Oracle SQL

יכולות מתקדמות

מדריך לשולף המהיר

כולל גרסה 11.2

עמיאל דייויס

עורך טכני: זהר אלקיים

הוצאת הוד-עמי
ספרי מחשבים

עריכה טכנית: **זהר אלקיים**

עריכה לשונית ועיצוב: **שרה עמיהוד, יצחק עמיהוד**

עיצוב עטיפה: **חיה וייט, נלי מגיט**

שמות מסחריים

הודעה

ת.ד. 6108 הרצליה 46160

טלפון: 09-9564716 פקס: 09-9571582

www.hod-ami.co.il

הודפס בישראל יולי 2011

מסת"ב ISBN 978-965-361-409-3

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

אני שמח לברך את כותבי הספר, ולא פחות מכך את הכתיבה והספר עצמו, שמחד מבטא הכרה בראשוניות וההובלה של חברת ORACLE העולמית בבסיסי נתונים וניהולם, ומאידך פורס בשפה העברית מדריך איכותי למשתמש עם עומק רב.

אני חש שהקריאה והשימוש בספר יביא לקורא בו ערך מוסף משמעותי, ועל כך תבוא הברכה על הכותבים.

משה חורב

VP באורקל העולמית
ומנהל כללי, ישראל

תודות

ברצוני להודות מקרב לב לאנשים הבאים, במקום המשמש לי כבית מקצועי חם, שנים רבות - "כללביט מערכות": **לקרן שטיינמץ כחילה** על קריאה פרטנית ומאומצת שהניבה הערות תורמות. **לרבקי פרץ** על הקריאה של הגרסה הראשונית של אחד מהפרקים המאתגרים בספר. **למירב שוקרון** ("פורפה") על העידוד והביקורת הבונה. **לסרגיי מסיונדז'ניק** על כל התשובות שנתת לי על נושאים טכנולוגיים שונים. **לאלון קבלו** על התמיכה והדחיפה (הפרק על XML מוקדש לך). **למשה צוריאל**, על האמונה. **לאילנית בוקרה**, המככבת בספר בלא ידיעתה, על האמון והתמיכה המתמשכת. **ליוסי טאבו** שהשכיל לראות בתוכי את ה-DBA מעבר למעטה המתכנת וטיפח אותי במשך שנים רבות. **לאודי פליישר**, אתה ואני יודעים למה. **לגיל ים**, על התמיכה הראשונית. **ליוסי ז'אן** על העידוד.

לקומץ מקצוענים, שהסכימו בנדיבות רבה לאפשר לי לצטט אותם: תום קייט (Tom Kyte), ג'ונתן לואיס (Jonathan Lewis), ג'יימס קופמן (James Koopmann), אנדרס גאבור (András Gábor) וג'אן גויברטס (Jan Goyvaerts) - הידע שלכם ולא פחות חשוב, הנכונות לחלוק אותו עם העולם הגדול, השפיעו ועיצבו את ה-"אני מאמין" שלי מבחינה מקצועית, במשך שנים ארוכות.

בהוצאת "**הוד-עמי**" - **ליצחק עמיהוד**, העורך הלשוני של ספר: תודה רבה. אני יודע שאני תופעה לשונית מצערת. **לשרה עמיהוד**, על האמונה המתמשכת בפרויקט הארוך הזה.

לעורך הטכני של הספר, **זהר אלקיים**, שנטל על עצמו משימה מאתגרת וביצע אותה על הצד הטוב ביותר, בפרק זמן קצר ולחוץ - תודה.

לציירת המדהימה של עטיפת הספר, **חיה וייט**, תודה על הסבלנות.

לצמד חברים שמלווה אותי שנים ארוכות - **ניר ושחר**.

לאשתי, **חגית** - "שלי ושלכם שלה הוא". לארבעת בניי, תודה שאתם קיימים.

תודה לבורא עולם.

תוכן עניינים

הקדמה

בראשית, או ליתר דיוק ב-1979[1], חברת Relational Software, Inc. הידועה היום כ-Oracle, הוציאה לשוק את בסיס הנתונים הטבלאי (הרלציוני) המסחרי הראשון.

מאז, ניב **SQL** של **Oracle** התפתח מאוד והגיע לרמות מרשימות של תחכום ועוצמה, במיוחד בגרסאות 10 ו-11. היכולת המובנית של SQL, עיבוד בקבוצות, הקנתה לשפה זו עוצמה רבה. השילוב של יכולת עיבוד זו, יחד עם יכולות עיבוד פרוצדורליות מגוונות, מציג רמה חדשה של תחכום, עוצמה וביצועים שלא היו קיימים קודם לכן ב-SQL. פעולות שקודם לכן חייבו שימוש בשפות אחרות, כגון PL/SQL, JAVA, ו-C, ניתנות לביצוע כולן בתוך פקודות SQL. חישובים וטרנספורמציות שבעבר דרשו כלי צד שלישי, מובנים כעת בתוך מנוע SQL עצמו, ובכך מפחיתים את העומס על הרשת, התוכניתן ומינהלן בסיס הנתונים (DBA). הגברת היכולת של SQL, גרמה למורכבות טבעית ומובנת בתחביר של השפה ולעתים גם לרתיעה מסוימת מיכולות אלו. ספר זה מציג מספר רב של יכולות מתקדמות בצורה פשוטה, ברורה ובהירה, תוך כדי שימוש רב בדוגמאות ובשיטת צעד אחר צעד בנבכי ההרחבות השונות של השפה.

ספר זה נולד מתוך צורך הולך וגובר במדריך דידקטי ומתודי ליכולות החדשות של גרסאות 10 ו-11. התיעוד של Oracle תמיד מומלץ, אבל הוא גם מניח, ברוב המקרים, שיש למשתמש ידע קודם שלא בהכרח נמצא ברשותו, או שהוא מפנה את הקורא לתיעוד אחר שלא נמצא ברשותו, או שאין לו את הזמן הפנוי להתעמק ולפענח אותו. במקרים שבהם יש צורך בידע קודם, ננסה להעביר את עיקרי הדברים תוך כדי העמקה בתכנים רלוונטיים לנושא, מבלי להיכנס לנקודות עמומות או שאינן חיוניות. הספר נכתב מתוך כוונה להציג ולעסוק ביכולות החדשות, אבל הוא גם כולל את יכולות השפה שהתווספו בגרסאות קודמות.

[1] תשע שנים לאחר הפרסום הפומבי של המאמר המפורסם " A Relational Model of Data for Large Shared Data Banks" על ידי ד"ר Edgar Frank "Ted" Codd ותרגומו המעשי לניב הראשון של שפת SQL על ידי ד"ר Donald D. Chamberlin וד"ר Raymond F. Boyce.

ידע מוקדם

אנחנו מניחים שיש לך ידע מוקדם והכרה של בסיסי נתונים טבלאיים (Relational Databases) ושפת SQL (Structured Query Language) – שפת שאילתות מובנית. אם לא כך הדבר, תוכל לחדש או לרענן את ידיעותיך בנספח ג', "מדריך SQL בסיסי". בכל מקרה, הנספח מומלץ בחום, מכיוון שיש בו מלבד ידע בסיסי משובח, גם סיפורים שיפתיעו גם אנשי אורקל מנוסים, כגון, דוגמה מעשית לשימוש מוצדק בשאילתה עם מכפלה קרטזית.

למי הספר מיועד

ספר זה מיועד לאנשים עובדים. אין תיאוריות גדולות בספר הזה, אלא רק פרקטיקה ועבודה מעשית כדי למקסם את הפעולות שאורקל יכול לעשות למענך. מי שמעוניין להתעמק ברזי השפה מטעמים אקדמיים או לימוד מדעי המחשב, טוב ייעשה אם ירכוש ספר אחר. מי שרוצה ללמוד איך לא להמציא את הגלגל מחדש, ורוצה להימנע מכתיבת תוכניות ארוכות לביצוע פעולות על נתונים בבסיס הנתונים של אורקל – זה הספר בשבילו.

מבנה הספר

פרק 1 דן בהרחבות ל- Group By, וכולל דוגמה מעניינת לסוגיית Query Rewrite של שאילתה כבדה, כך שתופנה אל Materilized View. חלקו השני כולל פירוט של האופרטורים Pivot ו-Unpivot. בסיום הפרק נציג הודעות שגיאה רלוונטיות.

