלי אינו יכול ליצור מצב של קצר עם מוליך אחר, עמוד תומך של בניין חייב להופיע באותו מקום בכל הקומות, כל חשוד בקלסתרון חייב להופיע עם צילום של הפנים וצילום של הפרופיל וכד'.

- **מיגוון מצומצם של פעולות על הטבלאות:** המודל הטבלאי בכלל ושפת SQL בפרט תומכים במיגוון מצומצם של פעולות על הנתונים ובנייהם איחזור שורות, ביטול שורות, הוספת שורות ועדכון שורות. למרות היות שפת SQL שפה רבת עוצמה, קיים מיגוון רחב של יישומים שהיא אינה מתאימה עבורם. בין אלה נוכל למנות יישומים המבצעים סריקה של מבנים רקורסיביים כמו עצי מוצר, יישומים המבצעים תנועה בתוך סכימה חשמלית על פי הקרבה בין הרכיבים, יישומים שצריכים להציג את האובייקטים הנמצאים בתוך פוליגון נתון וכד'. שפת SQL אינה מאפשרת ליישום לשלוט על הניווט בתוך בסיס הנתונים. ניווט זה נקבע באופן בלעדי על ידי רכיב האופטימיזציה. עבור חלק מהיישומים, היעדר היכולת לשלוט באופן ישיר על הניווט בתוך בסיס הנתונים מהווה חסרון ולא יתרון.

חסרונות אלה של מערכות RDBMS, במקביל להופעת יישומים חדשים מבוססי מולטימדיה ולתפוצה הרחבה שזכו לה שפות תכנות מוכוונות אובייקטים, הביאו לחיפוש אחר פתרונות חדשים. לפני שנציג את הפתרונות השונים, נסקור מהם הגופים העיקריים הפעילים כיום בהגדרת מודל האובייקטים ואת מושגי היסוד של מודל האובייקטים.

הרקע להתפתחות סביבות מוכוונות אובייקטים

שורשי המושגים והרעיונות של תכנות מוכוון אובייקטים מקורם בשפת תכנות Simula שפותחה בסוף שנות ה-60 וייעודה העיקרי היה לאפשר פיתוח של מערכות סימולציה. בשלב מאוחר יותר, בתחילת שנות ה-70, חוקרים במרכז המחקר PARC (Palo Alto Research Center) של חברת Xerox פיתחו את שפת Smalltalk. שפה זו היא הראשונה שהגדירה רבים מהעקרונות והרעיונות שנחשבים כיום ליסודות של מודל האובייקטים. שפה זו נחשבת לשפה מוכוונת אובייקטים טהורה. בשלב מאוחר יותר שפת C הפכה לשפה נפוצה מאוד בקרב משתמשי מערכות Unix והורחבה לשפת ++C. שפה זו נחשבת לשפה מוכוונת אובייקטים מעורבת, כלומר משלבת את הרעיונות החדשים בתוך שפת תכנות קיימת. בשנים האחרונות פרצה לשוק שפת Java, שגם היא שפה מוכוונת אובייקטים בעלת דמיון רב ל- ++C.

להבדיל מהמודל הטבלאי, שהוגדר בצורה פורמלית על ידי Codd, עדיין אין הסכמה רחבה והגדרה פורמלית חד-משמעית לגבי חלק ניכר מהמושגים והרעיונות העומדים בבסיס תפישת מודל האובייקטים. במשך השנים ניסו מספר גופים להגדיר ולפתח את מודל האובייקטים בצורה פורמלית. בין גופים אלה ניתן למנות את הגופים העיקריים הבאים:

- **מכון התקינה האמריקאי ANSI**: מכון זה הקים קבוצת עבודה בשם X3H7 - Object Information Management, שהפיקה מסמך בשנת 1994 המתאר את התכונות העיקריות של מערכות מוכוונות אובייקטים.
- **ארגון OMG (Object Management Group):** ארגון ללא מטרות רווח שהוקם בשנת 1989 במטרה להרחיב, לשפר, לקדם ולהביא לסטנדרטיזציה בכל הקשור לשימוש באובייקטים. בארגון זה חברים כיום מעל 400 חברות מסחריות וגופי מחקר שונים. בין חבריו המובלים ניתן למנות חברות כגון יבמ, Microsoft, Compaq ואחרים. ארגון זה אינו ארגון תקינה פורמלי, כדוגמת ANSI או ISO ולכן הוא ממוקד יותר ביצירת תקנים בפועל, שיאומצו בשלב מאוחר יותר על ידי ארגוני התקינה הבינלאומיים. ארגון זה פרסם בשנת 1992 את המסמך Object Management Architecture, שמטרתו להביא להסכמה לגבי המושגים והמונחים העיקריים במודל האובייקטים. המסמך מגדיר את מודל האובייקטים OM (Object Model), את המנגנון המתווך ORB (Object Request Broker) המאפשר שיתוף פעולה בין אובייקטים הטרוגניים הפועלים בסביבה מבוזרת ואת השירותים והמנגנונים המשותפים Object Services and Common Facilities. מסמך זה היווה את הבסיס לפיתוח ארכיטקטורת CORBA (Common Object Request Broker Architecture) המאפשרת שיתוף פעולה בין אובייקטים שונים, ארכיטקטורה הנתמכת כיום על ידי רוב יצרני התוכנה והחומרה הגדולים (למעט Microsoft).

❖ **ארגון ODMG (Object Database Management Group):** ארגון שהוקם על ידי מספר יצרני מערכות ODBMS. ארגון זה ניסה לפתח מספר הגדרות מתוך מטרה לאפשר ניידות של יישומים בין מערכות ODBMS שונות ואת יכולת העבודה השיתופית של מערכות אלו. ארגון זה הפיק מסמך הנקרא ODMG-93 Specification ובו תיאור של מודל האובייקטים OM (Object Model); של השפה להגדרת אובייקטים ODL (Object Definition Language) שאינה תלויה בשפת תכנות כלשהי; של שפת השאילתות של האובייקטים OQL (Object Query Language) המזכירה במקצת את שפת SQL; ושל הממשק שבין מערכות ODBMS לבין שתי שפות התכנות מוכוונות האובייקטים C++ ו-Smalltalk. ארגון זה ממשיך לפעול ולעדכן את ההגדרות.

נכון להיום עדיין לא התגבש תקן מוסכם בכל הקשור למונחים ולצורות היישום של מודל האובייקטים.

מושגי יסוד במודל האובייקטים

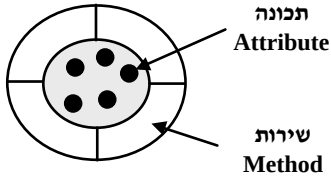
נתחיל בסקירת מושגי היסוד והרעיונות העיקריים עליהם מבוססים כל המודלים מונחי האובייקטים (Object Oriented Models).

אובייקט (Object)

אבן הבניין העיקרית של כל מודל האובייקטים הוא האובייקט. כל ישות בעולם הממשי ניתנת לייצוג **כאובייקט**. לדוגמה, אובייקט יכול להיות עובד, מכונית, שרטוט, מסמך, גרף, הקלטת קול וכדw.

אובייקט Object	***אובייקט* מוגדר כישות הניתנת לזיהוי באופן חד-ערכי ומורכבת משני מרכיבים: תכונות (Attributes) המתארות את מצב האובייקט (Object State) ושירותים (Methods) המתארים את הפעולות שהאובייקט מסוגל לבצע.**

הגדרה זו רחבה יותר מההגדרה של **קבוצת ישות** במודל ישויות-קשרים. נזכיר שישות הכילה תכונות בלבד, ואילו אובייקט מכיל תכונות ושירותים שהינם לוגיקה המבצעת פעולות כלשהן על האובייקט או בשם האובייקט. מתוך הגדרה פשוטה זו ניתן להבחין בשוני העיקרי של מודל האובייקטים לעומת המודלים האחרים. במודלים האחרים, והמודל הטבלאי ביניהם, קיימת אבחנה ברורה בין הנתונים לבין הלוגיקה. הנתונים מנוהלים בבסיס נתונים והלוגיקה ממומשת בתוכניות היישום, כלומר בתוכנה. במודל האובייקטים מחיצה זו נשברת, כי האובייקט מכיל את שניהם, גם נתונים וגם לוגיקה.



תרשים 18.2: אובייקט עם חמש תכונות וארבעה שירותים.

זיהוי אובייקט (Object Identification)

להבדיל מהגדרת המפתח של הטבלה במודל הטבלאי, מודל האובייקטים משתמש במנגנון שונה, המזהה של האובייקט.

מזהה האובייקט **Object Identifier**	**לכל אובייקט יש מזהה חד-ערכי** OID **(**Object Identifier**), המזהה אותו באופן חד-משמעי. מזהה זה נוצר על ידי המערכת ואינו תלוי בתכונות האובייקט.**

במודל הטבלאי המפתח מוגדר כאוסף של עמודות המזהות באופן חד-משמעי שורה כלשהי ולכן הוא מהווה חלק בלתי נפרד מהעמודות של הטבלה. לעומת זאת במודל האובייקטים, **מזהה האובייקט** OID אינו תלוי בתכולה של האובייקט וניתן להשתמש בו לזיהוי חד-משמעי של כל סוגי האובייקטים כגון עובדים או מכוניות, אך גם שרטוטים, צורות גיאומטריות, תמונות, רכיבים אלקטרוניים וכד׳. לדוגמה, שרטוט יכול להכיל שני משולשים זהים לחלוטין המופיעים במקומות שונים. כל אחד מהם הוא אובייקט ולכן לכל אחד מהם יש OID שונה וחד-ערכי.

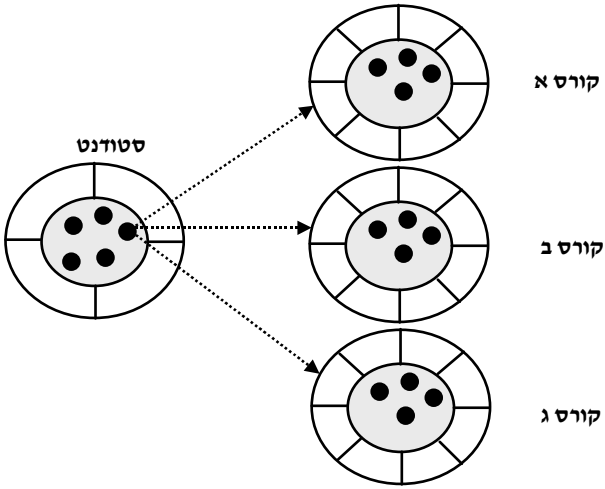
מזהה האובייקט ניתן על ידי המערכת כשנוצר אובייקט, והוא אינו ניתן לשינוי כל עוד האובייקט. יתר על כן, גם לאחר ביטול האובייקט, המזהה שלו לא ניתן לאובייקט אחר. נשים לב שלהבדיל ממפתח במודל הטבלאי שהינו חד-ערכי בתוך הטבלה אולם יכול לחזור על עצמו בטבלאות שונות, מזהה האובייקט, OID, הינו חד-ערכי ברמת המערכת כולה. המזהה הינו מנגנון יעיל וקומפקטי יותר ממפתח. מפתח יכול לפעמים להיות מורכב ממספר רב של עמודות. לעומת זאת, המזהה הינו בעל מבנה סטנדרטי ובדרך כלל קצר יחסית. המזהה גם יכול להיות שקוף מבחינת המשתמש ומשמש רק את המערכת, להבדיל מהמפתח שהינו חשוף למשתמש, מאחר והוא מורכב מעמודות המהוות חלק מהטבלה.

תכונות של אובייקט (Object Attributes)

לכל אובייקט יש אוסף של **תכונות** (Attributes), הנקראות לפעמים **משתני מופע** (Instance Variables), היכולים לקבל ערכים שונים לכל מופע של אובייקט. ההבדל בין אוסף העמודות של המודל הטבלאי לבין אוסף התכונות של האובייקט, הוא בעושר טיפוסי הנתונים. במודל הטבלאי, טיפוסי הנתונים של העמודות מצומצם למדי ויכול להיות מספר שלם, מספר עשרוני, מחרוזת תווים, תאריכים וכד׳. המודל הטבלאי דורש

גם שכל עמודה תכיל ערך אחד בלבד מתוך מרחב הערכים האפשרי של העמודה. תהליך הנירמול דואג לכך, שכל תכונה תהיה בעלת ערך בודד, וערכים שחוזרים על עצמם יועברו לטבלאות נפרדות.

במודל האובייקטים תכונה יכולה להיות **תכונה פשוטה** (Simple Attribute) שהיא בעלת טיפוס נתונים בסיסי (Primitive Data Type) ומקבלת ערך בודד, אולם היא יכולה גם להיות **תכונה מורכבת** (Complex Attribute) המבוססת על טיפוס נתונים מורכב ולקבל אוסף ערכים. ניתן להגדיר את מרחב הערכים של תכונה כאובייקט אחר, דבר שאיננו אפשרי במודל הטבלאי. לדוגמה, נניח שהתכונות של האובייקט סטודנט הן מספר סטודנט, שם סטודנט, כתובת מגורים, רשימת החוגים בהם הוא רשום ורשימת קורסים אותם הוא סיים. שתי התכונות האחרונות הן מסוג Reference Attribute המכילות הצבעה לאובייקטים אחרים (חוגים, קורסים).



תרשים 18.3: דוגמה לאובייקט עם תכונה שמרחב הערכים שלה הוא אובייקט אחר.

תרשים 18.3 מציג את האובייקט סטודנט עם התכונה קורסים שהוא סיים, תכונה מורכבת שמרחב הערכים שלה הוא אובייקט מסוג קורס. תכונה זו מצביעה בדוגמה על שלושה קורסים שונים שהסטודנט סיים.

למרות שברגע ראשון ניתן לחשוב שתכונות אלו מקבילות לרעיון המפתח הזר במודל הטבלאי, קיים שוני מהותי בין שתי תפישות אלו. למעשה, תכונות אלו מכילות רשימה של OID של האובייקטים הרלוונטיים ולא את המפתחות של השורות המתאימות בטבלאות אחרות. שיטה זו מאפשרת גמישות רבה בניהול קשרים בין אובייקטים ללא כל התייחסות לערכים של האובייקטים.

הגדרת תכונות שמרחב הערכים שלהן הוא אובייקטים אחרים הינה דרך אלגנטית לייצג קשרים מורכבים בין אובייקטים, ללא צורך בשימוש בטבלאות קשר מיוחדות. במובן הזה ניתן לומר שמודל האובייקטים תומך בייצוג ישיר של קשרים בין אובייקטים שונים.

שירותים (Methods)

כל אובייקט מסוגל לספק שירותים כלשהם.

שירות * **Method**	***שירות*** **מוגדר כפעולה שהאובייקט מסוגל לבצע. לכל שירות שני מרכיבים - הממשק של השירות והיישום של השירות.** ***הממשק*** **(**Method Interface**) מגדיר את שם השירות, מהם הפרמטרים לקלט ומהם הפרמטרים לפלט.** ***היישום*** **(**Method Implementation**) מכיל את הלוגיקה של השירות, אשר כתובה בשפת תכנות כלשהי.**

***הערה**: המונח Method מתורגם בדרך כלל כ'**שיטה**' (עבור שפות מוכוונות אובייקטים, כמו ++C, למשל). עם זאת, בחרנו להשתמש כאן במונח '**שירות**', מכיון שהוא מבטא בצורה טובה יותר את הכוונה.

אוסף כל השירותים שהאובייקט מספק מגדיר את **התנהגות האובייקט** (Object Behavior). שירות יכול להפעיל שירותים אחרים, הן של אותו אובייקט והן של אובייקטים אחרים.