פרק 2 דן בשליפות היררכיות. בחלקו הראשון של הפרק נתעסק בשליפות היררכיות בתחביר הוותיק של Oracle, ומובאת בו דוגמה מעשית לשיפור ביצועים מרשים בשאילתה בעייתית. בסיום הדיון בנושא זה נסקור בקצרה את הודעות השגיאה בשאילתות שנכתבו על פי התחביר הישן. בחלקו השני של הפרק נראה את התחביר החדש של גרסה 11.2 לביצוע שליפות היררכיות, ובסיומו יש סקירה קצרה של הודעות השגיאה העלולות להתקבל בשימוש מוטעה בתחביר החדש.

פרק 3 עוסק במגוון הפונקציות האנליטיות של SQL ובמידת הצורך מתעמק בהבדלים שביניהן. יש בו התייחסות מפורטת למשפט Window המאפשר לבצע סיכומים מצטברים וחישובים, כמו למשל חישוב של "ממוצע זז". נתייחס למילת המפתח Keep וליכולות האנליטיות החדשות שהתווספו בגרסה 11.2. הפרק כולל סקירה קצרה על השימוש ב-Udag (User Defined Aggregate). בסיום הפרק נסקור בקצרה את הודעות השגיאה שרלוונטיות לפונקציות אנליטיות ונצביע על הדרך לטפל בהן.

פרק 4 סוקר באופן יסודי את המשפט Model ומסביר את התחביר "המאיים" קמעה שלו. בפרק יש דיון בביצועים, וסקירה של הודעות שגיאה אפשריות.

פרק 5 מראה את היכולות של Oracle בתחום Regular Expression, ומציג דוגמאות מעניינות לעיבוד קוד PL/SQL באמצעות Regular Expression. הפרק סוקר את כל האופרטורים הקיימים ב-Oracle עבור Regular Expression, מתעמק בביצועים של Regular Expression וגם לא מתעלם מהמגבלות של Regular Expression. לסיום ניתנת סקירה של הודעות השגיאה הרלוונטיות.

פרק 6 מדגים יכולות XML של Oracle. יש בו תיאור מקיף של הפיכת נתונים טבלאיים למסמכי XML, כמו גם של המרת מסמכי XML למבנה טבלאי. נקרא גם לשירות רשת (Web Service) מתוך בסיס הנתונים ונציג פירוק של מסמך XML המתקבל למבנה טבלאי. מדי פעם תהיה סטייה מהנושא המרכזי, כדי להדגים יכולות מעניינות של XML (כמו למשל, קיבוץ מחרוזות הכולל גם מיון לתוצאה אחת של Clob, בפקודה אחת בלבד). בסוף הפרק נביא אוסף מייצג של הודעות שגיאה וכיצד מטפלים בהן.

הספר כולל ארבעה נספחים:

נספח א' כולל תכונות ומאפיינים ("פיצ'רים", בסלנג המקובל) נוספים של Oracle.

נספח ב' מראה אוסף רעיונות/טכניקות/מנהגים נלוזים, שלדעת המחבר מומלץ לא לבצע בבסיס נתונים של Oracle. בכל אחד ממקרים אלה יוצגו הטיעונים וההוכחות הדרושים.

נספח ג' הוא למעשה מדריך SQL בסיסי. אם אינך מכיר SQL בכלל, מומלץ מאוד לעבור על נספח זה כדי ללמוד וגם כדי לרענן ידיעות. בכל מקרה, מומלץ לעבור על הנספח הזה, ולו גם ברפרוף.

נספח ד' הוא אוסף של מקרים ותגובות, וגם אוסף של הודעות שגיאה. כל הפריטים מלווים בהסבר קצר, מה קרה ומדוע, וכיצד ניתן לטפל בהם.

דוגמאות הקוד

במהלך ההדגמות בספר נשתמש בכלי שורת פקודה של אורקל – Sql*Plus. בחרנו בו מכיוון שהוא נמצא בכל התקנת client של אורקל. הכלי קצת מסורבל, אבל מתרגלים אליו לאחר תרגול מתאים. מבחינת גרסאות של בסיס הנתונים, כל קובץ נבדק על רוב הגרסאות שקיימות כיום בשוק (גרסאות 9.2, 10.2, 11.2), ובמידה שיש הבדל שראוי לתשומת לב, הוא הוצג והוסבר. במחוון הפקודה, במקרה ונשתמש בפקודה שרלוונטית החל מגרסה מסוימת של אורקל נראה גם את מספר הגרסה. כאשר נדגים פקודה שהייתה קיימת "מאז ומעולם", כלומר מגרסה 10 ואילך, נקפיד להשתמש במחוון הסטנדרטי <SQL.

הדוגמאות שמוצגות בספר זמינות לכל דורש באתר הוד-עמי, ותוכל להריץ אותן בכלי התוכנה החביב עליך.

את הקבצים ניתן להוריד מאתר האינטרנט של הוצאת הוד עמי: www.hod-ami.co.il

מצא את הספר באתר ואת הלינק "**קוד מקור**" להורדת הקבצים. לחץ עליו ועקוב אחר ההוראות.

אם לא תגדיר אחרת, יועתקו הקבצים אוטומטית לדיסק שלך, לתיקייה זו:

C:\HodAmiBooks\59454\

תוכל לבחור בעת ההתקנה בכל תיקייה אחרת. בספר נתייחס לתיקיית ברירת המחדל.

חשוב הקובץ **אינו מכיל את Oracle או כל תוכנה אחרת.** את התוכנה יש לרכוש ולהתקין **לפני** תחילת השימוש בספר זה.

1

הרחבות ל- Group by

"הגלים ורוחות הים עומדים תמיד לצדו של הספן המיומן."[1]

פעם, לפני זמן לא רב כל כך, אחזור נתונים ("שליפות" בסלנג המקצועי) שכלל גם סיכומים והקבצות (דהיינו שליפות אגרגטיביות[2]), יכול היה להיעשות רק על ידי Group By ו-Having[3]. כאשר נדרשה פעילות מורכבת יותר, התוכניתן החרוץ נאלץ לייצא את תוצאות השליפה לכלי תוכנה אחר כלשהו, ולבצע דרכו ניתוחים מורכבים יותר. כיום, כל ההרחבות שרק ניתן לחשוב עליהן נמצאות באופן מובנה[4] ב-Oracle. לנו לא נותר אלא לסקור אותם באופן דידקטי ומתודי, בדרך שנשתמש בהן לרוב בספר. מה היינו עושים בלי ההרחבה המדוברת (למשל, בגרסאות קודמות או מתוך פחד/אימה/חוסר ידע) ואיך באופן אלגנטי, ההרחבה המדוברת פותרת לנו את הבעיה. נציג את ההרחבות שהתווספו על פי סדר כרונולוגי של הוספתן לבסיס הנתונים: Rollup, Cube, Grouping Sets. לאחר מכן נתעכב מעט על סוגיית Query Rewrite (יכולת לשכתב שאילתה כך שתעבוד מול תוצאות מחושבות מראש ועל ידי כך להאיץ ביצועים באופן דרמטי), ונכיר גם את שני האופרטורים החדשים Pivot ו-Unpivot.

---

[1] Edward Gibbon, 27 באפריל 1737 - 16 בינואר 1794. מתוך "שקיעתה ונפילתה של האימפריה הרומית".

[2] **שליפות אגרגטיביות** הן שליפות שמפעילות פונקציות אגרגטיביות, שהן פונקציות המחשבות את התוצאה לא ברמת השורה, אלא ברמת הקבוצה (המורכבת משורה אחת לפחות). נעסוק בהן בקצרה בנספח ג', ובפונקציות אנליטיות נעסוק בהרחבה בפרק 3.

[3] תקן SQL המוקדם ביותר הוא SQL-86, אשר כלל אך ורק Group by ו-Having. החל מתקן SQL-99 הופיעו ההרחבות שנעסוק בהן בפרק זה.

[4] הפונקציות Cube, Rollup וגם הפונקציה Grouping הופיעו לראשונה בגרסת 8i, ולטובת הדיוק ההיסטורי בגרסה 8.1.5. בגרסה 9i (9.0.1) התווספו הפונקציות Grouping_Id ו- Group_Id, יחד עם ההרחבה של Grouping Sets. בגרסה 11g הופיעו האופרטורים Pivot ו- Unpivot.