לדוגמה נניח שהאובייקט סטודנט תומך בשירותים הבאים: יצירת סטודנט חדש, שינוי נתוני סטודנט, רישום סטודנט לקורס, רישום סטודנט לחוג, הצגת ציוני הסטודנט, הצגת רשימת הקורסים שהסטודנט סיים בהצלחה וכד'. לדוגמה הממשק של השירות רישום סטודנט חדש לקורס יהיה:

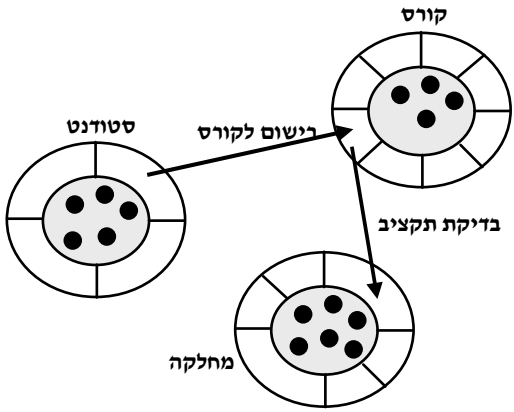
Course.Add_New_Student (Course_Number, Student_Number, Status)

השירות רישום סטודנט חדש לקורס של האובייקט קורס מגדיל את מספר הסטודנטים הרשומים לקורס מסוים. נשים לב לצורת הרישום המתחילה בשם האובייקט, שם השירות ורשימת הפרמטרים. במקרה זה הפרמטרים מספר קורס ומספר סטודנט מהווים קלט לשירות ואילו הפרמטר סטטוס הוא פרמטר פלט באמצעותו השירות מודיע אם הרישום בוצע בהצלחה.

הדרך היחידה לבקש מהאובייקט לבצע שירות כלשהו היא רק על ידי שיגור **הודעה** (Message) ובה שם השירות והפרמטרים המתאימים. משמעות הדבר היא שהאובייקטים "משוחחים" זה עם זה על ידי שיגור הודעות וקבלת תשובות.

תרשים 18.4 מדגים מצב שבו האובייקט סטודנט שולח הודעה רישום סטודנט חדש לקורס לאובייקט קורס כדי להגדיל את מספר הסטודנטים הרשומים לקורס מסוים. תוך כדי ביצוע השירות הזה, האובייקט קורס שולח הודעה לבדיקת תקציב לאובייקט מחלקה. האובייקט סטודנט אינו מודע כלל לצורה בה מתבצע השירות רישום סטודנט חדש לקורס ולעובדה שתהליך הרישום מבצע בדיקה תקציבית של המחלקה המציעה את הקורס.

הפנייה לאובייקט להפעלת שירות על ידי שיגור הודעה, מאפשרת את הסתרת המבנה הפנימי של האובייקט ואת צורת היישום של השירותים שלו.



תרשים 18.4: הודעות בין אובייקטים.

כמיסה – אריזת המצב וההתנהגות (Encapsulation)

כפי שראינו, לכל אובייקט יש **מצב** (State) המיוצג על ידי ערכי תכונות האובייקט בנקודת זמן כלשהי; **והתנהגות** (Behavior) המיוצגת על ידי אוסף השירותים שהאובייקט מבצע ומספק.

המצב וההתנהגות של האובייקט ארוזים יחדיו וניתן לגשת אליהם רק באמצעות הודעות הפונות לממשק של השירות. לעיקרון זה שבו המצב וההתנהגות ארוזים יחדיו מקובל לקרוא בשם **כמיסה** (Encapsulation), או **כימוס**.

כמיסה הוא תפישה שהתפתחה במשך השנים כתוצאה ממחקר בתחום הנדסת התוכנה. מטרתו העיקרית היתה לספק רמה גבוהה של מודולריזציה, או הפרדה גבוהה ככל הניתן בין רכיבי תוכנה שונים. ההנחה הבסיסית היא שככל שתוכנה תהיה מורכבת מרכיבים עצמאיים (Software Components) המבצעים את משימותיהם בצורה שאינה שקופה לרכיבים אחרים, הפיתוח והתחזוקה יהיו קלים יותר והתוכנה שתתקבל תהיה בעלת רמת אמינות גבוהה יותר.

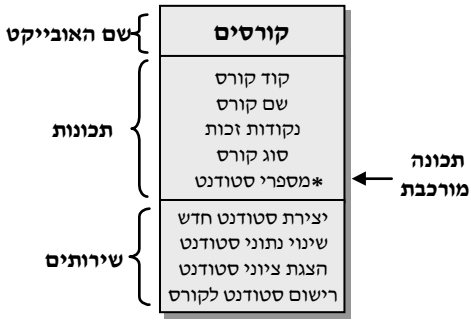
נתבונן ביישום הפועל בסביבה רגילה עם מערכת RDBMS ומנהל, בין השאר, את נתוני המרצים במכללה. כל תוכניות היישום פונות באמצעות פקודות SQL אל טבלת המרצים ולכן מבנה הטבלה חשוף ומוכר להן. שינוי במבנה הטבלה עלול לגרום לתופעות לוואי וצורך בביצוע שינויים במספר רב של תוכניות יישום. לעומת מצב זה, מודל האובייקטים מתייחס למרצה כאל אובייקט ולכן הנתונים והשירותים שלו ארוזים יחד. כל תוכניות היישום הפונות לאובייקט אינן מכירות את מבנה הנתונים של המרצה, אלא רק את השירותים שהוא מספק: הקמת מרצה חדש, שינוי נתוני מרצה, קידום בדרגה, סיום עבודה, הצגת ותק מרצה, מעבר לחלקיות משרה וכד'.

לתכונה זו של מודל האובייקטים קוראים **הסתרת מידע** (Information Hiding) והיא זו שמבטיחה רמת מודולריזציה גבוהה. אף אובייקט אינו יודע כיצד אובייקט אחר בנוי, לא מבחינת מבנה הנתונים שלו ולא מבחינת צורת יישום השירותים שלו. הוא מכיר רק את הממשק להפעלת השירותים של האובייקט. שינוי פנימי של האובייקט, מבחינת מבנה הנתונים ומבחינת צורת יישום השירות הינו שקוף מבחינת האובייקטים האחרים, כל עוד אין שינוי בממשק להפעלת השירות. לדוגמה, ניתן לשנות את האלגוריתם של חישוב ותק המרצה או לשנות את מבנה טבלת המרצים בלי שאף אובייקט אחר יהיה מודע לשינוי זה.

מחלקת אובייקטים (Object Class)

מחלקת אובייקטים, או מחלקה **Object Class**	***מחלקת אובייקטים*, או *מחלקה*, מוגדרת כאוסף של כל האובייקטים מאותו הסוג, שהם בעלי אותן תכונות ואותה התנהגות.**

כל אובייקט שייך למחלקה אחת בלבד. ניתן להתייחס למחלקה כאל **תבנית** (Template), המגדירה את התכונות של כל האובייקטים המשתייכים אליה. לדוגמה, אוסף כל האובייקטים מסוג קורס שייכים למחלקה קורסים.

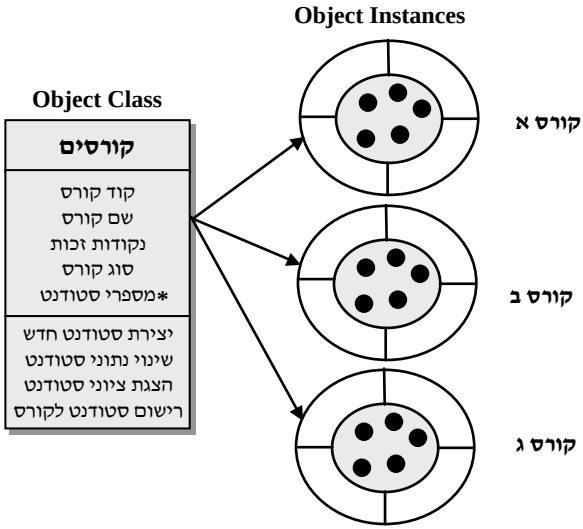


תרשים 18.5: מחלקת האובייקטים קורסים.

תרשים 18.5 מראה את שיטת הייצוג של מחלקת אובייקטים. לכל מחלקה יש לציין את שם המחלקה, התכונות של המחלקה ושמות כל השירותים (Methods) שהיא מספקת. בדוגמה שראינו, התכונה מספרי סטודנט היא מסוג Reference Attribute שמרחב הערכים שלה הוא המחלקה סטודנטים הרשומים לקורס.

יש להבחין בין **מחלקת האובייקטים** לבין **מופעי האובייקט** (Object Instance), שהם האובייקטים עצמם.

נחזור ונעיין בסוגיית מזהה האובייקט, OID. קיימות מספר חלופות למימוש מזהה זה: אחת מזהה את האובייקט באופן חד-ערכי בתוך המחלקה, והשנייה – מזהה את האובייקט בכלל, ללא תלות במחלקה שהוא שייך לה.

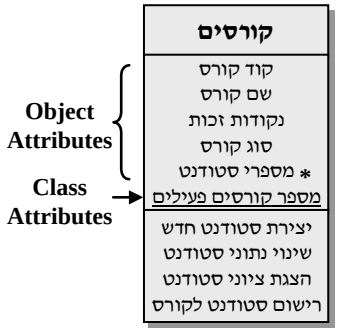


תרשים 18.6: מחלקת אובייקטים ואובייקטים.

- **זיהוי אובייקט בתוך מחלקה:** מכיון שכל אובייקט משתייך למחלקת אובייקטים מסוימת, ניתן ליישם את מזהה האובייקט על ידי שני ערכים: (Object Class Name, Object Id). המרכיב הראשון מזהה את מחלקת האובייקטים שאליה משתייך האובייקט, והמרכיב השני מזהה את המופע של אובייקט מסוים בתוך המחלקה. שיטת זיהוי זו מאפשרת למערכת המקבלת הודעה כלשהי לזהות מיידית את המחלקה שאליה מופנית ההודעה ולבצע מספר בדיקות ראשוניות (כמו למשל, האם הפרמטרים המופיעים בהודעה הם אכן פרמטרים חוקיים ובעלי טיפוס נתונים נכון, האם ההודעה מפעילה שירות חוקי ועוד). הבעיה העיקרית בשיטת זיהוי זו היא הקושי בהעברת אובייקט ממחלקה למחלקה. לדוגמה, העברת האובייקט סטודנט מהמחלקה סטודנטים למחלקה בוגרים תחייב את שינוי CID שלו.
- **זיהוי גלובלי של אובייקט:** ניתן לזהות אובייקט על ידי מספר חד-משמעי, שאינו כולל את שם המחלקה. שם המחלקה שהאובייקט משתייך אליה מוסתר כאחת התכונות של האובייקט עצמו. בשיטה זו, כל הודעה מחייבת קודם כל את שליפת האובייקט לצורך זיהוי המחלקה שהוא משתייך אליה, ואחר כך בדיקת חוקיות ההודעה. כמובן שתהליך זה פחות יעיל, אולם יתרונו בהעברה קלה יחסית של אובייקט בין מחלקות שונות.

בנוסף לניהול תכונות של כל אובייקט, מודל האובייקטים מאפשר לנהל גם את **תכונות המחלקה** (Class Attributes). תכונות אלו שייכות למחלקה ככלל, ולא לאובייקט מסוים המשתייך למחלקה. לדוגמה, ניתן להגדיר תכונה כגון מספר הקורסים הפעילים, כתכונה של המחלקה קורסים. תכונה זו תכיל את מונה מספר האובייקטים מסוג קורס הנמצאים ברגע נתון במחלקת הקורסים.

כך למשל, בכל פעם שמוסיפים קורס חדש למחלקה, השירות המתאים יגדיל את המונה ובכל פעם שמבטלים קורס, השירות המתאים יקטין מונה זה.

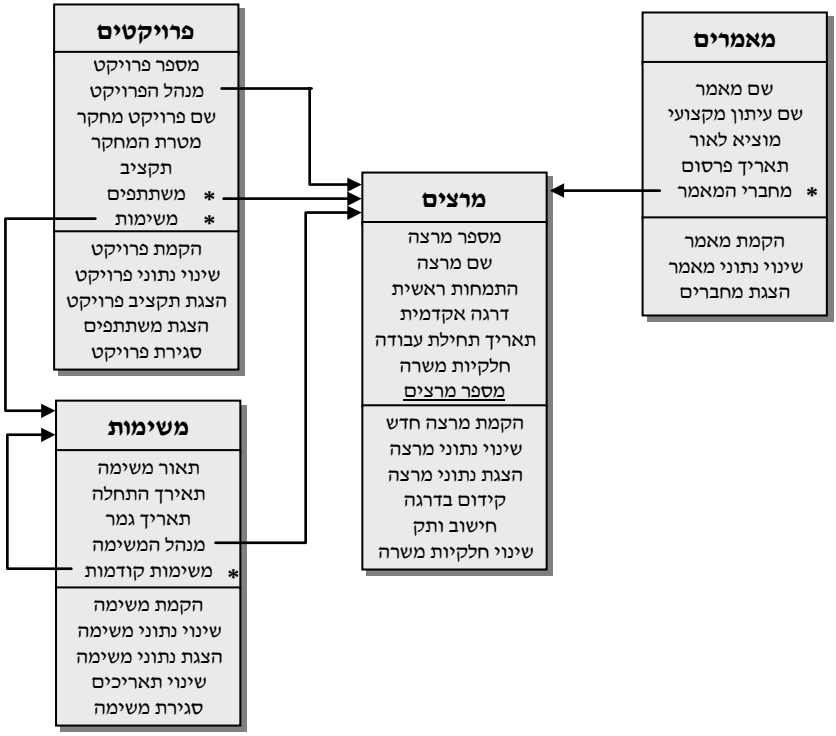


תרשים 18.7: מחלקת אובייקטים עם תכונות ברמת האובייקט וברמת המחלקה.

מודל האובייקטים מאפשר גם להגדיר שירותים ברמת המחלקה ולא רק ברמת האובייקט הבודד. למשל, הפקודה NEW שמייצרת אובייקט חדש היא ברמת המחלקה. מקובל לקרוא לשירותים ברמת המחלקה גם בשם **בנאים** (Constructors) אם הם בונים מופעים חדשים, או **מפרקים** (Destructors) אם הם מבטלים מופעים של אובייקטים.

הבה נבחן תרשים המציג סכימה של מספר אובייקטים: מרצה, מאמרים מקצועיים שמרצים השתתפו בכתיבתם, פרויקטי המחקר שבהם מרצים מעורבים ומשימות של כל פרויקט מחקר כזה.

נעיין בתרשים 18.8 ונשים לב שהקשרים בין האובייקטים מבוטאים על ידי תכונות מסוג **תכונת ייחוס** (Reference Attribute) המסומנים בכוכבית (*) ליד שם התכונה. במחלקה מרצים מוגדרת תכונה ברמת המחלקה, בנוסף לתכונות ברמת האובייקט. נשים לב לאובייקט משימות, המכיל מצביע אל מנהל המשימה ומצביע אל כל המשימות שמקדימות את המשימה המסוימת.



תרשים 18.8: מודל האובייקטים.

אובייקטים זמניים ואובייקטים קבועים

אחת הנקודות המעניינות, בעיקר מנקודת מבט של מערכות לניהול בסיסי נתונים, היא כיצד האובייקטים נשמרים לאורך זמן.

אובייקט חולף ואובייקט מתמיד **Transient and Persistent Objects**	***אובייקט חולף*** **(**Transient Object**) קיים רק בזמן ריצת התוכנית ובסיומה הוא נעלם.** ***אובייקט מתמיד*** **(** Persistent Object**) קיים תמיד ואינו תלוי בזמן הריצה של תוכנית היישום.**

כל **שפות התכנות מוכוונות האובייקטים** (Object Oriented Languages) מייצרות ומבטלות אובייקטים במהלך ריצה. אובייקטים אלה קיימים בזיכרון המחשב כל עוד התוכנית פועלת, ולכן מקובל לקרוא להם בשם אובייקטים חולפים. השאלה היא כיצד שומרים את האובייקטים החולפים מעבר לזמן הריצה של התוכנית, וכיצד הופכים אותם לאובייקטים מתמידים, או קבועים. הפיכת אובייקט חולף לאובייקט מתמיד מחייבת את שמירתו בקובץ או במערכת לניהול בסיסי נתונים.