דוגמה פשוטה ל-Rollup

הנה בקשה קלאסית ולגיטימית של מנהל, אשר עשויה להיות מוצגת בחיים האמיתיים: "כתוב לי בבקשה דוח שמראה את כמות הבלה בלה, מקובץ לפי ידה ידה"[5]. בעברית פשוטה ולא של ג'רי סיינפלד, המנהל היה אומר למשל: "כתוב לי בבקשה דוח שמראה את כמות המוצרים שנמכרו מקובצים לפי סוכני מכירות וסוג מוצר". מקצוען אמיתי כותב את השליפה בפחות מדקה, בעיצוב הדוח הוא ישקיע עוד כחצי שעה, ולבסוף, שמח וטוב לב, הוא מגיש את הדוח למנהל. מכאן יכולים לקרות שני דברים: אפשרות ראשונה, הנדירה: המנהל מרוצה ממה שהוגש לו (כולל העיצוב); והאפשרות השנייה: המנהל מבקש שהדוח יוגש בגופן שונה, כותרת אחת או יותר תוחלפנה ו/או תודגשנה, ועוד. אם האפשרות הראשונה מתקיימת - סיימת את המשימה. אבל, אם האפשרות השנייה מתקיימת (וזה לרוב מה שקורה), אתה צריך לשוב אל מחולל הדוחות האהוב עליך ולבצע את השינויים הדרושים ("קטנוניות של מנהלים" אתה בוודאי אומר בליבך). כעת אתה מגיש את הדוח המשופץ למנהל, הוא סוקר אותו בעניין, ואז מאירות עיניו בשמחה ביקורתית ניהולית אמיתית: "מה זה? חסרים כאן סיכומי ביניים וסיכום כללי. ככה מגישים דוח?"

מכאן אתה יכול לעשות שני דברים: אפשרות ראשונה, ככל הנראה נדירה יחסית - להתפטר, להגר למדינה רחוקה, להתמקם על גדת נהר ולעסוק בדייג, וכך גם לוותר על המשך הקריאה בספר זה. אפשרות שנייה, שהיא הנפוצה - לחזור לעמדה שלך ולתקן את הדוח באופן שיכיל את כל הסיכומים הדרושים לפי סוכנים ומוצרים, למשל. לפני שהכרת את האופרטור Rollup ייתכן שהיית כותב את שאילתת האחזור בדרך זו:

קוד 1-1:

```
SQL>select * from
  2   (select agent_no, product, sum(commission)
  3   from commission_for_agent
  4   group by agent_no, product
  5   union all
  6   select agent_no, null, sum(commission)
  7   from commission_for_agent
  8   group by agent_no
  9   union all
 10   select null, null, sum(commission)
 11   from commission_for_agent)
 12   order by agent_no, product;
```

[5] אם יש לכם מנהל שמצטט מ-"סיינפלד", מצבכם טוב. אם הייתם חוקרים את שורשי הביטוי המפורסם, קרוב לוודאי שהייתם מגלים שהוא היה קיים גם לפני "סיינפלד" (עם כל הכבוד, ויש כבוד).

```
  AGENT_NO PRODUCT          SUM(COMMISSION)
---------- --------------- ---------------
         1 Flat Screen                  10
         1 Mobile Computer              20
         1                              30
         2 Flat Screen                   5
         2 Mobile Computer              10
         2                              15
         3 Flat Screen                  40
         3 Mobile Computer              25
         3                              65
         4 Flat Screen                  30
         4 Mobile Computer               5
         4                              35
                                       145

13 rows selected.
```

קוד זה אכן עושה את העבודה, ללא ספק. שורות 2 עד 4 מספקות את הסיכום הפרטני ברמת המוצר והסוכן; שורות 6 עד 8 מייצרות את הסיכום ברמת הסוכן (ללא קשר למוצר); שורות 10 עד 11 מציגות את הסיכום הכללי (סה"כ). בין שאילתה לשאילתה נמצאת הפקודה Union All והמבנה הזה כולו מוקף בשאילתה חיצונית כדי שנוכל לבצע מיון על הכל. אבל, האם זה חייב להיות מסורבל עד כדי כך? חיסרון נוסף, ולא פחות חשוב - בהנחה שהשאילתות שאתה מציג בחיים האמיתיים מתחקרות טבלאות קצת יותר גדולות ומורכבות מהדוגמה שהבאנו כאן, ולעתים כוללות גם אופרטורי Join המחברים כמה טבלאות יחדיו, סביר שהמעבר המשולש על הנתונים לא יתרום למהירות הביצוע. כאן ניתן להיעזר באופרטור Rollup שעורך מעבר יחיד על הטבלה:

קוד 2-1:

```
SQL>select agent_no, product,
  2  sum(commission) from commission_for_agent
  3  group by rollup (agent_no, product);

  AGENT_NO PRODUCT          SUM(COMMISSION)
---------- --------------- ---------------
         1 Flat Screen                  10
         1 Mobile Computer              20
         1 <Null>                       30
         2 Flat Screen                   5
         2 Mobile Computer              10
         2 <Null>                       15
         3 Flat Screen                  40
         3 Mobile Computer              25
         3 <Null>                       65
         4 Flat Screen                  30
         4 Mobile Computer               5
         4 <Null>                       35
    <Null> <Null>                      145

13 rows selected.
```

הסבר פשוט

האופרטור Rollup מחשב סיכומים[6] ברמות פירוט שונות, מהרמה הנמוכה ביותר ועד לרמה הגבוהה ביותר. בדוגמה הקודמת, חישבנו לכל סוכן את סכום העמלות שלו אשר מפורטות על פי המוצרים שמכר (כמו שעושים על ידי Group By רגיל). לאחר מכן חישבנו סיכומי ביניים לכל סוכן ולבסוף חישבנו את הסיכום הכללי. כדי להציג את סיכומי הביניים ואת הסך הכל, התווספו בתוצאות חמש שורות, ארבע שורות של סיכומי ביניים ושורה אחת של הסך הכל הכללי. כלומר, על שני הביטויים ב- Group By יצר האופרטור Rollup שלוש רמות של סיכום: המפורטת ביותר, במקרה זה סוכן ומוצר ללא ערכי Null באף שדה; בשורות שהתווספו על ידי Rollup הוצג סיכום לסוכן שכולל את מספר הסוכן וערכי Null במקום המוצר; ולבסוף ברמה הגבוהה ביותר - סיכום כללי לכל הסוכנים והמוצרים, כאשר במקום הערך של הסוכן והמוצר הוצגו ערכי Null.

הפונקציה Grouping

פעולה זו מעלה את השאלה הבאה: כיצד נבחין בין שורות המכילות ערכי Null שהיו קודם לכן בטבלה, לבין שורות שיש בהן ערך Null המייצג סיכומי ביניים וסיכומים שנוצרו על ידי Rollup? כאן באות לעזרתנו שלוש פונקציות אגרגטיביות. נדגים תחילה את הפונקציה הראשונה והוותיקה מביניהן - Grouping:

קוד 3-1:

```
SQL>select grouping(agent_no) ga,
  2          grouping(product) gp,
  3          agent_no,
  4          product,
  5          sum(commission) sum_co
  6 from commission_for_agent
  7 group by rollup (agent_no, product);

GA GP    AGENT_NO PRODUCT             SUM_CO
-- -- ---------- --------------- ----------
 0  0          1 Flat Screen             10
 0  0          1 Mobile Computer         20
 0  1          1 <Null>                  30
 0  0          2 Flat Screen              5
 0  0          2 Mobile Computer         10
 0  1          2 <Null>                  15
 0  0          3 Flat Screen             40
 0  0          3 Mobile Computer         25
 0  1          3 <Null>                  65
 0  0          4 Flat Screen             30
```

[6] לרוב משתמשים בו עם הפונקציה Sum ולכן הביטוי "סיכומים".

```
 0   0           4 Mobile Computer          5
 0   1           4 <Null>                  35
 1   1    <Null>     <Null>                145

13 rows selected.
```

הפונקציה Grouping תחזיר ערך 0 כאשר יש ערך שמתקבל מהטבלה (גם אם הוא Null) ותחזיר ערך 1 אם יש ערך Null שנוצר מחישוב סיכום ביניים או סך הכל. את הסיכומים אפשר לקבל על ידי אחד מהאופרטורים Rollup/Cube/Grouping Sets (כל הפעולות האלו יוסברו בהמשך), אם הארגומנט שהפונקציה קיבלה תואם לביטוי ב- Group By. בדוגמה שלנו קראנו לפונקציה פעמיים: בשורה הראשונה העברנו לה ארגומנט של סוכן (השדה בטבלה Agent_no), ובשורה השנייה העברנו לה ארגומנט של מוצר (השדה Product). בנתונים שנשלפו, הדגשנו שתי שורות: השורה הראשונה מראה את סיכום הביניים של העמלות שהתקבלו על ידי מכירת המוצרים של סוכן 1, ואכן העמודה gp, שנוצרה כתוצאה מהקריאה הזו - grouping(product) - מראה את הערך 1. השורה האחרונה מראה את הסיכום של כל המוצרים וכל הסוכנים, ובה שתי הקריאות ל- grouping(agent_no) ול- grouping(product) ושתיהן מראות את הערך 1. כדי שהדוח יהיה יותר קריא (שיפור הקריאות שלו), ניתן לשלב את הפונקציות בביטוי Case כדי ליצור כותרות בדוח. במקום ביטוי Case יכולנו גם להשתמש בפונקציה הוותיקה Decode[7], אבל לא בטוח שהשאילתה הייתה קריאה כל כך. שימו לב למבנה ה- case בשורות 1 עד 11:

קוד 1-4:

```
SQL>select case when
  2           grouping(agent_no) = 0 and
  3           grouping(product) = 0 then
  4                       to_char(agent_no)
  5           when grouping(product) = 1 and
  6                 grouping(agent_no) = 0  then
  7                   'Total for ' ||  to_char(agent_no) || ':'
  8          when  grouping(product) = 1 and
  9                 grouping(agent_no) = 1  then
 10                  'General total:'
 11          end as agent_no,
 12          product,
 13          sum(commission) sum_co
 14  from commission_for_agent
 15  group by rollup (agent_no, product);
```

[7] מעניין לציין ש- Decode, פונקציה ישנה מאוד (קיימת מימי גרסה 6 לפחות) ועוד כמה פונקציות ותיקות אחרות אינן נתמכות ישירות ב- PL/SQL. פקודת ההשמה הבאה לדוגמה, הייתה נכשלת (גם ב- 11g):

ll_temp := decode(ll_temp, null, 2);

```
AGENT_NO         PRODUCT          SUM_CO
---------------- ---------------- ------
1                Flat Screen          10
1                Mobile Computer      20
Total for 1:     <Null>               30
2                Flat Screen           5
2                Mobile Computer      10
Total for 2:     <Null>               15
3                Flat Screen          40
3                Mobile Computer      25
Total for 3:     <Null>               65
4                Flat Screen          30
4                Mobile Computer       5
Total for 4:     <Null>               35
General total:   <Null>              145

13 rows selected.
```

את שתי הפונקציות הנוספות, Grouping_Id ו- Group_Id, נדגים בהמשך. כסיכום ביניים ל- Rollup נציין את הנקודות הבאות:

- על ידי Rollup ניתן לחשב סיכומי ביניים בהתאם לביטויים שרשומים לאחר Group By. הערכת הביטוי נעשית משמאל לימין. בתחילה מחושבות השורות הרגילות (כלומר, כמו ב- Group By רגיל) ולאחר מכן מחושבים סיכומי ביניים ולבסוף מחושב הסיכום הכללי. כזכור, הדוגמה שלנו הייתה:

```
group by rollup (agent_no, product)
```

כלומר, קודם חושבו העמלות לסוכן באופן "רגיל", לאחר מכן חושב סך העמלות למוצרים שנמכרו על ידי הסוכן, ולבסוף חושב הסיכום הכללי של העמלות של כל המוצרים וכל הסוכנים.

- שורות חדשות נוצרות על ידי Rollup. ראינו זאת קודם, ונדגיש זאת שוב. רמות הפירוט של הסיכום יכללו תמיד את מספר הביטויים הכלולים בתוך אופרטור ה-Rollup, ביחד עם שורה של סיכום כללי.
- לא חייבים לבצע Rollup על כל השדות שמופיעים ב- Group By, אבל כן חייבים לקבץ לפיהם. לזה קוראים Partial Rollup. נתייחס לזה מייד.
- בדוגמה שלנו השתמשנו בפונקציה Sum. עם זאת, ניתן להשתמש בפונקציות אגרגטיביות אחרות כולל פונקציות אגרגטיביות שהוגדרו על ידי המשתמש (UDAG)[8].
- כאשר משתמשים בפונקציה אגרגטיבית עם אופרטור Distinct בין הסוגריים, עלולות לצוץ אי-הבנות בהשוואה בין הסיכום הכללי לסכומי הביניים. בגרסאות קודמות,

[8] הפונקציונליות המרשימה הזאת התווספה בגרסה 9.1 (יולי 2001).

נספחים

א'

תכונות מעניינות של Oracle

"אנחנו לא צריכים קסם כדי לשנות את העולם, אנו ככבר נושאים בתוכנו את כל הכוח שאנו צריכים." [1]

בנספח זה, נראה עוד כמה דברים יפים, שלא הצדיקו, או שלא התאימו מבחינת תוכן לאחד מהפרקים בספר. יכולנו לוותר עליו ולכתוב ספר נוסף ומורחב על נושאים אלה, אבל סברנו שדרך זו טובה יותר. התכונות, או כפי שמקובל לעתים לכנותן "הפיצ'רים", מסודרות על פי סדר הופעתן בספר, ואין בכך שום רמז לגבי מידת החשיבות שאנו מקנים להן.

הפקודה Explain plan והצגת התוצאות שלה

הפקודה Explain plan שייכת למשפחת פקודות DML. מטרת הפקודה להראות את תוכנית הגישה לפקודת SQL שהמשתמש עומד להריץ, ובתנאים מסוימים להראות גם את תוכנית הגישה שהורצה בפועל, לאחר ביצוע הפעילות. במהלך הספר ראינו בעיקר תצוגות של תוכניות גישה לשאילתות, אך בסעיף הזה נתמקד בשלוש יכולות של הפקודה Explain Plan:

1. הצגה של תוכנית הגישה לפקודות שאינן שליפה.
2. קבלת מידע מדויק מתוכנית הגישה.
3. קבלת מידע על השאילתה שהרצנו, לאחר ההרצה שלה.

[1] ג'יי. קיי. רולינג, נולדה ב-31 ביולי 1965.

מעטים שמים לב לכך שאת הפקודה Explain plan ניתן להפעיל גם על הפקודות הבאות:

- **DML** שאיננו select. כלומר, insert, delete, update ו-merge.
- **DDL**, והכוונה כאן לפקודות ה-DDL האלה: create index, create table as select

בדוגמה הבאה נראה את תוכנית הגישה ליצירה של שני אינדקסים עבור טבלה. את האינדקס הראשון גם ניצור בפועל, כדי להראות שאורקל יכול להשתמש בו להאצת היצירה של האינדקס השני:

קוד א-1:

```
SQL>explain plan for
  2  create index all_objects_copy_a1
  3  on all_objects_copy(object_id, object_name);

Explained.

SQL>select * from table(dbms_xplan.display(null, null, 'basic'));
PLAN_TABLE_OUTPUT
----------------------------------------------------------
Plan hash value: 3262534006
----------------------------------------------------------
| Id  | Operation              | Name                |
----------------------------------------------------------
|   0 | CREATE INDEX STATEMENT |                     |
|   1 |  INDEX BUILD NON UNIQUE| ALL_OBJECTS_COPY_A1 |
|   2 |   SORT CREATE INDEX    |                     |
|   3 |    TABLE ACCESS FULL   | ALL_OBJECTS_COPY    |
----------------------------------------------------------

10 rows selected.

SQL>create index all_objects_copy_a1
  2  on all_objects_copy(object_id, object_name);

Index created.

SQL>explain plan for
  2  create index all_objects_copy_a2
  3  on all_objects_copy(object_name);

Explained.

SQL>select * from table(dbms_xplan.display(null, null, 'basic'));
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------
Plan hash value: 1214343037
--------------------------------------------------------
| Id  | Operation              | Name                 |
--------------------------------------------------------
|   0 | CREATE INDEX STATEMENT |                      |
|   1 |  INDEX BUILD NON UNIQUE| ALL_OBJECTS_COPY_A2  |
|   2 |   SORT CREATE INDEX    |                      |
|   3 |    INDEX FAST FULL SCAN| ALL_OBJECTS_COPY_A1  |
--------------------------------------------------------

10 rows selected.
```

בכל הדוגמאות בספר ביצענו את הפקודה Explain Plan, ואחר כך השתמשנו ב-API הוותיק dbms_xplan (קיים מגרסה 9.2), כדי להציג את התוצאות של הפקודה Explain. כעת נתעמק ביכולות של הפונקציה הכי מוכרת ב-API זה, הפונקציה display. מכיוון שלפונקציה יש פרמטרים עם ערכי ברירות מחדל, ניתן לקרוא לה מבלי לציין פרמטר כלשהו. לפני שנדון ביכולת עיצוב התוצאה של הפונקציה, נכיר את כל הפרמטרים שלה:

טבלה א.1:

שם פרמטר	הסבר	ברירת מחדל
table_name	שם הטבלה שבה נשמר המידע על תוכנית הגישה	PLAN_TABLE
statement_id	מזהה מחרוזתי של הבקשה לתוכנית גישה	NULL
format	עיצוב הפלט של תוכנית הגישה	TYPICAL
filter_preds	אפשרות לסנן רשומות מתוך הטבלה שבה נשמר המידע על תוכנית הגישה	NULL

פרמטר הפורמט שבו נעסוק כעת מורכב ממילות מפתח מופרדות על ידי רווח. כל מילת מפתח מוסיפה עוד מידע לתצוגה, ניתן להקדים מינוס ("-") לפני כל מילת מפתח כדי "להעלים" חלקים בתצוגה. יש מילות מפתח שכוללות מילות מפתח אחרות. הטבלה הבאה מציגה את כל מילות המפתח (בגרסה 11.2) ותפקידיהן:

טבלה א.2:

מילת מפתח	הסבר קצר	תוצג בשימוש ב:
Rows	מספר השורות שאמורות לחזור בכל שלב	All, Typical
Bytes	מספר הבתים שאמורים לחזור בכל שלב	All, Typical
Cost	תחזית האופטימייזר ל-"מחיר" של כל כל שלב	All, Typical

מילת מפתח	הסבר קצר	תוצג בשימוש ב:
Partition	מידע על פעולות במחיצות	All
Parallel	מידע על פעולות המבוצעות במקביל	All
Predicate	תנאים שהופעלו בשאילתה	All, Typical
Projection	מניפולציות על העמודות שנשלפו	All
Alias	כינויים לאובייקטים בשליפה, כולל אובייקטים שנוצרו במזמן השליפה	All
Remote	פעולות שבוצעו בבסיס נתונים מרוחק	All, Typical
Note	הודעות	All

הדוגמאות הבאות מראות את השתנות ההצגה של תוכנית הגישה. במהלך ההדגמה נראה גם יכולות שונות של האופטימייזר, ובזמן ההצגה נתעכב עליהן בקצרה.

קוד א-2:

```
SQL>explain plan for
  2   select count(*) from all_objects_copy_par
  3   where object_type = 'RULE';

Explained.

SQL>select * from table
  2  (dbms_xplan.display(null, null, 'all -cost -bytes -rows'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
Plan hash value: 524930559
-------------------------------------------------------------------------------------
| Id  | Operation              | Name                | Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                     | 00:00:01 |       |       |
|   1 |  SORT AGGREGATE        |                     |          |       |       |
|   2 |   PARTITION LIST SINGLE|                     | 00:00:01 |   KEY |   KEY |
|   3 |    TABLE ACCESS FULL   | ALL_OBJECT_COPY_PAR | 00:00:01 |    36 |    36 |
-------------------------------------------------------------------------------------
 Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$1
   3 - SEL$1 / ALL_OBJECT_COPY_PAR@SEL$1

Column Projection Information (identified by operation id):
-----------------------------------------------------------
    1 - (#keys=0) COUNT(*)[22]
```

ב'

מה אסור (לדעתנו) לעשות ביישום שמתחבר לאורקל

"החוכמה היא בתו של הניסיון"[1]

במהלך הספר, הצגנו בעיות שונות ומגוונות שניתן לפתור בקלות באמצעות SQL. שימוש נכון בשפה יחסוך מקרים בהם ננסה להרחיב את השפה, כמו למשל באמצעות קוד מעשי ידינו ב-PL/SQL, במקום שבו צריך להתעקש ולגלות כיצד ניתן לבצע זאת באמצעות היכולות המובנות ב-SQL. שימוש מוטעה ביכולות האלו, או טעויות שיקול דעת נפוצות, נדגים כעת. אלו עלולות לגרום לבעיות ביצועים, בעיות אבטחה ולטעויות לוגיות שונות.

הנספח בנוי מסעיפים קצרים שלא בהכרח שלובים זה בזה. מומלץ בחום לעיין בנספח הזה, ולו גם ברפרוף מהיר.

אי-שימוש ב-Bind במערכת OLTP

זהו נושא שחוק, שכבר דנו בו רבות, אך למרות זאת, עדיין רואים את הטעות הזאת. בסעיף קצר זה, נדגים דרכים לתקן יישומים שאינם משתמשים ב-bind, מבלי לתקן את הקוד, ונדגיש שהדבר הנכון ביותר הוא לתקן את הקוד. לטובת מי שלא מכיר כלל את הנושא, להלן הסבר קצר.

[1] לאונרדו דה וינצ'י, 15 באפריל 1452 - 2 במאי 1519.

כאשר אורקל מקבל פקודת SQL בפעם הראשונה, הוא מבצע את הפעילויות הבאות:

1. בדיקת תחביר. בבדיקה זו מוודאים שכל מילות המפתח נכונות, יש סוגריים נפתחים ונסגרים בהתאמה, ועוד.
2. בדיקת אובייקטים. בבדיקה זו מוודאים שכל האובייקטים שפונים אליהם בפקודה קיימים, ויש לנו גם הרשאה לפנות אליהם.
3. בדיקה אם הפקודה נמצאת כבר ב- shared pool.
4. בניית תוכנית גישה אופטימלית.
5. בניית התוכנית בפועל ואיחסונה בבסיס הנתונים לשימוש עתידי.
6. ביצוע התוכנית שנבנתה. אם הפקודה היא שליפה, מחזירים את התוצאות ללקוח, אחרת הפקודה מתבצעת ומחזירה קוד הצלחה או כישלון.

כאשר אורקל יקבל את הפקודה פעם נוספת, ניתן יהיה, בתנאים מסוימים, לדלג על השלבים זוללי המשאבים 4 ו-5. בספרות, השם הכולל לשלבים 1 עד 3 הוא soft parse; כאשר שלבים 4 ו-5 מתבצעים בנוסף לשלבים הקודמים להם, קוראים לתהליך כולו hard parse. מערכות OLTP, מתאפיינות בפקודות מהירות, החוזרות על עצמן בשינויים של ערכים בלבד. השאיפה היא לבצע hard parse פעם אחת בלבד, ולאחר מכן לבצע בכל שליחה נוספת של הפקודה אך ורק soft parse. ההנחה היא שתוכניות הגישה לפקודות כאלו אינן משתנות ואין הצדקה לבזבז משאבים על אופטימיזציה מחודשת.

כדי לממש את הרעיון הזה, אורקל מאפשר לנו להשתמש במנגנון Bind. בהתאם לשפה שבה אנחנו מפתחים, ניתן להשתמש בסימון מיוחד המחליף את הערכים שברצוננו להעביר לפקודה. בזמן ביצוע הפקודה נעביר לאורקל את הערכים שאנחנו רוצים בהם בפועל, ואורקל יחליף את הסימון הזה בערכים שלנו. המימוש המהיר והאינטואיטיבי ביותר של Bind הוא בשפת PL/SQL. רוב הקוד שנכתוב בשפה הזו, יהנה באופן אוטומטי ממנגנון Bind. למעשה, נצטרך להתאמץ כדי למנוע מאורקל שימוש ב-Bind כאשר כותבים ב-PL/SQL.

הנה דוגמה קצרה לשימוש ב-PL/SQL, לאחר הקוד אנחנו נריץ שאילתה על אחד מהמבטים (view) הדינמיים, ונראה כמה פעמים בפועל התבצע parse לשאילתה שלנו. שימו לב לזמן שנדרש לנו לבצע מאה אלף פקודות:

קוד ב-1:

```
SQL>set timing on
SQL>declare
  2  ll_ret number;
  3  begin
```

```
  4   for i in 1..100000 loop
  5      select 1 as search_for_me
  6      into ll_ret
  7      from dual where i = i;
  8   end loop;
  9  end;
 10  /

PL/SQL procedure successfully completed.

Elapsed: 00:00:03.42

SQL>select fetches, executions, parse_calls from v$sqlarea
  2  where sql_text like '%SEARCH_FOR_ME%';

   FETCHES EXECUTIONS PARSE_CALLS
---------- ---------- -----------
    100000     100000           1
         1          1           1
```

סיימנו. קראנו לשליפה שלנו מאה אלף פעמים, ועשינו parse פעם אחת בלבד. כפי שנאמר, כדי למנוע שימוש ב-Bind ב-PL/SQL נצטרך להתאמץ. המאמץ יבוא לידי ביטוי בכתיבת SQL דינמי: שימוש ב-API הוותיק dbms_sql או על ידי הפקודה execute immediate או פתיחת ref cursor והחזרתו ליישום עם מחרוזת של select.