קיימות מספר שיטות להפיכת אובייקט חולף לאובייקט מתמיד. שיטה אחת מניחה שהאובייקט הוא תמיד מתמיד ולכן, בעת יצירתו על ידי פקודה בשפת התכנות, כמו הפקודה NEW, הוא הופך לקבוע ונרשם מיידית במערכת ניהול בסיס הנתונים. שיטה אחרת היא להבחין בין שני סוגי האובייקטים – אובייקטים חולפים ואובייקטים מתמידים. הקמת אובייקט בזמן ריצה מגדיר אותו כחולף, וכדי להפוך אותו למתמיד דרושה פעולה מיוחדת. שיטה נוספת היא להגדיר את סוג מחלקת האובייקטים. אם מחלקת האובייקטים היא מסוג חולף כל אובייקט שיווצר בה יהיה מסוג זה, ואם מחלקת האובייקטים היא מסוג מתמיד כל אובייקט שיווצר בה הופך באופן אוטומטי לסוג זה.

נקודה נוספת שחשוב להדגיש היא שאובייקט חולף מוכר לתוכנית היישום שייצרה אותו אבל אינו נגיש על ידי תוכניות יישום אחרות. לעומת זאת, אובייקט מתמיד המאוחסן בבסיס נתונים כלשהו חייב להיות נגיש על ידי מספר רב של תוכניות יישום שונות. הבדל זה מחייב הרחבת רעיונות הלקוחים ממערכות ניהול בסיסי נתונים, כמו אינדקסים, נעילות, תנועות, התאוששות אל מערכות ODBMS.

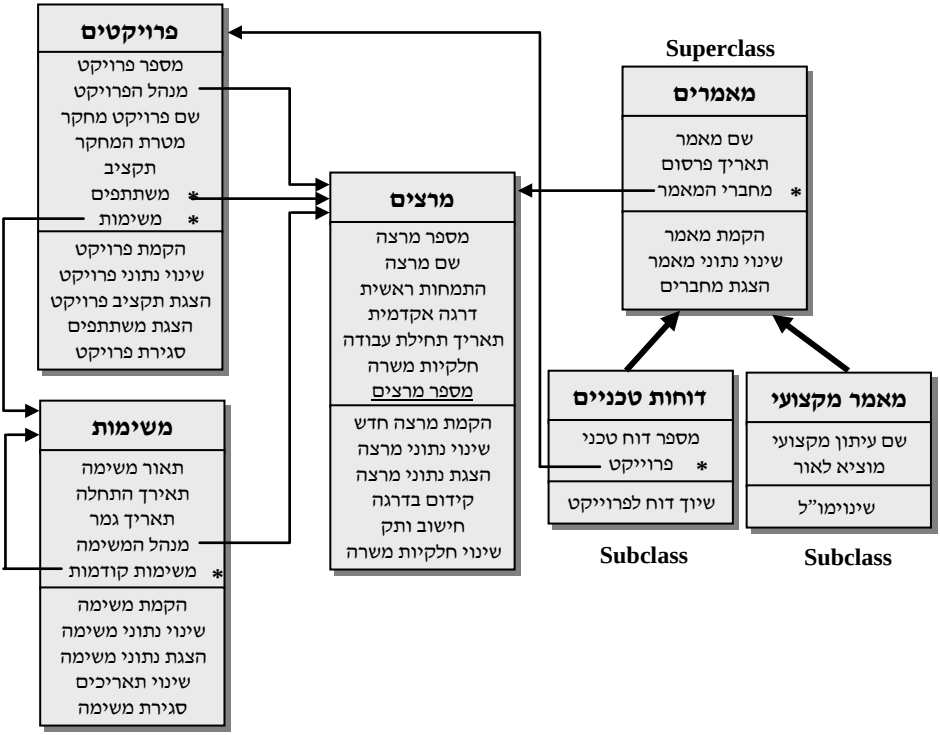
היררכיה של אובייקטים

אחד המאפיינים של מודל האובייקטים הוא שניתן ליצור **היררכיות של הכללה** (Generalization Hierarchies) בין מחלקות שונות של אובייקטים. המשמעות היא, שניתן לפצל מחלקת אובייקטים (Object Class) לתת-מחלקות (Sub Class), וכל תת-מחלקה לתת-מחלקות נוספות וכך הלאה. לדוגמה, ניתן לפצל את המחלקה מאמרים לשתי תת-מחלקות: דוחות טכניים ומאמרים מקצועיים. מודל האובייקטים תומך במושג **ההורשה** (Inheritance), ולפיו כל תת-מחלקה יורשת את **כל** התכונות והשירותים של כל המחלקות שנמצאות מעליה בהיררכיה.

על פי עיקרון ההורשה יורשות שתי תת-המחלקות, מאמר מקצועי ודוחות טכניים, את כל התכונות של המחלקה מאמרים. בנוסף לתכונות אלו, יש לכל תת-מחלקה מספר תכונות ייחודיות. כלל זה תקף גם לגבי השירותים (השיטות).

ככלל, ניתן לפצל מחלקה למספר בלתי מוגבל של תת-מחלקות. יש מערכות המאפשרות לתת-מחלקה להשתייך למחלקה אחת בלבד (Single Inheritance), ואחרות המרשות לתת-מחלקה להשתייך בו-זמנית למספר מחלקות (Multiple Inheritance). במקרה זה תת-המחלקה הנוצרת יורשת תכונות ופרוצדורות ממספר מחלקות שונות.

השימוש בהיררכיות הכללה של מחלקות מאפשר התפתחות הדרגתית של בסיס הנתונים. בכל נקודת זמן ניתן להוסיף תת-מחלקה חדשה למחלקה כלשהי, או לחילופין, להוסיף מחלקה מעל מחלקה קיימת. גמישות זו המשולבת עם עיקרון ההורשה הינם מאפיינים חשובים של מודל האובייקטים ושל מערכות ODBMS.



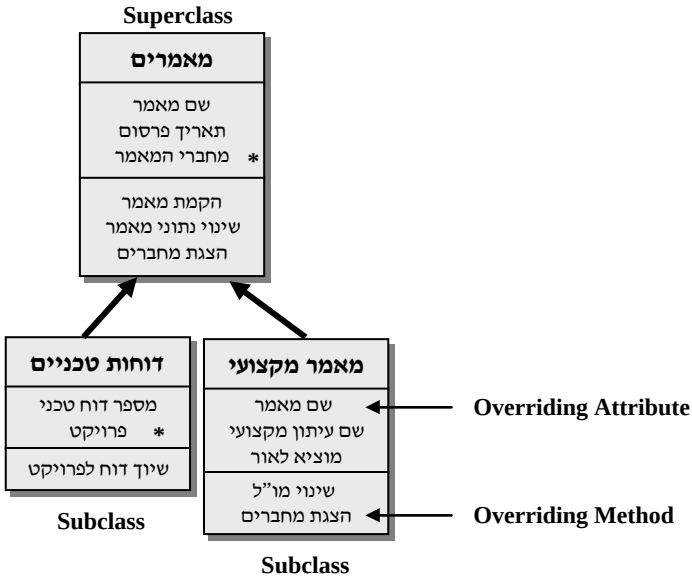
תרשים 18.9: מחלקה ותת-מחלקות.

ההורשה יכולה גם ליצור סתירות מסוימות. לדוגמה, כאשר בתת-מחלקה מופיעה תכונה בעלת שם זהה לשם תכונה במחלקת האם, לא מופעלת ההורשה. אם במחלקה יש שירות הפועל על תכונה, מה קורה כאשר השירות עובר בהורשה אולם התכונה לא?

היררכיית ההכללה בלתי תלויה (אורטוגונלית) לקשרים הנוצרים על ידי הגדרת תכונות ייחוס (Reference Attributes). באמצעות שני מנגנונים בלתי תלויים אלה ניתן לבנות בסיס נתונים מוכוון אובייקטים בכל רמת מורכבות נדרשת.

דריסה (Overriding)

תכונות ושירותים עוברים בירושה ממחלקת העל (Superclass) אל כל תת-המחלקות שלה. עם זאת, אם תכונה או שירות קיימים בתת-מחלקה, הם גוברים על ההגדרה שעוברת בירושה. לתהליך זה קוראים **דריסה** (Overriding). לדוגמה, נניח שבמחלקה מאמרים קיים שירות בשם הצגת נתוני מחברים וגם בתת-מחלקה מאמר מקצועי קיים שירות בשם דומה. לפיכך, הודעה שתופנה אל האובייקט מאמר מקצועי להפעלת השירות הצגת נתוני מחברים, תפעיל את היישום המופיע בתת-מחלקה, ולא את היישום המופיע במחלקת העל. עיקרון הדריסה חשוב – הוא מאפשר לבצע התאמות ברמות נמוכות של ההיררכיה, ולהגדיר תכונות או שירותים שיהיו בתוקף במקום אלה הנובעים מההורשה.



תרשים 18.10: דריסת תכונות ושירותים.

ריבוי צורות (Polymorphism)

הדריסה היא מקרה פרטי של תפישה כללית יותר הקרויה **ריבוי צורות** (Polymorphism) או **העמסת שירותים** (Method Overloading). אין בכך כל חידוש והדבר קיים בשפות תכנות רבות. לדוגמה, אם שני משתנים הם נומריים, אזי האופרטור '+' יבצע חיבור אלגברי ביניהם. לעומת זאת, אם שני המשתנים הם אלפאנומריים אזי האופרטור + יבצע שרשור (Concatenation) ולא חיבור אלגברי. כלומר, אותו אופרטור פועל בצורה שונה על פי סוג המשתנים.

נוכל להרחיב רעיון זה אל שירותים של אובייקטים. נוכל להגדיר שירותים בעלי ממשק זהה שישאו אותו שם ופרמטרים, אבל יהיה להם יישום שונה במחלקות שונות. לדוגמה, סביר להניח שלכל אובייקט יהיה שירות השולח את תוכנו למדפסת כלשהי. נתבונן בתוכנית יישום השולפת אובייקטים שונים ומדפיסה אותם. אם לכל אובייקט יש שירות הדפסה בעל שם שונה, תוכנית היישום תבצע קטע קוד כמו זה, למשל:

```
1.  for x in X do
2.     begin
3.        case of type (x);
4.              student:   print_student (x);
5.              teacher:  print_teacher (x);
6.              course:   print_course (x);
7.        end;
8.     end;
```

אם נרצה להוסיף הדפסה של אובייקט חדש, נצטרך לערוך שינוי בתוכנית היישום הזאת. במקום שיטה מסורבלת יחסית זו, ניתן לבנות שירות בעל אותו שם בכל האובייקטים, נניח שירות בשם display_data. היישום של שירות זה בכל מחלקת אובייקטים יהיה שונה אבל הממשק שלו זהה. עכשיו, במקום קטע קוד הקודם נוכל לרשום:

1. **for x in X do display_data (x);**

ההחלטה איזה שירות להפעיל תתבצע בזמן ריצה. אם האובייקט x הוא סטודנט יופעל היישום הדפסת נתוני סטודנט ואם x הוא מרצה, יופעל השירות הדפסת נתוני מרצה. מנגנון ריבוי צורות מחייב קישור דינמי (Dynamic Binding) בין שם השירות לבין היישום שלו, והדבר יכול להתבצע רק בזמן ריצה ולא בזמן הידור.

נשים לב שמנגנון ריבוי הצורות בלתי תלוי בהורשה. בתהליך ההורשה ניתן להשתמש בדריסה כדי להבטיח שהיישום המתאים יופעל, ואילו ריבוי הצורות מאפשר הגדרת יישום שירות נפרד שיופעל בלי כל קשר להורשה.

שפה לטיפול באובייקטים

בדומה לשפת טיפול בנתונים (DML) של מודל הנתונים הטבלאי, גם למודל האובייקטים יש להגדיר **שפת גישה**. שפה זו צריכה לאפשר למשתמשים לשלוף, להוסיף, לבטל ולעדכן את האובייקט על ידי משלוח ההודעה המתאימה למחלקה. מערכות ODBMS מאפשרות בדרך כלל שליפה של האובייקטים בשתי דרכים שונות:

- **ניווט בין האובייקטים בהתבסס על OID:** שיטה זו מאפשרת לשלוף כל אובייקט על פי הזיהוי שלו (OID) ללא התייחסות לתוכנו. שיטה זו מאפשרת גישה לאובייקטים מורכבים. לדוגמה, גישה לסכימה חשמלית וממנה לכל הרכיבים האלקטרוניים הכלולים בה.
- **שליפת אובייקטים על פי שאילתה:** גישה זו דומה לשפת SQL. השאילתה בוחרת מחלקת אובייקטים העונים על תנאי כלשהו. הגישה לאובייקטים שנשלפו יכולה להתבצע על ידי שליפה המבוססת על הזיהוי שלהם. שפת השאילתות צריכה לאפשר שליפת אובייקטים השייכים למחלקת אובייקטים מסוימת, או אובייקטים ממספר מחלקות אובייקטים שונות. שליפת אובייקטים ממספר מחלקות דומה בעיקרון לפעולת Join במודל הטבלאי.

המבנה של בסיס נתונים מוכוון אובייקטים מכיל היררכיות הכללה של מחלקות וקשרים בין מחלקות, ועל כן מורכב וקשה יותר לבנות רכיב אופטימיזציה הבוחר בתהליך הביצוע היעיל ביותר של השאילתה. עדיין לא התגבשה הסכמה כללית באשר למבנה שפת DML מוכוונת אובייקטים, ולכן לא נרחיב את הדיון בנושא זה. ארגון OMDG פרסם מסמך ובו הצעה לשפת שאילתות עבור אובייקטים בשם OQL (Object Query Language), שאמורה למלא תפקיד דומה לזה ששפת SQL ממלאת בעולם הטבלאי. הצעה זו עדיין לא אומצה ומומשה על ידי היצרנים של מערכות ODBMS.

מערכות ODBMS

הצורך במערכות לניהול בסיסי נתונים שתתמוכנה במיגוון גדול ומורכב של סוגי נתונים ויישומים, הביא למאמץ מחקרי גדול באוניברסיטאות ובחברות מסחריות. מאמץ זה התבסס במידה רבה על ההתפתחות המקבילה שהתרחשה בתחום הנדסת התוכנה ובמיוחד בתחום **שפות תכנות מוכוונות אובייקטים** (Object Oriented Programming Languages), שהן חדשות יחסית ונועדו לאפשר פיתוח מהיר ואמין יותר של יישומים מורכבים ותחזוקה נוחה יותר שלהם. בין שפות תכנות אלו ניתן למנות את C++, Java, Smalltalk, Lisp ואחרות.

נסיון החיבור בין שפות תכנות מוכוונות אובייקטים לבין בסיסי נתונים טבלאים לא עלה יפה בגלל השוני המהותי בין שתי התפישות. היה קושי בגישור על פני שתי התפישות ונוסף לכך הצורך ששפות תכנות מוכוונות אובייקטים תנהלנה אובייקטים מתמידים ביחידות אחסנה כמו דיסקים, ולא רק אובייקטים חולפים המנוהלים בזיכרון הפנימי. כל אלה הביאו לפיתוח זן חדש של מערכות לניהול בסיס נתונים : **מערכות בסיסי נתונים מוכוונות אובייקטים** – ODBMS (Object Oriented DBMS).

מערכות אלו נועדו בתחילה לתמוך בעיקר בשפות תכנות מוכוונות אובייקטים ולכן עוצבו ופותחו במיוחד לעבודה בסביבת תכנות זו. מפתחי המערכות האמינו שהם מתבססים על מודל עשיר ומתאים יותר ליישומים החדשים וביכולתם להחליף את המודל הטבלאי, בדיוק כפי שהמודל הטבלאי החליף את המודל הרשתי במהלך שנות ה-80. מטרתם המוצהרת של היצרנים היתה לבנות את הדור הבא בתחום המערכות לניהול בסיסי נתונים.

בין מערכות ODBMS המסחריות המובילות ניתן למנות את Versant של חברת Versant Technologies, Object Store של חברת Object Design, Objectivity של Objectivity Inc., Gemstone/Opal של ServioLogic, Matisse של Mattise SA, ONTOS של Ontologic ו-O2 של O2 Technology.

נכון לזמן כתיבת ספר זה, מערכות אלו לא זכו לתפוצה דומה לזו שהיתה נחלת המערכות הטבלאיות. למרות שהן קיימות כבר מספר שנים, התברר שהן אינן מסוגלות להחליף את המערכות הטבלאיות בכל הקשור ליישומי עיבוד נתונים ותנועות רגילים, אלא בעיקר להשלים אותן ולפעול לצידן. התברר שמערכות ODBMS מתאימות עבור פלחים מסוימים של יישומים (Niche Markets) – מערכות ניהול רשתות תקשורת, מערכות CAD וכד'.

הקורא המעוניין להרחיב ידיעותיו במערכות אלו מופנה לספרות של היצרנים ולספרים המפורטים ברשימה הביבליוגרפית [BM95], [CR96], [KW94], [LM95].

סקירת ההבדלים בין מערכות RDBMS לבין מערכות ODBMS

לפני שנציג את ההרחבות של המודל הטבלאי שרובן מבוססות על הטיוטה לתקן SQL3, נסכם בקצרה את ההבדלים העיקריים בין מערכות RDBMS המבוססות על תקן SQL2 לבין מערכות ODBMS:

- המודל הטבלאי מבוסס על התורה המתמטית של יחסים ויוצר את הקשרים בין טבלאות מנורמלות על ידי השוואת **ערכים** בעמודות שונות. מודל האובייקטים משתמש במצביעים מפורשים בשם **OID** (מזהי אובייקטים) ליצירת קשרים בין אובייקטים שונים. שיטה זו מאפשרת יצירת קשרים בין אובייקטים רבים ושונים (שרטוטים, מסמכים, גרפים וכד'), בלי צורך לוודא שיש לכולם תכונות ממרחב ערכים זהה.
- המודל הטבלאי מוגבל מבחינת הגדרת טיפוסי הנתונים. טיפוסי הנתונים יכולים להיות בסיסיים (Primitive) בלבד, כמו מספרים, תווים, תאריכים וכד'. במודל האובייקטים, מרחב טיפוסי הנתונים גמיש, בעל יכולת התרחבות ויכול להיות גם אובייקט אחר, והדבר מאפשר ליצור מבנים מורכבים.
- המודל הטבלאי מתייחס לבסיס נתונים כאל אוסף של טבלאות עם קשרים ביניהן. מודל האובייקטים מתייחס לבסיס הנתונים כאל אוסף של אובייקטים עם קשרים ביניהם. מודל האובייקטים מבחין בין קשרים רגילים (בין אובייקטים) לבין **היררכיות** (בין מחלקות על ותת-מחלקות).
- המודל הטבלאי אינו מאפשר למשתמש סריקה או ניווט בבסיס הנתונים, אלא דרך שפת השאילתות בלבד. הניווט נקבע אוטומטית על ידי המערכת והמשתמש אינו שולט על הפרוצדורה המבצעת את הסריקה. במודל האובייקטים יש אפשרות לבצע סריקה הנשלטת על ידי המשתמש בהתבסס על OID. במובן הזה, מזכיר מודל האובייקטים יותר את המודל הרשתי וההיררכי מאשר את המודל הטבלאי.
- במודל הטבלאי הסמנטיקה של הטבלה נמצאת בתוכניות היישום השונות. המודל עצמו מספק את ארבע פעולות היסוד בלבד (הוספה, קריאה, ביטול, עדכון), ואינו תומך בפעולות בעלות משמעות כגון רישום לקורס או עדכון ציוני בחינה. במודל האובייקטים, הסמנטיקה של התנהגות האובייקט מנוהלת יחד עם האובייקט עצמו; כלומר, האובייקט מתאר את עצמו (Self Describing) על ידי אוסף השירותים שהוא מכיר ומיגוון ההודעות שהוא מגיב להם. מודל האובייקטים תומך בהסתרת מידע, כך שמפתח היישום מקבל רק את המידע הדרוש לטיפול באובייקטים. דבר זה מאפשר גמישות רבה יותר בהגדרת המבנים הפיסיים של האובייקט והסתרת המודל בתוך בסיס הנתונים.
- מודל האובייקטים תומך בהורשה בין מחלקות שונות המשתתפות בהיררכיה כלשהי. ההורשה מתייחסת הן לתכונות של האובייקט והן להתנהגות שלו, כלומר הפונקציות שהוא מבצע. המודל הטבלאי אינו מכיר בעיקרון זה.

מערכות Object-Relational DBMS

במקביל להופעת מערכות ODBMS, חיפשו גם יצרני מערכות RDBMS אחר דרכים להתמודד עם מגבלות המודל הטבלאי ולהעשיר אותו ברעיונות ותכונות שנלקחו ממודל האובייקטים. יצרני מערכות RDBMS הן חברות גדולות בעלות מחזורי מכירות של מיליארדי דולרים ובאופן טבעי לא מתכוונות לתת לשוק שלהן לגווע ולאפשר לספקים אחרים "לאכול להם את הארוחה". חברות אלו משקיעות סכומי כסף גדולים בשיפור והעשרת המערכות שלהן, כך שתהיינה מתאימות גם עבור היישומים החדשים.

המשך המחקר בתחום המערכות הטבלאיות והתפוצה הרבה לה זכו שפות התכנות מוכוונות האובייקטים, הביאו לפיתוח המודל הטבלאי המשופר, **ERDM** (Extended Relational Data Model). במאמץ זה השתתף גם Codd, אבי המודל הטבלאי, שפרסם בשנת 1979 מאמר ובו מודל שהוא קרא לו בשם RM/T (T מציין Tasmania, כי המאמר הוצג בכנס של הארגון האוסטרלי למדעי המחשב שנערך בטסמניה) ובשנת 1990 הוא פרסם ספר בשם The Relational Model for Database Management / Version 2 ובו הציג מודל בשם RM/V2 (Relational Model / Version 2) [CE90].

במקביל הוקם פרויקט מחקרי בשם Postgres באוניברסיטת קליפורניה שבברקלי בראשות Dr. Stonebraker, שהינו אחד החוקרים המובילים בתחום המערכות הטבלאיות. הוא הגה ופיתח את מערכת Ingres ואת שפת QUEL שבמשך תקופה מסוימת התחרתה מול שפת SQL על ההגמוניה בשפות השאילתות. פרויקט Postgres היווה מקור לרעיונות וזרז לשיפור המודל הטבלאי על ידי שילוב רעיונות הלקוחים ממודל האובייקטים. Stonebraker הקים את חברת Illustra שפיתחה מערכת ניהול בסיסי נתונים חדשה ששילבה רעיונות של מודל האובייקטים עם המודל הטבלאי. החברה נרכשה בהמשך על ידי חברת Informix ומוצריה שולבו בקו המוצרים של חברת Informix. Stonebraker קרא לדור חדש זה של מערכות בשם **מערכות אובייקטים**, או **טבלאות לניהול בסיסי נתונים** – **ORDBMS** (Object-Relational DBMS).

גם תקן SQL3, שנכון להיום פורסמה רק טיוטה שלו, מכיל מספר הרחבות לכיוון ניהול אובייקטים. חלק גדול מהרעיונות המופיעים בטיוטה מבוססים על הרעיונות שפותחו במסגרת פרויקט Postgres. למרות שהתקן עדיין לא פורסם ואושר, החלו יצרנים שונים לשלב בתוך המערכות שלהם כמה מהרעיונות והמושגים שבטיוטה. הבעיה היא כמובן, שעם פרסום הגירסה הסופית של התקן והשינויים שיחולו בו עד אז, ייווצרו עובדות בשטח וההתכנסות של מערכות אלו סביב התקן תיקח זמן רב.

כיום, רוב יצרני מערכות RDBMS המובילות הוסיפו מספר תכונות חשובות למערכות שלהם וביניהם נציין את יבמ עם DB/2 Universal Database, חברת אינפורמיקס עם Informix Universal Server וחברת אורקל עם המוצר Oracle 8. למרות שמערכות אלו אינן מספקות את כל התכונות של מודל האובייקטים שמערכות ODBMS מספקות, הן משמשות מספר רב של יישומים חדשים, הן נהנות מגב כלכלי אדיר ונתמכות על ידי אוסף עצום של כלי תוכנה הפועלים בשפת SQL.

חלק מהרעיונות והדוגמאות המוצגים בהמשך לקוחים מספרו המצוין של Stonebraker [SM96] ומתוך הטיוטה של תקן SQL3. עדיין לא הוגדר תקן מחייב ולכן רעיונות וגישות ממומשים בצורות שונות על ידי יצרנים שונים ועלולים להשתנות עם הזמן. הצגת הדוגמאות בהמשך נועדה בעיקר להצביע על כיוונים עתידיים של מערכות RDBMS ושפת SQL ואין כוונה להציג מבנה מדויק של פקודות, שעדיין נמצא בתהליך התגבשות.

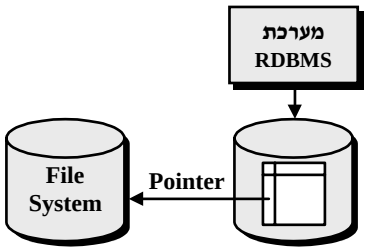
טיפוס הנתונים BLOB (Binary Large Object)

כפי שכבר אמרנו, מערכת RDBMS רגילה מספקת מספר מצומצם מאוד של טיפוסי נתונים היכולים לשמש להגדרת עמודות של טבלה (מספרים שלמים, מספרים עשרוניים, מחרוזת תווים, מחרוזת סיביות וכד'). כדי להתמודד עם אתגרי ניהול הנתונים עבור יישומים חדשים נדרשת תמיכה במיגוון הרבה יותר גדול של טיפוסי נתונים.

כמענה ראשוני להרחבת טיפוסי הנתונים הנתמכים, הוסיפו כמעט כל יצרני מערכות RDBMS אפשרות תמיכה ב**אובייקטים בינאריים גדולים** – **BLOB** (Binary Large Object). הם הוסיפו את טיפוס הנתונים BLOB ושאר טיפוסי הנתונים הנתמכים. נתון מסוג BLOB הוא למעשה רצף גדול של סיביות, חסרות כל משמעות מבחינת מערכת RDBMS. לא ניתן לנהל אובייקטים כאלה בתור טיפוס נתונים רגיל, כמו למשל VARCHAR או BIT בגלל מגבלת הגודל שלהם.

להבדיל משאר הנתונים המנוהלים באופן ישיר בעמודות של הטבלאות בבסיס הנתונים, עמודה שהיא מטיפוס נתונים BLOB אינה מכילה את האובייקט הבינארי עצמו אלא רק הצבעה לקובץ חיצוני המכיל אותו. הסיבה לכך היא כמובן שאובייקט בינארי כזה, המוצג בתרשים 18.11 ומכיל למשל תמונה, טקסט, וידאו, הקלטת קול, יכול להיות גדול מאוד וניהולו הישיר בתוך הטבלה יכול לפגום קשות בביצועים, בתהליכי הגיבוי והשחזור וכד'.

הגדרת עמודה בעלת טיפוס נתונים BLOB מאפשרת לנהל בתוך שורה בטבלה מצביע אל אובייקט בינארי כגון תמונה, מסמך Word, קובץ גיליון אלקטרוני Excel, סרט וידאו בפורמט MPEG וכד'. מערכת RDBMS אינה מכירה את האובייקט הזה ולכן אינה מסוגלת לבצע עליו פעולות כלשהן ואינה מאפשרת התייחסות אל תכונותיו בתוך שפת SQL. הדבר היחידי שהמערכת מאפשרת היא להציג את האובייקט הקשור לשורה שנשלפה בשאילתה כלשהי. כדי להציג את האובייקט דרושה התוכנה שבנתה אותו, או תוכנת תצוגה (Viewer) מתאימה.



תרשים 18.11: ניהול אובייקט בינארי מחוץ לבסיס הנתונים.

מאחר והאובייקטים הבינאריים מנוהלים במערכת הקבצים (File System) ומחוץ לשליטת מערכת RDBMS, לא ניתן להגן עליהם ולא ניתן לשתף אותם בתנועות ולבצע עליהם פעולות כלשהן. למרות מגבלות אלו, היכולת של מערכת RDBMS לנהל את המצביעים אל אובייקטים בינאריים חיצוניים מיוחדים אלה איפשרה לפתח יישומי SQL רגילים והרחבתם להצגת נתונים בלתי שגרתיים. הוספת התמיכה הזו חייבה שינויים מסוימים בשפת SQL ובעיקר ברכיבי Network SQL. ביישום שרת/לקוח למשל, אין כל הגיון להעביר ממחשב השרת אל מחשב הלקוח את האובייקט הבינארי בשעה שמעבירים את טבלת התוצאה בגלל העומס הרב שהדבר יכול להטיל על רשת התקשורת. אובייקטים מסוג זה ישוגרו אל מחשב הלקוח רק על פי בקשה מפורשת של היישום.

דוגמה : נוסיף לטבלת סטודנטים גם את תמונת הסטודנט.

```
1.  CREATE TABLE STUDENTS
2.          (STUDENT_ID   CHAR (5),
3.           NAME         CHAR (25),
4.           CITY         CHAR (20),
5.           PICTURE      BLOB)
```

בשורה 5 מוצגת עמודה בעלת טיפוס נתונים BLOB המכילה את תמונת הסטודנט. הרחבה זו מאפשרת פיתוח יישום השולף את השורות של כל הסטודנטים שגרים בעיר מסוימת והצגת התמונות שלהם. בצורה דומה ניתן לסרוק את תעודת הבגרות של סטודנט ולהוסיף עמודה מטיפוס BLOB שתצביע אל הקובץ הנוצר מסריקת התעודה.

נדגיש שהתמיכה בטיפוס נתונים BLOB עדיין אינה הופכת מערכת RDBMS למערכת ORDBMS. זהו צעד ראשון בהעשרת סוגי הנתונים שניתן לנהל במערכת RDBMS.

לדוגמה, במערכת DB2 Universal Database של יבמ ניתן להשתמש בטיפוס נתונים חדש בשם Datalink. עמודה זו בטבלה תכיל מצביע אל אובייקט המאוחסן **מחוץ** למערכת RDBMS. מצביע זה עשוי להכיל גם כתובת אינטרנט, URL (Uniform Resource Locator). המערכת מתבססת על ההרחבה של DB2 עם הרכיב DB2 File Manager שמאפשר להתייחס לאובייקטים המאוחסנים מחוץ לבסיס הנתונים כאילו היו חלק ממנו. למשל, בעת גיבוי בסיס הנתונים באמצעות תוכניות השירות הרגילות של DB2 ניתן להשתמש בפקודות GRANT ו-REVOKE כדי להגדיר זכויות גישה לאובייקטים אלה.

טיפוסי נתונים מופשטים (Abstract Data Types)

עד כה ראינו שטיפוסי הנתונים שניתן להשתמש בהם במערכת RDBMS מוגבלים ומוגדרים מראש על ידי יצרן המערכת. אלה הם **טיפוסי נתונים בסיסיים** (Base Data Type). יישום יכול להשתמש רק בטיפוסי נתונים בסיסיים ואינו יכול להוסיף טיפוסי נתונים חדשים בטבלאות של בסיס הנתונים. מערכת ORDBMS המבקשת לתמוך במיגוון יישומים מתוחכמים, חייבת לאפשר את הרחבת טיפוסי הנתונים הנתמכים ותמיכה בהגדרת

טיפוסי נתונים חדשים. טיפוסי הנתונים החדשים האלה נקראים **טיפוסי נתונים מופשטים** – **ADT** (Abstract Data Types), והם עשויים להיות פשוטים או מורכבים.

הרחבת טיפוסי הנתונים הבסיסיים

נתחיל בהגדרת טיפוס נתונים מופשט חדש וכך נרחיב את אוסף טיפוסי הנתונים הבסיסיים (Base Data Type Extension) העומדים לרשות היישום.

תחביר הפקודה להגדרת טיפוס נתונים מופשט:

```
1. CREATE TYPE Type_Name
2.     (InternalLength = Internal Length in Bytes,
3.      InputProc = Input Procedure Name,
4.      OutputProc = Output Procedure Name,
5.      Default = Default Value)
```

הגדרת טיפוס נתונים מופשט חדש כוללת את המרכיבים האלה:

- שם טיפוס הנתונים החדש.
- האורך של טיפוס הנתונים בבתים.
- שם הפרוצדורה הממירה את הנתונים ממבנה של ASCII לצורת ייצוג פנימית, לצורך קליטת הנתון.
- שם הפרוצדורה הממירה את צורת הייצוג הפנימית למבנה ASCII, לצורך הצגת הערכים על מסך או מדפסת.
- ברירת מחדל, במידה ויש.