מייד לאחר בלוק הפקודות שנדגים, נבצע שוב שליפה מה-view הדינמי v$sqlarea. הפעם נבצע parse מלא, אשר כולל את כל השלבים (hard parse). שימו לב להבדלים במספר פעולות ה-parse. לולא היינו משתמשים בשליפה אגרגטיבית, היינו מקבלים מה-view את כל הרשומות שעדיין נותרו בזיכרון[2]. בנוסף, יש גידול חד בזמן הביצוע, מזמן של כ-4 שניות עברנו לזמן של 70 שניות:

קוד ב-2:

```
SQL>declare
  2  ll_ret number;
  3  begin
  4    for i in 1..100000 loop
  5       execute immediate
  6       'select 1 as search_for_me from dual
  7       where ' || to_char(i) ||
```

[2] אורקל שומר את פקודות SQL ופקודות PL/SQL בזיכרון מטמון המתבסס על מנגנון Least Recently Used. פירושו, שהסיכוי שפקודה תישמר בזיכרון מותנה בכמות הפעמים שישתמשו בה. כלומר, בהתייחס לדוגמה שלנו, יש סיכוי שלא נקבל את הרשומות, מכיוון שאיש לא השתמש בשליפה שלנו שוב, ולכן היא פונתה מזיכרון המטמון כדי שיהיה בו מקום לפקודות חדשות, "פופלריות" יותר.

```
 8       ' = ' || to_char(i)
 9       into ll_ret;
10    end loop;
11  end;
12  /

PL/SQL procedure successfully completed.

Elapsed: 00:01:11.26

SQL>select sum(fetches), sum(executions), count(*) from v$sqlarea
  2  where sql_text like '%search_for_me%';

SUM(FETCHES) SUM(EXECUTIONS)   COUNT(*)
------------ --------------- ----------
        6075            6077       6077
```

העמודה count מראה ש"רק" 6,077 פעמים בוצע hard parse גוזל המשאבים, בניגוד לשליפה עם bind שביצעה hard parse פעם אחת בלבד. הבזבזנות בקטע הקוד האחרון הייתה גדולה ביותר. למעשה, מכיוון שלא השתמשנו ב-bind, אורקל לא היה מסוגל להחזיק את כל מאה אלף השליפות שלנו בזיכרון והוא "דחק" מתוכו את השאילתות הראשונות שבוצעו.

כאשר כותבים מערכת חדשה בסביבת OLTP, או מוסיפים יכולות למערכת ישנה, יש לשאוף לשימוש מרבי ב-Bind. התעלמות מההמלצה הזו תגרום לבעיות אלו:

1. בעיות ביצועים בגלל ביצוע מיותר של hard parse. השאיפה שלנו היא לבנות מערכות שהן מהירות וברות גדילה תמיד. עודף ביצוע hard parse הוא גורם מגביל מאוד.
2. בעיות אבטחה בגלל חשיפה ל- SQL Injection.

ראוי לציין כי גם בכתיבת SQL דינמי בשפת PL/SQL ניתן להשתמש ב-Bind. בדוגמה שלנו, כדי להראות את המצב הגרוע ביותר לא ביצענו זאת. מה ניתן לעשות במידה ואין לנו גישה או יכולת לשנות את הקוד? במרוצת הגרסאות של אורקל התווספו והבשילו פתרונות לאתגר הזה, והדבר מוכיח את חשיבות הנושא:

1. גרסה 8.1.6 (בשלהי המילניום הקודם) הציגה את הפרמטר cursor_sharing המאפשר לבצע Bind אוטומטי. בגרסה זו, לפרמטר היו שני ערכים: exact, שהוא ערך ברירת המחדל ו-force, שממיר באופן אוטומטי את כל הקבועים בפקודה ל-Bind.
2. גרסה 9.1 הוסיפה לפרמטר cursor_sharing אפשרות לביצוע החלטה מושכלת על ידי בחינה אם יש להמיר את הקבועים בפקודה ל-bind. אם יש חשש שהמרה כזו תשנה את תוכנית הגישה, לא תתבצע המרה.

ג'

מדריך SQL בסיסי

"הדבר החשוב היותר בשפת תכנות הוא השם שלה. שום שפת תכנות לא תצליח, אם אין לה שם טוב. לאחרונה המצאתי שם מצוין ואני מחפש עכשיו שפת תכנות שתתאים לו."[1]

בנספח זה נציג את יסודות השפה **SQL** – Structured Query Language, ונתמקד בעיקר במשפט Select. בפרק זה נציג זאת באופן בסיסי, ובשאר פרקי הספר נדון בהרחבות רבות העוצמה שנוספו למשפט זה.

נספח זה נועד ללמד SQL בניב המורחב של אורקל, ולכן בחרנו להציג את הדוגמאות על ידי **טבלאות מילון הנתונים**, להלן **dict**[2], של אורקל. אנחנו מקווים להרוויח בכך שני דברים:

1. בדרך זו תוכל לתרגל את הדוגמאות בנספח זה מהר ככל האפשר מבלי להריץ script כלשהו, בהנחה שיש לך התקנה של Oracle client.
2. השוק מוצף בסביבות עבודה ויזואליות שיודעות לחלץ את הנתונים הללו מתוך הטבלאות ולהציג אותם בצורה נאה. למרות זאת, אנחנו מאמינים שתמיד טוב לפתח את היכולת לעשות שליפות כאלו בעצמך.

כיום מעטים היישומים שמאחוריהם לא נמצא בסיס נתונים כלשהו, וביניהם שולט בסיס הנתונים של אורקל הידוע ביכולותיו ותכונותיו. בנספח זה נראה דוגמאות לחדשנות וליצירתיות בפיתוח שפת SQL מצידה של אורקל, שהקדימה במקרים רבים את מתחריה. תכונות מגוונות של השפה אינן מבטאות את הכל.

---

[1] Donald Ervin Knuth, נולד ב־10 בינואר 1938.

[2] נדגיש: טבלאות המערכת הן למעשה תצוגות view, או מבטים, של הטבלאות הפנימיות של Oracle.

לאלו יש להוסיף תכונות חשובות שבהן מצטיין אורקל: שרידות[3], התאוששות וגיבויים[4] וכמובן - ביצועים[5].

ניתן לחלק את שפת SQL באורקל לחמישה חלקים עיקריים:

DDL (Data Definition Language) - באמצעות ארבע הפקודות create, alter, drop ו-truncate ניתן ליצור אובייקטים בבסיס הנתונים, לשנות אותם בשעת הצורך, להשמיד אותם, ובמקרה של טבלאות - לרוקן אותן בצורה מהירה מאוד. פקודות DDL נוספות מאפשרות לנהל הרשאות גישה לאובייקטים, לאסוף נתונים סטטיסטיים כדי לשפר ולייעל את עבודת בסיס נתונים, לשנות שמות אובייקטים ועוד...

DML (Data Manipulation Language) - פקודות שמבצעות פעולות או מניפולציה על נתונים (על פי האקדמיה העברית: DML היא שְׂפַת טִפּוּל נְתוּנִים, מלשון טיפול). לדוגמה, לאחר יצירת טבלה על ידי פקודת DDL נוכל: להזין אותה בנתונים על ידי הפקודה insert; לעדכן את הנתונים על ידי הפקודה update; למחוק אותם על ידי הפקודה delete; לעדכן או ליצור רשומות "באותה נשימה" באופן מהיר ויעיל על ידי הפקודה merge; ולבסוף נזכיר את הפקודה הנפוצה שבה אנו דנים בהרחבה בספר זה: הפקודה Select, אשר מאפשרת לשלוף את הנתונים ולבצע בהם עיבודים מתוחכמים. המשפט האחרון דורש הבהרה. בשלב זה נסתפק בהערה הבאה: מה שאתה תוכל לעשות באמצעות פקודת Select אחת, יקח לאנשים אחרים, בהרבה מקרים, כתיבת תוכנית שלמה, על כל התקורה הנדרשת לכך.