דוגמה א': נגדיר טיפוס נתונים חדש בשם location שתופס שמונה בתים בדיסק ויכיל צמד מספרים המייצגים את נקודות הציון (הקואורדינטות) של נקודת יישוב כלשהי. ההמרה של צמד המספרים המופרדים ביניהם בפסיק, ממבנה ASCII למבנה הפנימי, יבוצע על ידי הפרוצדורה new_type_input וההמרה החוזרת מהמבנה הפנימי לצמד מספרים במבנה ASCII המופרדים ביניהם בפסיק יבוצע על ידי הפרוצדורה new_type_output.

```
1. CREATE TYPE LOCATION
2.     (InternalLength = 8,
3.      InputProc = new_type_input,
4.      OutputProc = new_type_output,
5.      Default = '0')
```

ברגע שטיפוס הנתונים החדש הוגדר, ניתן להתחיל להשתמש בו בעמודה בטבלה. בדוגמה הבאה מופיעה בשורה 5 העמודה מיקום העיר, המכילה את הקואורדינטות של עיר המגורים של הסטודנט.

```
1. CREATE TABLE STUDENTS
2.          (STUDENT_ID      CHAR (5),
3.           NAME            CHAR (25),
4.           CITY            CHAR (20),
5.           CITY_LOCATION   LOCATION)
```

בנוסף להגדרת טיפוסי נתונים חדשים, מערכת ORDBMS צריכה לאפשר גם הגדרת פונקציות (שירותים) המסוגלות לטפל בטיפוס הנתונים החדש. הפונקציה יכולה להכתב בשפת תכנות כלשהי כגון C++, Java, Smalltalk, Visual Basic וכד'.

תחביר הפקודה להגדרת הפונקציות של טיפוס נתונים מופשט :

להלן המבנה הכללי להגדרת פונקציה חיצונית, הכתובה בשפת תכנות כלשהי ומבצעת פעולה כלשהי על טיפוס הנתונים המופשט החדש :

```
1. DECLARE EXTERNAL FUNCTION function_name (type_name_1, …type_name_n)
2.     RETURNS  type_name
3.     AS file_name
4.     LANGUAGE language_name;
```

הסבר:

- שורה 1 מגדירה את שם הפונקציה ואת הפרמטרים שהיא מקבלת כקלט. לכל פרמטר יש להגדיר את טיפוס הנתונים.
- שורה 2 מגדירה את טיפוס הנתונים של התוצאה המוחזרת על ידי הפונקציה.
- שורה 3 מגדירה את שם הקובץ המכיל את הפונקציה.
- שורה 4 מגדירה את שם שפת התכנות שבה כתובה הלוגיקה הנמצאת בקובץ.

דוגמה ב' : נגדיר פונקציה שמחשבת את המרחק בין עיר המגורים של הסטודנט לבין המכללה.

```
1. DECLARE EXTERNAL FUNCTION  DistanceFromCollege  (location)
2.     RETURNS float
3.     AS 'dist_comp'
4.     LANGUAGE C;
```

הפונקציה DistanceFromCollege מקבלת כקלט משתנה אחד מטיפוס נתונים מיקום, מפעילה פרוצדורה בשפת C הנמצאת בקובץ dist_comp ומחזירה מספר עשרוני המייצג את המרחק בקילומטרים מהמכללה. כמובן שהפרוצדורה מכילה במשתנה פנימי את נקודות הציון של מיקום קמפוס המכללה ולכן מסוגלת לחשב בקלות את המרחק.

לאחר שהפונקציה החדשה הוגדרה ונרשמה, ניתן להתחיל להשתמש בה בשאילתות SQL. לדוגמה, אם נרצה לשלוף את כל הסטודנטים הגרים במרחק שמעל 25 קילומטר מהמכללה, נוכל לרשום את השאילתה הזו:

```
1. SELECT NAME
2. FROM STUDENTS  S
3. WHERE DistanceFromCollege  (CITY_LOCATION) > 25
```

מערכת ORDBMS תשלוף את שורות הסטודנטים, תפעיל את הפונקציה DistanceFromCollege לאחר שתעביר לה כפרמטר את המיקום של עיר המגורים. היא תקבל מהפונקציה את המרחק מהמכללה, ותשלוף רק את אלה המקיימים את התנאי המבוקש.

דוגמה ג': נבנה פונקציה DistanceAtoB, המחשבת את המרחק בין שתי נקודות.

```
1. DECLARE EXTERNAL FUNCTION  DistanceAtoB  (location, location)
2.     RETURNS float
3.     AS 'dist_a_to_b'
4.     LANGUAGE C;
```

נבנה עכשיו את השאילתה המציגה את כל הסטודנטים הגרים במרחק של פחות מקילומטר זה מזה וגרים במרחק של פחות מ-10 ק"מ מהמכללה.

```
1 .  SELECT NAME
2 .  FROM STUDENTS  S, STUDENTS  T
3 .  WHERE DistanceFromCollege   (CITY_LOCATION) <  10    AND
4 .                DistanceAtoB  (S.CITY_LOCATION, T.CITY_LOCATION) < 1
```

כפי שניתן לראות מתוך דוגמאות אלו, היכולת להוסיף טיפוס נתונים חדש והיכולת להגדיר פונקציות חדשות ולשלב אותן בתוך שאילתות SQL מאפשרת פיתוח יישומים מתוחכמים בצורה נוחה יחסית.

אם לא ניתן היה להרחיב את טיפוסי הנתונים, היינו צרכים לממש את המיקום כשתי עמודות נפרדות בטבלת הסטודנטים. כתיבת היישום היתה הופכת למורכבת הרבה יותר כי היינו צריכים לכתוב תוכנית יישום השולפת שורה של סטודנט, בודקת אם הוא גר במרחק מתאים מהמכללה, ואחר כך לשלוף את כל יתר הסטודנטים כדי לבדוק באיזה מרחק הם גרים זה מזה.

לדוגמה, מערכת Informix Universal Server כוללת מנגנון בשם DataBlade המאפשר להגדיר טיפוסי נתונים חדשים ואת הפונקציות לטיפול בהם. קיימים DataBlades המכילים מיגוון רחב של טיפוסי נתונים ופונקציות עבור יישומים מרחביים דו-מימדיים, תלת-מימדיים, עיבוד טקסט, עיבוד תמונות וכד'. לדוגמה DataBlade המטפל בטיפוסי נתונים עבור יישומים דו-מימדיים כולל טיפוסי נתונים כגון point, line, polygon, circle, ellipse ופונקציות כגון distance (point, point), distance (point, polygon), contained (point, rectangle), overlaps (polygon, polygon) ועוד. DataBlade המטפל בתמונות מסוגל

לקלוט תמונות בפורמטים שונים (JPEG, tiff, gif וכד') ופונקציות כגון rotate (image, angel), enhance (image) ועוד רבות אחרות. גם מערכות RDBMS אחרות כגון Oracle 8I ו-DB2 מכילות הרחבות המיועדות לטיפול במולטימדיה.

טיפוס נתונים מורכב (Composite Data Type)

נעבור עכשיו להגדרת טיפוסי נתונים מורכבים, שבנויים מטיפוסי נתונים בסיסיים וטיפוסי נתונים מופשטים גם יחד. תקן SQL3 תומך בהגדרת טיפוס נתונים מופשט מורכב, כלומר טיפוס נתונים הכולל נתונים שונים ופונקציות. זוהי הצורה בה תקן SQL3 מאפשר להגדיר מחלקת האובייקטים (Object Class).

תקן SQL3 מניח שלכל מופע של אובייקט, כלומר לשורה בטבלה המכילה את הנתונים, יש מזהה שורה חד-ערכי ברמת המערכת, שאינו ניתן לשינוי או לניצול מחדש. זהו מזהה האובייקט (OID) ממודל האובייקטים. ניתן לממש OID על ידי משתנה בינארי באורך כלשהו, למשל 64 סיביות, כאשר חלק מהסיביות מיועד לזיהוי הטבלה וחלקן מיועד לזיהוי השורה בתוך הטבלה. מזהה זה אינו מיועד לשימוש תוכניות היישום אלא רק לשימושה של מערכת RDBMS עצמה. התקן מאפשר להגדיר טיפוס נתונים מופשט ללא OID על ידי שימוש בפרמטר WITHOUT OID בזמן ההגדרה.

תחביר הפקודה:

```
1 .  CREATE TYPE  Type_Name (
2 .   PRIVATE
3.       Attribute_Name_1 Data_Type_1,
4 .      …,
5.       Attribute_Name_n Data_Type_n,
6.   PUBLIC
7.       Attribute_Name_m Data_Type_m,
8 .      …,
9.       Attribute_Name_z Data_Type_z,
10 .  EQUALS….,
11 .  LESS THAN….,
12 .  CONSTRUCTOR     FUNCTION
13.      RETURNS      XX
14.      BEGIN
15.         Code for object construction;
16.      END,
17.   DESTRUCTOR     FUNCTION
18.      RETURNS      XX
19.      BEGIN
20.         Code for object deletion;
21.      END,
22.   ACTOR     FUNTION
23.      RETURNS      XX
24.         Code for actor function;
25.      END
26 .  );
```

הסבר לתחביר הפקודה:

- שורה 1 מגדירה את שם טיפוס הנתונים החדש.
- שורות 2 עד 5 מגדירות אוסף של עמודות מוגנות (Private), שאינן חשופות באופן ישיר למשתמשים. עמודות אלו מנוהלות כחלק מהטבלה אבל נגישות רק באמצעות פונקציות פנימיות (Actor Functions). שיטה זו מאפשרת כמיסה (Encapsulation) של נתונים בתוך הטבלה, אשר יהיו מוכרים רק לאובייקט עצמו, אבל לא לעולם החיצון. לכל עמודה יש להגדיר את שם העמודה ואת טיפוס הנתונים שלה. עמודה יכולה להיות **אמיתית**, שהיא חלק מבסיס הנתונים, או **מדומה** (Virtual) שאינה קיימת במציאות אלא מחושבת כל פעם מחדש מתוך עמודות אחרות.
- שורות 6 עד 9 מגדירות את כל העמודות הרגילות של הטבלה, אשר חשופות לכל המשתמשים והיישומים והגישה אליהם היא באמצעות פקודות SQL רגילות. לכל עמודה יש להגדיר את שם העמודה ואת טיפוס הנתונים שלה. ניתן להגדיר לכל עמודה אם היא מיועדת לקריאה בלבד – ReadOnly; אם היא בת-עדכון – Updatable, או שהיא מכילה ערך קבוע – Constant. עמודה המוגדרת כ-Constant יכולה לקבל את הערך רק בעת יצירת האובייקט. טיפוס הנתונים של עמודה יכול להיות אחד מהטיפוסים הבסיסיים המסופקים על ידי מערכת RDBMS, טיפוס נתונים חדש או ADT אחר. עמודה יכולה להיות אמיתית כחלק מבסיס הנתונים, או מדומה (Virtual) שמחושבת כל פעם מחדש מתוך עמודות אחרות.
- שורות 10 ו-11 מאפשרות הגדרה של אופרטורי יחס בין אובייקטים – השוויון (Equals) וקטן מ- (Less Than). הגדרת שני יחסים אלה מאפשרת להגדיר את כל שאר היחסים כגון לא שווה, גדול מ-, לא גדול מ- וכד'. ניתן להגדיר שאופרטורים אלה יפעלו בצורה רגילה כברירת מחדל (Default), להגדיר שהם לא קיימים (None) ולכן לא ניתן לבדוק יחס בין אובייקטים, או לחילופין להגדיר כיצד לבצע את בדיקת היחס אובייקטים. הגדרת היחס בין אובייקטים נועד לאפשר הגדרה של אובייקטים מורכבים שבהם היחס אינו מובן מאליו, כמו למשל שוויון בין תמונות יכול להתבסס על זהות מלאה מבחינת הסיביות או רק דמיון מסוים.
- שורות 12 עד 16 מגדירות את הפונקציה שבונה שורה חדשה – **בנאי**. הפונקציה Constructor Function משמשת לבניית מופע חדש של האובייקט וקובעת את הערכים ההתחלתיים של העמודות. אם לא מגדירים פונקציה זו, הפקודה Insert תבנה אובייקט חדש בצורה רגילה.
- שורות 17 עד 21 מגדירות את הפונקציה שמבטלת שורה קיימת – **מפרק**. הפונקציה Destructor Function מבטלת מופע של האובייקט ומבצעת את פעולות ה"ניקוי" הנדרשות, כגון ביטול אובייקטים אחרים, שחרור משאבי מערכת, עדכון מצביעים וכד'. אם לא מגדירים פונקציה זו, הפקודה Delete תבצע ביטול רגיל של האובייקט.

- שורות 22 עד 25 מגדירות פונקציות פנימיות בשם Actor Functions, המבצעות פעולות שונות על האובייקט. ניתן להגדיר מספר בלתי מוגבל של פונקציות מסוג זה. זוהי הצורה בה מממש המודל הטבלאי המשופר את השירותים שאובייקט נותן על פי הגדרת מודל האובייקטים.

כפי שניתן לראות, הגדרת אובייקט חדש בעל טיפוס נתונים מופשט הינה מסובכת יותר מאשר הגדרת טבלה חדשה, אולם היא מאפשרת הגדרת אובייקטים מורכבים ומופשטים המותאמים לצרכים המיוחדים של היישומים. נשים לב שטיפוס הנתונים החדש מאפשר הגדרה של נתונים ופונקציות יחד, ולא רק נתונים כמו בטבלה רגילה.

טיפוסי נתונים שונים

דוגמה א' : נגדיר טיפוס נתונים חדש, סטודנט, המכיל את כל נתוני הסטודנט.

```
1.  CREATE TYPE STUDENT (
2.   PRIVATE
3.        BIRTH_DATE    DATE,
4.   PUBLIC
5.        STUDENT_ID        CHAR (5),
6.        NAME              CHAR (25) NOT NULL,
7.        CITY              CHAR (20),
8.        AGE               UPDATEABLE VIRTUAL
9.            GET WITH GET_AGE,
10.           SET WITH SET_AGE,
11.       CITY_LOCATION   LOCATION,
12.  ACTOR FUNCTION GET_AGE (S  STUDENT)
13.       RETURNS  INTEGER
14.       BEGIN
15.         Code for calculating age;
16.       END,
17.  ACTOR FUNCTION SET_AGE (S  STUDENT)
18.       RETURNS  INTEGER
19.       BEGIN
20.         Code for updating date of birth;
21.       END);
```

טיפוס הנתונים המורכב החדש עושה שימוש בנתונים בעלי טיפוס נתונים בסיסי או מופשט.

הסבר:

- שורה 3 מגדירה עמודה מוגנת המכילה את תאריך הלידה של הסטודנט. תאריך זה אינו מוצג בשאילתות ואינו בר עדכון באופן ישיר על ידי תוכניות היישום. הדרך היחידה לעדכן אותו היא על ידי הפונקציות המופיעות בהמשך.
- שורות 5 עד 11 מגדירות את עמודות הטבלה, אשר חשופות לתוכניות היישום.

- שורות 8 עד 10 מגדירות עמודה מדומה המכילה את גיל הסטודנט. הגיל מחושב על ידי הפונקציה get_age ומעודכן על ידי הפונקציה set_age. פונקציות אלו בלבד יכולות לגשת לתאריך הלידה ולחשב את גיל הסטודנט.
- שורה 11 מגדירה את העמודה מיקום עיר המגורים שהיא בעלת טיפוס נתונים מופשט ולא בסיסי.
- שורות 12 עד 21 מגדירות את הפונקציות לחישוב ולעדכון גיל הסטודנט.