פקודות לניהול טרנזקציות (תנועות) - פקודות אלו עוסקות בשמירת הנתונים לאחר השינויים שבוצעו בבסיס הנתונים באמצעות פקודות DML. הפקודה commit מורה לבסיס הנתונים לשמור באופן קבוע בבסיס הנתונים את השינויים שנעשו בו. כדי למחוק את הנתונים, נצטרך לבצע באופן מפורש פעולה לשינוי שלהם. אחת הסיבות ש-Oracle הוא בסיס הנתונים הטוב ביותר בשוק, נובעת מאמינות שמירת הנתונים שלו. כאשר Oracle מותקן כראוי, גם אם מייד לאחר ביצוע commit השרת נפגע מסיבה כלשהי, כגון תקלת חשמל או פגיעה פיזית, אף נתון לא ילך לאיבוד, וזה בדוק. הפקודה ההפוכה ל-commit היא הפקודה rollback אשר אומרת לבסיס הנתונים: "בטל מה שנעשה, טעיתי, התחרטתי", או "גלגל לאחור" את השינויים שבוצעו. לדוגמה, לאחר פעולת delete לרשומה כלשהי,

---

[3] בהתקנה נכונה של Oracle, תוך שימוש נכון בכל התכונות הרלוונטיות, גם אם האתר המרכזי יקרוס וייהרס לחלוטין, אתר הגיבוי (או האתרים) עדיין יתפקדו.

[4] באמצעות שימוש בכלי הגיבוי והשחזור המוביל של Oracle בשם Recovery Manger, ניתן לשחזר נתונים שגויים גם ברמת הבלוק באופן קל ומהיר, שלא לדבר על טכנולוגיות Flashback השונות.

[5] התוכנה Oracle מככבת באופן קבוע במבחני ביצועים משווים, לעתים היא מובילה ותמיד היא נמצאת במקום מכובד בין המקומות הראשונים.

ניתן לשחזר אותה לטבלה על ידי rollback. כך גם לאחר הוספת רשומה לטבלה, הפעלת rollback תמחק אותה. שלוש פקודות נוספות שקשורות לתנועות ואותן נסקור בקצרה בסוף הנספח.

פקודות session - אוסף פקודות המשפיע על ה-session הנוכחי. המונח session מייצג "קשר", "שיח" או "מושב" בין התחנה או עמדת העבודה לבין השרת[6]. פקודות אלו עשויות להשפיע על הדרך שבה מוצגים נתונים למשתמש באותו session.

פקודות מערכת - פקודות המיועדות לשנות הגדרות מהותיות בבסיס הנתונים, כגון שינוי משתני אתחול, טיפול בקבצים ועוד. בדרך כלל, מי שמריץ פקודות כאלו הוא מנהל בסיס הנתונים, ה-DBA (DataBase Administrator)[7].

פקודות משובצות (embedded) - פקודות אלו מייצגות פקודות בשפות תכנות שונות כמו למשל C, C++, C#, Cobol ועוד. אורקל מספקת תוכניות מיוחדות (precompiler) המתרגמת את קריאות SQL המשובצות בקובץ המקור של התוכנית לקריאות ייעודיות ל-API של אורקל. יכולות מסוימות ניתן לממש רק באמצעות פקודות משובצות, אך השימוש שלהן אינו נפוץ ונעשה רק כאשר אין פתרון סביר לבעיה על ידי היכולות המובנות של אורקל[8].

שפות אלו הן שפות לקוח, client, והכוונה לכך שהתוכנית שנכתבות תפעלנה בדרך כלל במחשב של המשתמש או בשרת ייעודי ליישומים ולא בשרת בסיס הנתונים. להבדיל משפות אלו, ההרחבה הפרוצדורלית PL/SQL[9] של אורקל לשפת SQL מורצת בשרת בסיס הנתונים עצמו. יש לכך יתרונות רבים. במהלך הספר נציג בדוגמאות השונות גם תוכניות בשפת PL/SQL[10].

---

[6] המונח העברי לא נקלט, והכל אומרים session וחבל. בתעשייה מקובל לעתים להשתמש בביטויים באנגלית, גם כאשר יש מונח עברי יפה וקליט, כמו למשל "נתונים" ולא "data". הבעיה מתעוררת במיוחד כשרוצים להציג את המונח ברבים, בהטיה או בסמיכות ואז מקבלים שפה עילגת, שאינה עברית, ובוודאי לא אנגלית, כמו לדוגמה session-ים.

[7] חשוב מאוד להבהיר שמנהל בסיס הנתונים (DBA) אחראי לבסיס הנתונים, אבל הוא אינו הבעלים של הנתונים.

[8] יכולת כזאת של precompiler היא למשל, scrollable cursor, יכולת שנשמעת ייחודית במבט ראשון אבל בפועל ניתן לממש יכולות דומות מאוד גם בדרכים אחרות.

[9] PL/SQL - Procedural Language/Structured Query Language.

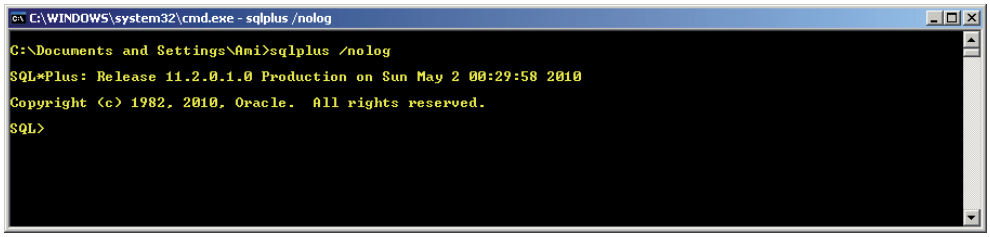
[10] בניגוד ליצרני מערכות ניהול של בסיסי נתונים טבלאיים (RDBMS) אחרים המציעים שפות script לטיפול בנתונים, PL/SQL היא שפה עשירה במיוחד. בתעשייה השמרנית של ניהול בסיסי נתונים, טוב לדעת שיש לה וותק, מאז שנת 1992 (גרסה 7).

בכל ההדגמות בספר זה נשתמש בכלי שורת הפקודה הוותיק של אורקל, Sql*Plus. בחרנו בו מכיוון שהוא נמצא בדרך כלל בכל התקנת client של אורקל. הכלי קצת מסורבל, אבל לאחר שמתרגלים להשתמש בו, רוב הקשיים נעלמים. ראוי לציין שרוב קבצי הקוד המוצגים בספר נבדקו עם רבות מהגרסאות שמקובלות בזמן כתיבת הספר. כאשר היה שוני בהפעלת הקוד בגרסאות שונות, טרחנו לציין זאת.

סקירה קצרה של Sql*Plus

כלי שורת הפקודה הנפוץ של אורקל יכול לסייע בפעולות רבות, החל בשליפות נתונים מבסיסי נתונים, כתיבת סקריפטים (שהם מעין תוכניות קטנות), הכנת דוחות פשוטים וגם פעולות ניהול כגון גיבוי, שחזור והעלאה והורדה של בסיס הנתונים. כפי שכבר נאמר, בשוק יש סביבות עבודה ויזואליות רבות, חלקן של חברת אורקל ורובן של חברות אחרות[11]. בחרנו ב- Sql*Plus בעיקר בגלל הפשטות שבו, אך הדבר אינו מחייב איש. הדוגמאות שמוצגות בספר זמינות לכל דורש באתר של הוד-עמי[12], ותוכל להריץ אותן בכלי התוכנה החביב עליך.

לטובת מי שלא מכיר Sql*Plus, נציג בקצרה מספר פקודות Sql*Plus חשובות, אשר שונות מפקודות SQL ומפקודות PL/SQL. צילום המסך להלן (והיחיד בספר) מראה את התוכנית sqlplus.exe במערכת ההפעלה Windows:



ניתן לראות ש- Sql*Plus הוא כלי שורת פקודה קלאסי. כאמור, מעתה לא נציג צילומי מסך נוספים ונסתפק במחוון שורת הפקודה בלבד, הפקודה או הפקודות שנריץ והתוצאות. לעתים נפרמט את תוצאות השליפה או השליפות לצורך קריאה נוחה יותר. כאשר אורך פקודה עולה על שורה אחת, Sql*Plus מציג גם את מספרי השורות בצד שמאל. כאשר נשתמש בפקודה שרלוונטית החל מגרסה מסוימת של אורקל, נראה במחוון הפקודה גם

---

[11] סביבות עבודה הוויזואליות המקובלות ביותר בישראל הן PL/SQL Developer של חברת Allround Automations ההולנדית, ו- Toad של חברת Quest Software האמריקאית.

[12] היכנסו לאתר http://www.hod-ami.co.il ובחרו בשם ספר זה. באתר הוד-עמי תמצאו גם ספר הדרכה בסיסי שפת SQL.

את מספר הגרסה. כאשר נדגים פקודה שהייתה קיימת בגרסה 10 ואילך של אורקל, נקפיד להשתמש במחוון הפקודה הסטנדרטי (<SQL). הספר מניח שהקורא יודע להתחבר לבסיס נתונים פעיל. לעתים נשנה את מחוון הפקודה, בעיקר בהדגמות המציגות שני session במקביל.

לפניכם מקבץ שימושי של פקודות Sql*Plus[13], וחשוב להדגיש שהן **אינן פקודות SQL**.

Describe (ניתן לרשום Desc), לאחריה שם אובייקט. משמשת לקבלת מידע בסיסי על האובייקט. נניח שאנחנו רוצים לדעת אילו עמודות מרכיבות את טבלת מילון הנתונים המציגה את כל המשתמשים בבסיס הנתונים, הטבלה all_users. נקליד desc ואת שם הטבלה:

קוד ג-1:

```
SQL>desc all_users

Name                 Null?    Type
 ------------------ -------- ------------
 USERNAME            NOT NULL VARCHAR2(30)
 USER_ID             NOT NULL NUMBER
 CREATED             NOT NULL DATE
```

List (ניתן לרשום L או l). מציגה את הפקודה האחרונה שהזנו ב-Sql*Plus. נניח שהרצנו פקודת SQL או בלוק של קוד בשפת PL/SQL וקיבלנו תשובה כלשהי. עכשיו נקליד list (או l) ונקבל את הפקודה האחרונה שהרצנו. שימו לב לכוכבית בשורה 3 (מודגשת), אשר אומרת שזוהי השורה הנוכחית ועליה ניתן להפעיל פקודות נוספות של Sql*Plus, שלא נדגים כאן, כגון עריכה של השורה הנוכחית ועוד).

קוד ג-2:

```
SQL>select count(*) from all_objects
  2  where object_type='TABLE'
  3  and trunc(last_ddl_time) = trunc(sysdate);

  COUNT(*)
----------
        19

SQL>l
  1  select count(*) from all_objects
  2  where object_type='TABLE'
  3* and trunc(last_ddl_time) = trunc(sysdate)
```

[13] הפקודות בשפות Sql*Plus, SQL ו- PL/SQL אינן רגישות לאותיות רישיות (גדולות) או רגילות (קטנות).

Run (ניתן לרשום R או r). הצגה של הפקודה האחרונה אשר הוזנה לזיכרון של Sql*Plus והרצה חוזרת שלה. פקודה דומה היא הלוכסן ("/") אשר מריצה גם כן את הפקודה האחרונה מבלי להראות לנו אותה. נשתמש שוב בקוד ג-2 שכבר הרצנו:

קוד ג-3:

```
SQL>r
  1  select count(*) from all_objects
  2  where object_type='TABLE'
  3* and trunc(last_ddl_time) = trunc(sysdate)

  COUNT(*)
----------
        19

SQL>/

  COUNT(*)
----------
        19
```

אם נסיים הקלדה של פקודת SQL בנקודה-פסיק (;) ונקיש Enter, היא תורץ מייד[14]:

קוד ג-4:

```
SQL>select sysdate from dual;

SYSDATE
---------
02-MAY-10
```

פקודת Sql*Plus נוספת היא edit או ed. כאשר יש פקודות בזיכרון של Sql*Plus, היא מפעילה עורך טקסט. במערכת ההפעלה Windows, העורך הוא עורך הטקסט הפשוט Notepad. במערכת ההפעלה Linux, העורך הוא תוכנת הטקסט הוותיקה vi.

בשפת Sql*Plus יש פקודות רבות נוספות, שלא נעסוק בהן במסגרת ספר זה.

סקירה מהירה של הפקודה select

כעת נעבור לביצוע פעולת שליפה שמתמקדת לשם הפשטות בהצגת שמות הטבלאות שבבעלות המשתמש הנוכחי ומתחילות באות A:

קוד ג-5:

```
SQL>select table_name from user_tables where table_name like 'A%';
```

[14] לעומת זאת, ב-PL/SQL התו נקודה-פסיק הוא חלק מובנה בשפה ומציין סיום משפט. כדי להריץ פקודות PL/SQL מתוך Sql*Plus, צריך להקליד לוכסן ("/") בסיום רצף הפקודות ולהקיש Enter.

ד'

מקרים ותגובות

"The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals. We cause accidents."[1]

הנספח הזה הוא ניסיון אמיץ לדחוס ניסיון רב שנים בזיהוי ובפתרון של בעיות במסגרת דפים מעטים. נתאר כל מקרה לגופו, ולאחר מכן נראה פתרונות אפשריים. נציג בתחילה שגיאות זמן ריצה, ולאחר מכן שגיאות הידור.

שגיאות ORA שונות

ORA-00001 - שחזור השגיאה:

קוד ד-1:

```
SQL>create table pk_uk_demo(col_pk number,
  2  col_uk number,
  3  constraint pk_uk_demo_pk primary key(col_pk),
  4  constraint pk_uk_demo_uk unique (col_uk));

Table created.

SQL>insert into pk_uk_demo(col_pk) values(1);
1 row created.
SQL>insert into pk_uk_demo(col_pk) values(1);

insert into pk_uk_demo(col_pk) values(1)
*
```

[1] Nathaniel Borenstein, 09/23/1957.

```
ERROR at line 1:
ORA-00001: unique constraint (AMI.PK_UK_DEMO_PK) violated

SQL>insert into pk_uk_demo(col_pk, col_uk) values(2, 1);

1 row created.

SQL>insert into pk_uk_demo(col_pk, col_uk) values(3, null);

1 row created.

SQL>insert into pk_uk_demo(col_pk, col_uk) values(4, 1);
insert into pk_uk_demo(col_pk, col_uk) values(4, 1)
*
ERROR at line 1:
ORA-00001: unique constraint (AMI.PK_UK_DEMO_UK) violated
```

הסבר: שגיאה זו מתרחשת כאשר מֵפֵרים אילוץ של primary key או של unique. שימו לב ש-constraint של unique יכול להכיל ערך null ביותר מרשומה אחת ואין זה נחשב כהפרה שלו, מכיוון ש-null אחד לעולם אינו זהה ל-null אחר. ראוי לציין כי זו שגיאה "טובה". כלומר, שהתכנון המוקדם של בסיס הנתונים מאפשר לו לשמור על נכונות הנתונים ומונע מצב שבו קוד שגוי יעוות אותם. קרוב לוודאי שמשהו בזרימה של התוכנית שלך גרם לזה. הבעיה מכה במלוא העוצמה כאשר ה-insert איננו insert בדיד, אלא insert as select, אשר מכניס לטבלה יותר מרשומה אחת. לנסות, למשל, להכניס מיליון רשומות לטבלה ולהיכשל בתהליך כולו אך ורק בגלל רשומה כפולה אחת (מרפי ידאג שהיא תמיד תהיה הרשומה האחרונה), זו בהחלט חוויה עצובה ומיותרת.

פתרון הבעיה: עם התקדמות הגרסאות של Oracle סופקו דרכים שונות לאתר, או להתעלם מרשומות כפולות או מרשומות שמפרות אילוצים אחרים מבלי לגרום לקריסת התהליך, כמו למשל האילוץ not null או הגדרת סוגי נתונים שאינם תואמים. נציג אותם על פי סדר הופעתם הכרונולוגי בגרסאות Oracle.

בעבר הרחוק, רחוק לפחות עד גרסה 7, ניתן היה לבצע פעולה זו: להוריד את המשפט constraint, להריץ שוב את קטע הקוד החשוד, ולאחר מכן באמצעות המשפט EXCEPTIONS INTO לבצע enable למשפט constraint שהופר. כתוצאה מכך, זהויות (rowid) של הרשומות הלא תקינות הצטברו בטבלה מיוחדת[2]. זו הייתה הדרך היחידה לפתרון הבעיה עד לגרסה 9.2. נדגים זאת לטובת ארגונים שמרנים, שמפעילים עדיין את Oracle בגרסה ישנה זו:

---

[2] טבלה זו נוצרת באמצעות הקוד (script) הבא: utlexcpt.sql. אם הטבלה שרוצים לתחקר בנויה כ- IOT (Index Organized Table)), צריכים להשתמש ב-utlexpt1.sql הסקריפטים האלה נמצאים בתיקיית הסקריפטים של admin.