בדוגמה זו לא הגדרנו את פונקציות הבנאי והמפרק. לכן, בעת ביצוע הפקודות Insert או Delete השורה תבנה או תימחק בצורה רגילה. הכללת טיפוסי נתונים מופשטים בתוך טיפוסי נתונים מופשטים מאפשרת הגדרה מקוננת (Nested) של עמודות בתוך טבלה, דבר שאינו נתמך על ידי תקן SQL2. כפי שנראה בדוגמה ה' בהמשך, אם רוצים להסתיר את המבנה הפנימי של האובייקט ניתן להשתמש בפונקציות מסוג Actor.

עכשיו ניתן להשתמש בהגדרה זו של טיפוס הנתונים להגדרת הטבלה סטודנטים. הצורך בהגדרת טבלה, גם במצב שבו טבלה מכילה רק טיפוס נתונים מופשט אחד, נועד בעיקר לשמור על התאימות עם גרסאות קודמות של מערכות RDBMS.

1. CREATE TABLE STUDENTS OF TYPE STUDENT

הגדרה זו בונה טבלה חדשה, טבלת סטודנטים, המורכבת משורות שטיפוס הנתונים שלהן הוא מסוג סטודנט.

דוגמה ב' : נגדיר טיפוס נתונים חדש מסוג כתובת מגורים המכיל את שם העיר, שם הרחוב, מספר הבית ומיקוד.

```
1. CREATE TYPE ADDRESS
2.          (CITY     CHAR (20),
3.           STREET   CHAR (25),
4.           NUM      INTEGER,
5.           ZIP      INTEGER)
```

ניתן להשתמש בטיפוס נתונים כתובת מגורים בזמן הגדרת טיפוס הנתונים מסוג סטודנט.

```
1.  CREATE TYPE STUDENT (
2.   PRIVATE
3.        BIRTH_DATE    DATE,
4.   PUBLIC
5.        STUDENT_ID            CHAR (5),
6.        NAME                  CHAR (25) NOT NULL,
7.        STUDENT_ADDRESS  ADDRESS,
8.        AGE                   UPDATEABLE VIRTUAL
9.            GET WITH GET_AGE,
10.           SET WITH SET_AGE,
11.       CITY_LOCATION         LOCATION,
12.  ACTOR FUNCTION GET_AGE (S  STUDENT)
13.       RETURNS  INTEGER
14.       BEGIN
15.          Code for calculating age;
16.       END,
17.  ACTOR FUNCTION SET_AGE (S  STUDENT)
18.       RETURNS  INTEGER
19.       BEGIN
20.          Code for updating date of birth;
21.       END);
```

נשים לב שבשורה 7 מופיע עכשיו טיפוס נתונים מופשט חדש, כתובת סטודנט, במקום עיר המגורים שהיתה בעלת טיפוס נתונים בסיסי. נגדיר את טבלת הסטודנטים:

```
1.  CREATE TABLE  STUDENTS OF TYPE STUDENT
```

שורה של סטודנט תהיה מורכבת משני נתונים בעלי טיפוס נתונים בסיסי, מספר הסטודנט ושם הסטודנט, נתון בעל טיפוס נתונים מורכב, כתובת הסטודנט ונתון בעל טיפוס נתונים חדש, מיקום העיר. כמו כן הוא יכיל עמודה מוגנת של תאריך לידה ועמודה מדומה של גיל הסטודנט.

כדי לאפשר פנייה אל מרכיב מסוים של ההגדרה המקוננת, תקן SQL3 משתמש בסימון של שתי נקודות (Double Dot Notation) כמו בשורה 3 בדוגמה ג' להלן.

דוגמה ג' : יש להציג את מספר הסטודנט, שם הסטודנט ואת המיקוד של כל הסטודנטים הגרים בחיפה.

```
1.  SELECT STUDENT_ID, NAME, STUDENT_ADDRESS..ZIP
2.  FROM STUDENTS
3.  WHERE STUDENT_ADDRESS..CITY = 'Haifa'
```

נשים לב שבשורות 1 ו-3 הפנייה אל מיקוד ואל עיר המגורים, המהווים חלק מטיפוס הנתונים המופשט המקונן כתובת מגורים, מחייבת שימוש בסימון המיוחד.

דוגמה ד' : נניח שבנינו פונקציה מיוחדת המחשבת את סכום הספרות של משתנה נומרי וקראנו לה בשם sum_digits. נשתמש בפונקציה זו כדי לשלוף את כל הסטודנטים הגרים בחיפה ושסכום הספרות של המיקוד שלהם הוא 50.

```
1.  SELECT STUDENT_ID, NAME, STUDENT_ADDRESS..ZIP
2.  FROM STUDENTS
3.  WHERE STUDENT_ADDRESS..CITY = 'Haifa' AND
4.            SUM_DIGITS (STUDENT_ADDRESS..ZIP) = 50
```

ניתן להשתמש בפונקציה זו גם כדי להציג את סכום הספרות של מספר הבית של כל הסטודנטים הגרים בחיפה ושסכום הספרות של המיקוד שלהם הוא 50.

```
1.  SELECT STUDENT_ID, NAME, SUM_DIGITS (STUDENT_ADDRESS..NUM)
2.  FROM STUDENTS
3.  WHERE STUDENT_ADDRESS..CITY = 'Haifa' AND
4.            SUM_DIGITS (STUDENT_ADDRESS..ZIP) = 50
```

דוגמה ה' : נראה כיצד ניתן לבנות פונקציה מסוג Actor שתהיה חלק מהגדרת טיפוס הנתונים המופשט וכיצד ניתן להשתמש בה בפקודות SQL. נבנה פונקציה המקבלת כפרמטר את הכתובת של הסטודנט ומחזירה את עיר המגורים.

```
1.  CREATE TYPE STUDENT (
2.   PRIVATE
3.        BIRTH_DATE    DATE,
4.   PUBLIC
5.        STUDENT_ID              CHAR (5),
6.        NAME                    CHAR (25) NOT NULL,
7.        STUDENT_ADDRESS  ADDRESS,
8.        AGE                     UPDATEABLE VIRTUAL
9.            GET WITH GET_AGE,
10.           SET WITH SET_AGE,
11.       CITY_LOCATION        LOCATION,
12.   ACTOR FUNCTION GET_AGE (S  STUDENT)
13.       RETURNS  INTEGER
14.       BEGIN
15.         Code for calculating age;
16.       END,
17.   ACTOR FUNCTION SET_AGE (S  STUDENT)
18.       RETURNS  INTEGER
19.        BEGIN
20.          Code for updating date of birth;
21.        END,
22.   ACTOR FUNCTION GET_CITY (AD ADDRESS),
23.       RETURNS CHAR (20)
24.       BEGIN
25.         SET C = AD..CITY;
26.         RETURN C;
27.       END);
```

הפונקציה החדשה Get_City מוצגת בשורות 22 עד 26. כפי שניתן לראות, באמצעות פונקציה זו ניתן לממש את עיקרון הכמיסה (Encapsulation) ולהסתיר את מבנה כתובת הסטודנט מתוכניות היישום. אם מסיבה כלשהי נצטרך לבצע שינוי בצורת הייצוג, כל שנצטרך לעשות הוא לשנות את הפונקציה ולא את תוכניות היישום.

עכשיו נוכל לבקש את עיר המגורים של כל הסטודנטים ששמם מתחיל באות A.

```
1. SELECT NAME, GET_CITY (STUDENT_ADDRESS)
2. FROM    STUDENTS
3. WHERE  NAME LIKE 'A%'
```

כמובן שנוכל להשתמש בפונקציה חדשה זו גם בתוך משפט WHERE. אם נרצה להציג את שמות הסטודנטים הגרים בחיפה נכתוב:

```
1. SELECT NAME,
2. FROM    STUDENTS
3. WHERE  GET_CITY (STUDENT_ADDRESS) = 'Haifa'
```

קשרים בין טבלאות על ידי טיפוס נתונים "מצביע" (Reference)

תקן SQL3 מאפשר יצירת קשר בין טבלאות, או בין שורות של אותה טבלה, על ידי שימוש בטיפוס נתונים מסוג חדש, **המצביע** (Reference). בתקן SQL2 הדרך היחידה לנהל קשרים בין טבלאות היא על ידי מפתחות זרים. כפי שאמרנו, יצירת קשר בין אובייקטים מורכבים אינו אפשרי תמיד על ידי מפתחות זרים ולכן נדרש מנגנון ליצירת קשר ישיר. התוכן של נתון בעל טיפוס נתונים מסוג מצביע הוא OID (מזהה אובייקט) של השורה המתאימה בטבלה אחרת או באותה טבלה.

דוגמה א' : נגדיר טיפוס נתונים חדש מפה, המכיל את התרשים של מפת עיר, את מספר התושבים בעיר ואת התאריך שבו נוסדה העיר.

```
1 . CREATE TYPE MAP
2 .          (MAP_ID              REF(MAP),
3 .           CITY_MAP            BLOB,
4.            POPULATION          INTEGER,
5 .           DATE_ESTABLISHED DATE )
```

נשים לב לעמודה המוגדרת בשורה 2, שהינה בעלת טיפוס נתונים מסוג Reference ומכילה את מזהה השורה. נגדיר את טבלת המפות, כלומר טבלה המכילה אוסף של מפות ערים.

```
1. CREATE TABLE  MAPS OF TYPE MAP
2.       VALUES FOR MAP_ID ARE SYSTEM GENERATED
```

שורה 2 מגדירה שמערכת RDBMS מחוללת בעצמה את הערכים של העמודה MAP_ID. נגדיר משתנה חדש בטבלת סטודנטים המצביע אל מפת העיר שבה מתגורר הסטודנט.

```
1.  CREATE TYPE STUDENT (
2.    PRIVATE
3.          BIRTH_DATE    DATE,
4.    PUBLIC
5.          STUDENT_ID              CHAR (5),
6.          NAME                    CHAR (25) NOT NULL,
7.          STUDENT_ADDRESS  ADDRESS,
8.          AGE                     UPDATEABLE VIRTUAL
9.              GET WITH GET_AGE,
10.             SET WITH SET_AGE,
11.         CITY_LOCATION         LOCATION,
12.         MAP_ID                REF (MAP),
13.   ACTOR FUNCTION GET_AGE (S  STUDENT)
14.         RETURNS  INTEGER
15.         BEGIN
16.           Code for calculating age;
17.         END,
18.   ACTOR FUNCTION SET_AGE (S  STUDENT)
19.         RETURNS  INTEGER
20.         BEGIN
21.           Code for updating date of birth;
22.         END);
```

העמודה MAP_ID בטבלת סטודנטים המוגדרת בשורה 12, תכיל את המזהה של השורה המתאימה בטבלה מפות.

כדי לפנות למשתנה בטבלת מפות, יש לבצע פעולה הנקראת **פיענוח המצביע** (Dereferencing). לשם כך מגדיר תקן SQL3 את האופרטור → (חץ ימינה) לסימון הפעולה. אם x הוא מצביע על שורה t כלשהי ו-a היא תכונה בתוך שורה t, אזי הדרך לפנות ל-a היא על ידי x → a. לאופרטור → יש ב-SQL3 אותה משמעות כמו בשפת C. ניתן גם לבצע את הפיענוח על ידי הפונקציה Deref, כך: deref(a).

נבהיר זאת על ידי הדוגמה של השאילתה המציגה את מספר התושבים בעיר המגורים של כל הסטודנטים המתגוררים במרחק גדול מ-25 ק״מ מהמכללה.

```
1.  SELECT STUDENT_ID, NAME, DEREF (MAP_ID).POPULATION
2.  FROM STUDENTS
3.  WHERE DistanceFromCollege  (CITY_LOCATION) > 25
```

נשים לב ששימוש שהפונקציה deref מחליפה את השימוש בפעולת Join בין שתי הטבלאות. הפונקציה deref משתמשת ב-OID שמאפשר תרגום מהיר למיקום פיסי של השורה בדיסק, ולכן פעולה זו יעילה יותר מאשר פעולת Join רגילה.

דוגמה ב׳: נשתמש בטיפוס נתונים מסוג מצביע כדי ליצור קשרים בין טבלאות במקום מנגנון המפתח הזר. נגדיר את הטבלאות סטודנטים, קורסים וציונים מחדש ונשתמש בטיפוס נתונים חדש זה. למען פשטות הדוגמה, מוצגת כאן הגדרה מקוצרת של כל טיפוסי הנתונים.

```
1.  CREATE TYPE STUDENT
2.              (STUDENT_ID            REF (STUDENT),
3.               STUDENT_NUMBER   CHAR (5),
4.               NAME                   CHAR (25),
5.               STUDENT_ADDRESS  ADDRESS,
6.               CITY_LOCATION       LOCATION);
7.
8.  CREATE TYPE COURSE
9.              (COURSE_ID             REF (COURSE),
10.              COURSE_NUMBER   CHAR (5),
11.              COURSE_NAME        CHAR (25));
12.
13. CREATE TYPE GRADE
14.             (COURSE_ID         REF(COURSE),
15.              STUDENT_ID        REF(STUDENT),
16.              FINAL_GRADE    INTEGER,
17.              SEMESTER          CHAR (10));
18.
19. CREATE TABLE STUDENTS OF TYPE STUDENT
20.    VALUES FOR STUDENT_ID ARE SYSTEM GENERATED;
21. CREATE TABLE CUM_LAUDE_STUDENTS OF TYPE STUDENT
22.    VALUES FOR STUDENT_ID ARE SYSTEM GENERATED;
23. CREATE TABLE COURSES OF TYPE COURSE
24.    VALUES FOR COURSE_ID ARE SYSTEM GENERATED;
25. CREATE TABLE GRADES OF TYPE GRADE
26.             SCOPE FOR COURSE_ID IS COURSES
27.             SCOPE FOR STUDENT_ID IS STUDENTS ;
```

הסבר:

- שורות 1 עד 6 מגדירות את נתוני הסטודנט על ידי טיפוס נתונים מסוג שורה. שורה 2 מגדירה את העמודה שתכיל את מזהה האובייקט (OID) של הטבלה.
- שורות 8 עד 11 מגדירות את נתוני הקורס על ידי טיפוס נתונים מסוג שורה. שורה 9 מגדירה את העמודה שתכיל את מזהה האובייקט של הטבלה.
- שורות 13 עד 17 מגדירות את נתוני הציון על ידי טיפוס נתונים מסוג שורה. בשורות 14 ו-15 השתמשנו בטיפוס נתונים מסוג מצביע ליצירת קשר רב-רב ערכי בין טבלת סטודנטים לטבלת קורסים.
- בשורות 19 עד 27 הגדרנו ארבע טבלאות המבוססות על שלושת טיפוסי הנתונים הקודמים. נשים לב שטיפוס הנתונים סטודנט משמש אותנו להגדרת שתי טבלאות שונות, סטודנטים וסטודנטים מצטיינים. מאחר ובתוך ההגדרה של טיפוס הנתונים מסוג מצביע מופיע השם של טיפוס הנתונים מסוג שורה ולא שם הטבלה שבה טיפוס נתונים זה מופיע, מגדיר תקן SQL3 את המשפט Scope. בשורה 27 מופיעה ההגדרה שעמודה STUDENT_ID שהיא בעלת טיפוס נתונים STUDENT מצביעה לטבלת STUDENTS ולא לטבלת CUM_LAUDE_STUDENTS.

נוכל עכשיו להגדיר את השאילתה המציגה את שמות ועיר המגורים של כל הסטודנטים שקיבלו ציון מעל 90 בסמסטר קיץ 1999 בדרך זו.

```
1.  SELECT DEREF (STUDENT_ID).STUDENT_NUMBER,
2.          DEREF(COURSE_ID).COURSE_NUMBER
3.  FROM    GRADES
4.  WHERE FINAL_GRADE > 90 AND
5.          SEMESTER = 'SUM1999'
```

נשים לב כיצד השימוש בטיפוס נתונים מסוג מצביע ובפונקציה deref מאפשר כתיבת שאילתה המבצעת Join בין שלוש טבלאות שונות. השאילתה מתבצעת על ידי סריקת השורות של טבלת ציונים ובדיקת קיום התנאי. אם התנאי מתקיים, השורה נשלפת ועל ידי שימוש במצביעים מוצגים שם הסטודנט ושם הקורס. ניתן לכתוב שאילתה זו גם בצורה רגילה יותר, תוך שימוש ב-Join.

```
1.  SELECT S.STUDENT_NUMBER, C.COURSE_NUMBER
2.  FROM    GRADES G, STUDENT S, COURSES S
3.  WHERE FINAL_GRADE > 90 AND
4.          SEMESTER = 'SUM1999' AND
5.          G.STUDENT_ID = S.STUDENT_ID AND
6.          G.COURSE_ID = C.COURSE_ID
```

דוגמה ג' : נראה דוגמה לעדכון עמודה המכילה מצביע. נניח שרוצים לעדכן בטבלה ציונים את הציון והסמסטר של סטודנט שמספר הסטודנט שלו הוא 115 ומספר הקורס M-110.

```
1.  UPDATE GRADES
2.  SET  GRADE = 95,
3.        SEMESTER = 'FALL1999'
4.  WHERE STUDENT_ID = (SELECT REF(STUDENT_ID)
5.                      FROM    STUDENTS
6.                      WHERE  STUDENT_NUMBER = '115')
7       AND COURSE_ID = (SELECT REF(COURSE_ID)
8.                      FROM    COURSES
9.                      WHERE  COURSE_NUMBER = 'M-110');
```

הסבר:

- שורות 2 ו-3 מעדכנות את העמודות המתאימות.
- שורות 4 עד 6 מגדירות מזהה אובייקט (OID) של שורת הסטודנט שמספרו 115.
- שורות 7 עד 9 מגדירות מזהה אובייקט של שורת הקורס שמספרו M-110.

נוכל לראות שהשתמשנו כאן בפונקציה REF המבצעת פעולה הפוכה מ-DEREF, כלומר זו פונקציה שמציגה את המצביע עצמו ולא את הפיענוח שלו.

פונקציות SQL

עד כה ראינו שניתן להוסיף פונקציות הכתובות בשפת תכנות כלשהי. מערכות ORDBMS תומכות גם בפונקציות פנימיות, פונקציות המשתמשות בהרחבות מסוימות של שפת SQL.

תחביר הפקודה:

```
1. DECLARE FUNCTION function_name (type_name_1, …type_name_n)
2.     RETURNS type_name
3.     AS SQL expression;
```

להבדיל מההגדרה של פונקציה חיצונית, כאן הפונקציה מוגדרת על ידי שימוש בביטוי SQL והרחבות מסוימות שניתן להוסיף.

דוגמה : כתוב פונקציה המחזירה את רשימת הסטודנטים אם נתונים הציון הסופי והסמסטר.

```
1. DECLARE FUNCTION  STUDENT_DATA  (integer, char)
2. RETURNS  STUDENT AS
3.         SELECT DEREF (STUDENT_ID)
4.         FROM    GRADES
5.         WHERE FINAL_GRADE = $1 AND
6.                 SEMESTER = $2;
```

הסבר:

- שורה 1 מגדירה את שם הפונקציה ואת טיפוס הנתונים של שני הפרמטרים שהיא מקבלת כקלט. הפרמטר הראשון חייב להיות מספר שלם כלשהו והפרמטר השני מחרוזת תווים כלשהי.
- שורה 2 מגדירה שהפונקציה מחזירה רשימה של נתונים בעלי טיפוס של סטודנט.
- שורות 3 עד 5 מגדירות את הפקודה SELECT שתתבצע עם הפעלת הפונקציה. נשים לב שבשורה 3 אנו משתמשים בפונקציה deref כדי לפענח את המצביע לטבלת סטודנטים. בשורה 5 ו-6 מופיעים שמות סימבוליים עבור הפרמטרים על פי סדר הופעתם.

נוכל עכשיו לכתוב שאילתה המציגה את רשימת הסטודנטים שקיבלו ציון 90 בסמסטר חורף 2000 בצורה הבאה:

```
1. SELECT STUDENT_DATA (90, 'WIN2000')
```

טיפוס נתונים מסוג "אוסף מצביעים" (SETOF)

טיפוס הנתונים Reference מכיל מצביע אחד בלבד. ניתן להגדיר טיפוס נתונים נוסף מסוג Setof המכיל מספר בלתי מוגבל של מצביעים. כל אחד מהמצביעים הוא במבנה של OID.

דוגמה : נגדיר עמודה בטבלת הסטודנטים המכילה את רשימת כל הקורסים אליהם הסטודנט נרשם.

```
1.  CREATE TYPE STUDENT
2.          (STUDENT_ID          REF (STUDENT),
3.           STUDENT_NUMBER      CHAR (5),
4.           NAME                CHAR (25),
5.           STUDENT_ADDRESS     ADDRESS,
6.           CITY_LOCATION       LOCATION,
7.           ENROLLMENT          SETOF (COURSE));
8.
9.  CREATE TABLE STUDENTS OF TYPE STUDENT
10.  VALUES FOR STUDENT_ID ARE SYSTEM GENERATED
11.  SCOPE FOR ENROLLMENT IS COURSES;
```

הסבר:

- בשורה 7 הגדרנו עמודה המכילה אוסף של מצביעים, כל אחד מצביע על אחד מהקורסים שהסטודנט נרשם אליו.
- בשורות 9 עד 11 הגדרנו את טבלת סטודנטים.
- שורה 10 מגדירה שהעמודה STUDENT_ID תכיל OID כפי שמערכת RDBMS מחוללת באופן אוטומטי.
- שורה 11 מגדירה שה-OID שבעמודה ENROLLMENT מצביע על שורות בטבלת קורסים.

נוכל עכשיו לכתוב שאילתה המציגה את שמות הקורסים אליהם רשומים סטודנטים ששמם מתחיל באות A. מאחר והפעם אנו מטפלים ברשימה של OID ולא ב-OID בודד, יש צורך לכתוב את השאילתה בצורה מיוחדת.

```
1.  SELECT DEREF (ENROLLMENT).COURSE_NAME
2.  FROM   (SELECT ENROLLMENT
3.              FROM STUDENTS
4.              WHERE STUDENT_NAME_LIKE   'A%' )
```

הסבר:

- שורה 1 מבצעת את הפיענוח של אוסף המצביעים לטבלת קורסים ומציגה רק את שם הקורס מתוך טבלה זו.
- שורות 2 עד 4 מגדירות את השאילתה הבונה את רשימת המצביעים.

אם היינו מבקשים להציג את כל העמודות של הקורסים אליהם נרשמו הסטודנטים, היינו כותבים:

```
1.  SELECT DEREF (*)
2.  FROM   (SELECT ENROLLMENT
3.              FROM STUDENTS
4.              WHERE STUDENT_NAME  LIKE 'A%')
```

טיפוס נתונים מסוג "מערך" (Array)

מערכת RDBMS אינה תומכת במערכים. כל עמודה חייבת לקבל ערך אחד בלבד. עבור יישומים מסוימים מגבלה זו גורמת לקושי וסרבול.

דוגמה : התקציב שכל מחלקה מקבלת הוא רבעוני ולכן יש לנהל את ארבעת התקציבים הרבעוניים בתוך שורת המחלקה. במקום להגדיר ארבע עמודות נפרדות או לפרק את התקציב לטבלה נפרדת, נגדיר עמודה בעלת טיפוס נתונים של מערך בגודל 4.

```
1.  CREATE TYPE DEPARTMENT
2.          (DEPART_ID        REF (STUDENT),
3.           DEPART_CODE      CHAR (2),
4            DEPART_NAME      CHAR (5),
5.           QTR_BUDGET       DECIMAL (8,2) [4]);
6.
7.  CREATE TABLE DEPARTMENTS OF TYPE DEPARTMENT
8.     VALUES FOR DEPART_ID ARE SYSTEM GENERATED;
```

בשורה 5 מופיעה עמודה שטיפוס הנתונים שלה הוא מספר עשרוני ומערך של 4 מופעים. ניתן עכשיו לפנות באופן ישיר לערך מסוים של המערך. לדוגמה, אם נרצה להציג את התקציב של הרבעון השני של עבור כל המחלקות בהן התקציב של הרבעון שני גדול מתקציב הרבעון הראשון, נכתוב :

```
1.  SELECT   DEPART_NAME, QTR_BUDGET [2]
2.  FROM     DEPARTMENTS
3.  WHERE    QTR_BUDGET [1] < QTR_BUDGET [2]
```

הורשה (Inheritance)

תקן SQL3 מאפשר לטיפוס נתונים מופשט ADT, להשתתף בהיררכיה של מחלקת על (Superclass) ותת-מחלקות (Subclasses) תוך קיום עיקרון ההורשה, הן של עמודות והן של פונקציות.

תחביר הפקודה :

```
1.  CREATE TYPE  Subclass_Type_Name UNDER Superclass_Type_Name
2.      (Attribute_Name_1 Data_Type_1,
3.       …,
4.       Attribute_Name_n Data_Type_n);
```

שורה 1 מגדירה שטיפוס נתונים זה הוא תת-מחלקה של מחלקה אחרת. מעבר לכך, ההגדרה של תת-מחלקה היא הגדרה רגילה של טיפוס נתונים מופשט. אם שם של עמודה או של פונקציה מוגדר מחדש בתת-מחלקה מתקיים עיקרון **העמסה** (Overloading) והגדרה זו מחליפה את ההגדרות הנובעות ממחלקת העל וההורשה. SQL3 תומך בחליפיות, כלומר בכל מקום בו ניתן להשתמש במחלקת על, ניתן להשתמש גם בתת-מחלקה.

דוגמה א' : יש להגדיר את מחלקת על עובדים במכללה ושתי תת-מחלקות, סגל אקדמי וסגל מנהלי.

```
1.  CREATE TYPE EMPLOYEE
2.          (EMPLOYEE_ID        REF (STUDENT),
3.           NUMBER     INTEGER,
4            NAME     VARCHAR (35));
5.
6.  CREATE TYPE FACULTY_STAFF UNDER TYPE EMPLOYEE
7.          (TITLE       CHAR (5),
8.           SPECIALIZATION  VARCHAR (30));
9.
10. CREATE TYPE ADMIN_STAFF UNDER TYPE EMPLOYEE
11.         (RANK        CHAR (5));
12.
13. CREATE TABLE EMPLOYEES OF TYPE EMPLOYEE
14.    VALUES FOR EMPLOYEE_ID ARE SYSTEM GENERATED;
15.
16. CREATE TABLE FACULTY_EMPLOYEES OF TYPE FACULTY_STAFF
17.    UNDER EMPLOYEES;
18.
19. CREATE TABLE ADMIN_EMPLOYEES OF TYPE ADMIN_STAFF
20.    UNDER EMPLOYEES;
```

הסבר:

- שורות 1 עד 4 מגדירות את מחלקת העל עובדים, ואת כל העמודות שלה.
- שורות 6 עד 8 מגדירות את תת-המחלקה סגל אקדמי ומוסיפה לה שתי עמודות חדשות, תואר והתמחות.
- שורות 10 ו- 11 מגדירות את תת-המחלקה סגל מינהלי ומוסיפה לה עמודה אחת, דרגה.
- שורות 13 עד 20 מגדירות את שלוש הטבלאות שיכילו את השורות של מחלקת העל ואת שתי תת-המחלקות.

נוכל עכשיו לבנות שאילתה המציגה את שמות העובדים השייכים לסגל האקדמי והם בעלי תואר פרופסור.

```
1. SELECT   NAME
2. FROM     FACULTY_EMPLOYEES
3. WHERE    TITLE = 'Prof.'
```

נשים לב שהעמודה NAME אינה מופיעה בטבלה FACULTY_EMPLOYEES אבל בגלל עקרון ההורשה מותר לנו להתייחס אליה כאילו היא חלק מטבלה זו ואין צורך לבצע Join בינה לבין הטבלה EMPLOYEES.

סיכום

מודל האובייקטים מתייחס באופן שונה לאבחנה המקובלת בין נתונים לבין לוגיקה יישומית. עד להופעת מודל זה היתה מקובלת אבחנה ברורה בין נתונים לבין לוגיקה. מערכות RDBMS מנהלות את הנתונים ותומכות בביצוע מספר פעולות בסיסיות בלבד מול נתונים אלה וכל שאר הלוגיקה של היישום נמצאת בתוכניות היישום. לעומת זאת, מודל האובייקטים משלב את הנתונים והשירותים בתוך מושג אחד, **האובייקט**. מודל זה מפשט את פיתוח היישומים כי הם יכולים להשתמש בשירותים בעלי משמעות רבה יותר, כמו רישום לקורס, העלאה בדרגה של מרצה, סגירת קורס, המהווים חלק מהאובייקט המתאים. מכאן נובע שפיתוח יישומים בסביבה מוכוונת אובייקטים מתבצע על ידי שיגור הודעות לשירותים של האובייקטים כדי לקבל את התוצאה הרצויה.

לדוגמה, האובייקט קורס יכול להכיל שירות רישום לקורס המטפל ברישום של סטודנט חדש לקורס כלשהו. שירות זה מבצע בדיקות שהסטודנט זכאי להירשם לקורס, שעדיין נותר מקום פנוי בקורס והוא מעדכן בסופו של דבר את כל האובייקטים הרלוונטיים האחרים. תוכניות היישום יכולות להשתמש עכשיו בשירות זה כ"קופסה שחורה", מבלי להכיר את פרטי הפעולות. במבט ראשון בהחלט יכול להיווצר בלבול מסוים באבחנה של קו התפר בין בסיס הנתונים לבין תוכניות היישום. חלק ניכר מהלוגיקה, שעד כה הורגלנו לראותה כחלק מתוכנית היישום, הופכת לשירותים המהווים חלק מהאובייקט. תוכנית היישום הופכת במובן זה לפשוטה יותר ועוסקת ברובה בלוגיקה של הפעלת הודעות וקבלת תגובות מאובייקטים שונים.

בעת כתיבת שורות אלו מערכות ODBMS מסחריות הוכיחו את עצמן במספר רב של יישומים ייחודיים, אולם לא בתחום עיבוד הנתונים והתנועות. עדיין נותרו מספר רב של שאלות פתוחות ולא התגבשה הסכמה כללית לגבי מודל האובייקטים, למרות שמספר רב של ארגונים פועל ליצירת סטנדרטיזציה.

הופעת מודל האובייקטים ומערכות ODBMS הביאה לטלטלה גם במודל הטבלאי אשר תרמה לעדכון ולשיפור שלו כדי שיוכל להתמודד עם האתגרים של היישומים החדשים. שיפורים אלה באים כיום לידי ביטוי בטיוטה של תקן SQL3 שנתמך באופן חלקי בלבד על ידי מספר יצרנים. ניתן לומר ברמה גבוהה של ודאות שעם הפיכת הטיוטה לתקן, נהיה עדים לשילוב חלק גדול מהתכונות בתוך מערכות RDBMS המסחריות. במשך הזמן נראה רעיונות של מודל האובייקטים משתלבים במודל הטבלאי, במקביל לשימוש במערכות ODBMS שאינן מבוססות על המודל הטבלאי.

ההיסטוריה של טכנולוגיית בסיסי נתונים רצופה במודלים שונים שהתחרו זה בזה. לכל אחד מהם היו יתרונות וחסרונות מסוימים. מודל האובייקטים מציג גישה המאפשרת גישור מסוים בין המודלים האלה ומבוסס על רמה גבוהה יותר של הפשטה וגמישות. המודל מאפשר לכל אחד להשתמש באלמנטים המתאימים לו ביותר, כמו למשל ניווט בבסיס הנתונים לעומת שפת שאילתות לא פרוצדורלית.

שאלות חזרה ותרגילים

שאלות חזרה

1. הגדר את המושגים: אובייקט, מזהה אובייקט, מחלקת אובייקטים, שירות, כמיסה, פולימורפיזם, הודעה, היררכיה סמנטית.
2. מהן התכונות ש-OID חייב לקיים?
3. הסבר את ההבדל בין אובייקט זמני לאובייקט קבוע.
4. סקור את ההבדלים העיקריים בין OODBMS לבין RDBMS. האם כדי לשמור על אובייקט קבוע יש צורך בבסיס נתונים מוכוון אובייקטים, או שניתן לנהל אותו גם בבסיס נתונים טבלאי? מה ההבדל בין שתי חלופות אלו?
5. הסבר מהי מערכת ORDBMS. מתי שימוש במערכת מסוג זה יכול להביא לתועלת רבה?
6. האם מערכת התומכת ב-BLOB היא מערכת מוכוונת אובייקטים? אם לא, הסבר את ההבדל העיקרי בין מערכת התומכת ב-BLOB לבין מערכת ODBMS.
7. הסבר מהו ADT. תן מספר דוגמאות לטיפוסי נתונים מופשטים היכולים לסייע ביישומים של בנק, של חברת ביטוח ושל בית חולים.

נספח: התוכנה First SQL וקבצי התרגול

התוכנה נמצאת באתר הוד-עמי www.hod-ami.co.il - מצא את הספר באתר (כיום נמצא בקטגוריה "מסדי נתונים") ולחץ על הלינק "קוד מקור". לאחר פתיחת הקובץ תמצא את הקבצים בתיקיה C:\HodAmiBooks\59252.

First SQL הינה תוכנית אינטראקטיבית שנועדה לערוך היכרות ראשונית עם מערכות בסיסי נתונים בכלל ושפת SQL בפרט. תוכנית זו לא נועדה לצרכים מסחריים, אלא מהווה כלי לימודי יעיל, פשוט ונוח למתחילים את דרכם בעולם מערכות המידע.

את התוכנה ניתן להתקין בכל מחשב.

שים לב: הכל מתבצע ב-DOS –
ההתקנה, ההפעלה וקריאת הוראות ההפעלה.

הוראות התקנה:

1. הקש: INSTALL.BAT
2. התוכנה תותקן תחת ספריה C:\FIRSTSQL בדיסק הקשיח.
3. להרצת התוכנה הקש firstsql בשורת הפקודה.
   (הערה: למשתמשים ב- Windows 3.x, יש קובץ בשם firstsql.pif)

בהרצת התוכנית יופיע מסך הפתיחה ולאחריו מסך העריכה הראשי של FirstSQL. לתוכנה זו נלווה בסיס הנתונים SCHOOL המשמש לתרגול. בבסיס הנתונים מצויות טבלאות המכילות נתונים. ניתן להוסיף או להוריד נתונים מהטבלאות וגם ליצור טבלאות חדשות. בנוסף תמצא בסיס נתונים FACTORY המשמש לתרגול של קובץ העזרה הנלווה לתוכנה.

בגירסה החופשית של FirstSQL לא ניתן ליצור בסיסי נתונים חדשים או לבטל קיימים, אך ניתן ליצור טבלאות חדשות (תחת בסיסי הנתונים הקיימים) ללא הגבלה (יצירת בסיסי נתונים חדשים דורשת רישוי תוכנה לגירסה מלאה).

קרא את הקובץ README1.TXT

X:\X>type readme1.txt | more

X = שם הכונן/התיקיה

תוכל גם להדפיס את הקובץ

X:\X>type readme1.txt > lpt1

תוכנית זו ניתנת כבונוס והוצאת הוד-עמי אינה תומכת בה.
התוכנית אינה חיונית לצורך הלימוד השוטף של הספר

ביבליוגרפיה

AD80	Armstrong W. W., Delobel C.	Decompositions and functional dependencies in relations, ACM Transactions on Database Systems, 5:4, pp. 404-430, 1980
AD93	Atzeni P., De Antonellis V.	Relational Database Theory, Benjamin/Cummings, 1993
AH95	Abiteboul S., Hull R., Vianu V.	Foundations of Databases, Addison-Wesley, 1995
AM76	Astrahan M. M. et al.	System R - a relational approach to data management, ACM Transactions on Database Systems 1:2, pp. 97-137, 1976
AN86	ANSI	American National Standard Institute for Information Systems, SQL ANSI X3.135, New York, 1986
AS92	Atre S.	Distributed Databases, Cooperative Processing and Networking, McGraw-Hill, 1992
AW74	Armstrong W. W.	Dependency structures of database relationships, Proceedings of the 1974 IFIP Congress, pp. 580-583, 1974
AZ96	Adriaans P., Zantinge D.	Data Mining, Addison-Wesley, 1996
BB95	ברברה בוצ'ינסקי	מערכות שרת/לקוח - טכנולוגיות ויישומים, הוצאת הוד-עמי, 1995
BC92	Batini C., Ceri S., Navathe S. B.	Conceptual Database Design - An Entity-Relational Approach, Benjamin/Cummings Publishing Company Inc., 1992
BG91	Booch G.	Object-Oriented Design with Applications, Benjamin/Cummings, 1991
BG92	Bell D., Grimson J.	Distributed Database Systems, Addison-Wesley, 1992
BM93	Bertino E., Martino L.	Object-Oriented Database Systems - Concepts and Architectures, Addison-Wesley Publishing Company, 1993
CB96	Connoly T., Begg C., Strachan A.	Database Systems, A Practical Approach to Design, Implementation and Management, Addison-Wesley Publishing Company, 1996

CE70	Codd E. F.	A relational model for large shared data banks, Comm. ACM, 13:6, pp. 377-387, 1970
CE72a	Codd E. F.	Relational completeness of database sublanguages, in Database Systems (R. Rustin ed.), Prentice-Hall, 1972
CE72b	Codd E. F.	Further normalization of the database relational model, in Database Systems (R. Rustin ed.), Prentice-Hall, 1972
CE79	Codd E. F.	Extending the database relational model to capture more meaning, ACM Transactions on Database Systems, 4:4, 1979
CE88	Codd E. F.	Domains, Primary Keys, Foreign Keys and Referential Integrity, Info DB, May, 1988
CE90	Codd E. F.	The Relational Model for Database Management: Version 2, Addison-Wesley, 1990
CJ95	Celko J.	SQL for Smarties, Morgan-Kaufman, 1995
CP66	Chen P. P.	The entity-relationship model: toward a unified view of data, ACM Transactions on Database Systems, 1:1, pp. 9-36, 1966
CP84	Ceri S., Pelegatti G.	Distributed Databases: Principles and Systems, McGraw-Hill, 1984
CR94	Cattel R. G. G.	Object Data Management, Addison-Wesley, 1994
CR96	Cattell R. G. G. (ed)	The Object Database Standard: ODMG-93 Release 1.2, Morgan-Kaufmann, 1996
DC90	Date C. J.	An Introduction to Database Systems, 5th ed., Addison Wesley. 1990
DD93	Date C. J., Darwen H.	A Guide to the SQL Standard, Addison-Wesley Reading, 1993
DW93	Date C. J., White C. J.	A Guide to DB2, 4th ed., Addison-Wesley, 1993
EN94	Elmasri R., Navathe S. B.	Fundamentals of Database Systems, 2nd ed. The Benjamin/Cummings Publishing Company Inc., 1994
FR78	Fagin R.	Multivalued dependencies and a new normal form for relational databases, ACM Transactions on Database Systems, 2:3, pp. 201-222, 1978
FV89	Fleming C., Von Halle B.	Handbook of Relational Database Design, Addison-Wesley, 1989
GR93	Gray J. N., Reuter A.	Transaction Procesing: Concepts and Techniques, Morgan-Kaufmann, 1993

GW94	Groff J. R., Weinberg P. N.	LAN Times - Guide to SQL, McGraw-Hill Inc., 1994
HM81	Hammer M., McLeod D.	Database Description with SDM - A Semantic Database Model, Tran. On Database Systems, Vol 6, No.3, 1981
HR84	הייפרמן רז	בסיסי נתונים - עקרונות, מודלים ויישומים, הוצאת הוד-עמי, 1984
HR93	Hackathorn R. D.	Enterprise Database Connectivity, Wiley Professional Computing, 1993
HR93	הייפרמן רז	בסיסי נתונים טבלאיים ושפת SQL, הוצאת הוד-עמי, 1993
HR95	הייפרמן רז	ארגון נתונים וקבצים, הוצאת הוד-עמי, 1995
HR99	הייפרמן רז	מחסני נתונים - עקרונות, ארכיטקטורה, עיצוב ויישום; הוצאת הוד-עמי, 1999
HT93	Harmon P., Taylor D.	Objects in Action: Commercial Applications of Object-Oriented Programming, Addison-Wesley, 1993
KW94	Kim W. (ed.)	Modern Database Systems: The Object Model, Interoperability and Beyond, ACM Press, 1994
LB93	Lucky B.	Advanced Topics in DB2, Addison-Wesley, 1993
LM90	Loomis M. E. S.	The Database Book, Maxwell Macmillan, 1990
LM91	Loomis M. E. S.	Objects and SQL: Accessing relational databases, Object Magazine, 1:3, pp. 68-78, 1991
LM95	Loomis M.E.S	Object Databases - The Essentials, Addison-Wesley Publishing Company, 1995
MD83	Maier D.	The Theory of Relational Databases, Computer Science Press, 1983
MH93	McFadden F. R., Hoffer J. A.	Modern Database Management, Benjamin/Cummings, 1993
MR92	Mannila H., Raiha K.	The Design of Relational Databases, Addison-Wesley, 1992
MS93	Melton J., Simon A. R.	Understanding the New SQL: A Complete Guide, Morgan-Kaufman, 1993
OV91	Oszu M. T., Valduriez P.	Principles of Distributed Database Systems, Prentice-Hall, 1991
PA87	Pratt P. J., Adamski J. J.	Database Systems: Management and Design, Boyd & Fraser Publishing Company, 1987

RB91	Rambaugh J., Blaha M., Lorensen W., Premerlani W., Eddy F.	Object-Oriented Modeling and Design, Prentice Hall, 1991
RP93	Renaud P. E.	Introduction to Client/Server Systems: A Practical Guide for Systems Professionals, Wiley Professional Computing, 1993
SM90	Stonebraker M. et al.	The third generation database system manifesto, in Proc. ACM SIGMOD, 1990
SM94	Stonebraker M. (ed.)	Readings in Database Systems, Morgan Kaufmann, 1994
SM96	Stonebraker M., Moore D.	Object-Relational DBMS's - The Next Great Wave, Morgan Kaufmann Publishers, 1996
SQ95	Simon A. R.	Strategic Database Technology: Management for the Year 2000, Morgan Kaufmann, 1995
SR86	Stonebraker M., Rowe L.	The design of POSTGRES, in ACM SIGMOD, 1986
SS80	Smith D. C., Smith J. M.	Conceptual Database Design, Infothec State of Art Report on Database Design, 1980
TG93	Tillmann G.	A Practical Guide to Logical Database Modelling, McGraw-Hill, 1993
TT90	Teorey T. J.	Database Modeling and Design: The Entity-Relationship approach, Morgan Kaufmann, 1990
UJ82	Ulmann J. D.	Principles of Database Systems, Computer Science Press, 1982
UJ88	Ulmann J. D.	Principles of Database and Knowledge-Base Systems, Vol I, Computer Science Press, 1988
UJ89	Ulmann J. D.	Principles of Database and Knowledge-Base Systems, Vol II, Computer Science Press, 1989
UW97	Ullman J.D., Windom J.	A First Course in Database Systems, Prentice-Hall International Inc, 1997
VG90	Vossen G.	Data Models, Database Languages ans Database Management Systems, Addison-Wesley, 1990
VS91	Vang S.	SQL and Relational Databases, MicroTrend Books, 1991
WC93	Wertz C. J.	Relational Database Design: A Practitioner's Guide, CRC Press, 1993
WC96	Widom J., Ceri S.	Active Database Systems, Morgan-Kaufmann, 1996
ZA96	Zantinge D., Adriaans P.	Managing Client/Server, Addison-Wesley, 1996

אינדקס

האינדקס מציג את המונחים העיקריים שנסקרו בספר זה. האינדקס מוצג במיון אלפביתי על פי המונח העברי. ליד כל מונח עברי מופיע המונח הלועזי, הקיצור המקובל למונח הלועזי ורשימת העמודים בהם המונח מופיע. מספר העמוד, בו מופיעה ההגדרה העיקרית של המונח, מופיע בצורה מודגשת.

ב

ח

ט

י

כ

ל

נ

ס

ע

ק

ר

ש

ת

אינדקס לועזי

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

ת.ד. 6108 הרצליה 46160 **הזמנות באתר**
info@hod-ami.co.il www.**hod-ami**.co.il

קטלוג 2019

מחירים מעודכנים והנחות באתר הוד-עמי

	כולל	עמ'	מחיר*
אינטרנט - מפתחי אתרים/גרפיקה			
אמא, אבא - בניתי אתר באינטרנט (HTML)		256	29
הגדל את הכנסות העסק שלך באמצעות **פרסום בגוגל Google AdWords**		300	249
HTML5 המדריך לבניית אתרים ולמערכות WEB, הדור הבא – מהד' 3		392	159
The **Java** Tutorial סדנת לימוד	CD	768	179
JavaScript סדנת לימוד		586	159
מדריך **ASP.NET MVC 4**		514	199
ASP.NET 3.5 סדנת לימוד בשפות #C ו-VB		824	99
תכנות			
Code Complete – מדריך מעשי לפיתוח תוכנה		288	139
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
Visual C# 3.0 סדנת לימוד		656	99
ללמוד **C** - מהד' 3		480	139
שפת **אסמבלי** למחשב האישי, מהד' 2		314	89
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 5		162	95
PC - חומרה, תוכנה ורשתות			
מדריך Hacking ואבטחת מידע, מהד' 2		428	169
מדריך **חומרה ותוכנה** לטכנאי PC - מהד' 5 (כולל חלונות 7/8)		752	189
מדריך **רשתות** לטכנאי PC ולמנהלי רשת - מהד' 4		608	219
Windows			
Windows 8.1 מדריך למשתמש		544	39
Windows 8 מדריך למשתמש		438	19
Windows 7 צעד-אחר-צעד		272	19
LINUX			
LINUX למתקדמים, טיפים, טריקים ותכנות ב-BASH		226	189

* מחיר מומלץ לצרכן כולל מע"מ. הספרים נמכרים בהנחה בהוצאה

היכנס לאתר להתעדכן בספרים החדשים ובמחירי המבצע בהוצאה

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il

09-9564716

	כולל	עמ'	מחיר*
גרפיקה			
Flash – ספר הדרכה ותרגילים		122	64
אינדיזיין – ספר הדרכה ותרגילים		132	72
Illusatrator – ספר הדרכה ותרגילים		120	64
Photoshop צעד אחר צעד (ש/ל, למתחילים), מהד' 3		200	89
מדריך לתוכנת העיצוב והאנימציה **3ds max** (2 כרכים)	CD	1400	289
OFFICE			
יישומי סטטיסטיקה בגיליון אלקטרוני Excel		86	69
סטטיסטיקה יישומית – א'+ב'		194	97
טבלאות ציר – ניתוח נתונים חכם		116	87
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 5		162	95
Access 2016 צעד אחר צעד		384	169
Word 2016 צעד אחר צעד		150	59
Excel 2016 צעד אחר צעד		336	129
עוד ספרים בגרסאות קודמות (2007 ו-2003) ניתן למצוא באתר הוד-עמי			
ניהול, כלכלה ושונות			
יסודות המימון שרוני ומופקדי		560	175
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
ניהול ממוקד לעשות יותר עם מה שיש (כריכה קשה) - מהד' 4		358	133
לי זה עולה יותר (תמחיר) (כריכה קשה) - מהד' 3		368	129
מערכות מידע			
Oracle SQL יכולות מתקדמות		626	249
SAS (**Statistical Analysis System**) – ספר לימוד		256	169
בסיסי נתונים ושפת **SQL** – עקרונות ועיצוב		648	149
ניתוח מערכות מידע כולל מתודולוגיית ה- **UML**		818	229
המדריך העברי השלם **UML**		346	329
ספרים דיגיטליים			
לרכישת ספרים לצפייה בפורמט PDF היכנס לאתר לקטגוריה "ספרים דיגיטליים"			
קבצי תרגול לספרים			
קבצי תרגול לספרים שונים תמצא באתר בקטגוריה "קבצי תרגול לספרים"			

* קטלוג 2019. מחיר מומלץ לצרכן כולל מע"מ. הספרים נמכרים בהנחה בהוצאה

היכנס לאתר להתעדכן בספרים החדשים ובמחירי המבצע בהוצאה

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il

09-9564716