ADO.NET
סדנת לימוד

פיתוח מסדי נתונים עם
C# ו- VB.NET

קרא בעמוד 12 על קבצי קוד המקור

עורך ראשי ומקצועי: **זהר עמיהוד**

יועץ מקצועי והתאמה ל- C#: **מאיר קלטר**

תרגום: **דביר כפרי**

עריכה לשונית ועיצוב: **טליה טופז**

עיצוב עטיפה: **שרון רז**

שמות מסחריים

שמות המוצרים והשירותים המוזכרים בספר הינם שמות מסחריים רשומים של החברות שלהם. הוצאת הוד-עמי והוצאות Pearson Education ו- Addison Wesley Professional עשו כמיטב יכולתן למסור מידע אודות השמות המסחריים המוזכרים בספר זה ולציין את שמות החברות, המוצרים והשירותים. שמות מסחריים רשומים (registered trademarks) המוזכרים בספר צוינו בהתאמה.

הודעה

ספר זה מיועד לתת מידע אודות מוצרים שונים. נעשו מאמצים רבים לגרום לכך שהספר יהיה שלם ואמין ככל שניתן, אך אין משתמעת מכך כל אחריות שהיא.

המידע ניתן "כמות שהוא" ("as is"). הוצאת הוד-עמי והוצאת Addison Wesley Longman אינן אחראיות כלפי יחיד או ארגון עבור כל אובדן או נזק אשר ייגרם, אם ייגרם, מהמידע שבספר זה, או מהתקליטור/דיסקט שעשוי להיות מצורף לו.

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

ADO.NET
סדנת לימוד

פיתוח מסדי נתונים עם
C# ו- VB.NET

Jeffrey P. McManus
Jackie Goldstein

ADDISON-WESLEY

הוצאת
הוד-עמי
ספרי מחשבים

Database Access with Visual Basic .NET

By **Jeffrey P. McManus, Jackie Goldstein**

Editor: **Z. Amihud**

Authorized translation from the English language edition, entitled Database Access with Visual Basic®.Net, 3rd Edition, by Jeffrey P. McManus and Jackie Goldstein, published by Pearson Education Inc, publishing as Eddison Wesley Professional, Copyright © 2003 by Pearson Education Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Hebrew language edition published by Hod-Ami Computer Book Ltd. Copyright © 2004.

©

כל הזכויות שמורות

הוצאת הוד-עמי לספרי מחשבים בע"מ

ת.ד. 6108 הרצליה 46160

טלפון : 09-9564716 פקס : 09-9571582

info@hod-ami.co.il

תרגום מורשה של ההוצאה המקורית באנגלית כמפורט למעלה, שהודפסה על ידי
SAMS Publishing, Copyright © 2004.

אין להשאיל ו/או לעשות שימוש מסחרי ו/או להעתיק, לשכפל, לצלם, לתרגם, להקליט, לשדר, לקלוט ו/או לאחסן במאגר מידע בכל דרך ו/או אמצעי מכני, דיגיטלי, אופטי, מגנטי ו/או אחר - בחלק כלשהו מן המידע ו/או התמונות ו/או האיורים ו/או כל תוכן אחר הכלולים ו/או שצורפו לספר זה, בין אם לשימוש פנימי או לשימוש מסחרי. כל שימוש החורג מציטוט קטעים קצרים במסגרת של ביקורת ספרותית אסור בהחלט, אלא ברשות מפורשת בכתב מהמוציא לאור.

הודפס בישראל 2004

All Rights Reserved
HOD-AMI Ltd.
P.O.B. 6108, Herzliya
ISRAEL, 2004

מסת"ב ISBN 965-361-348-0

תוכן עניינים מקוצר

הקדמה 11

פרק 1: תחילת עבודה עם SQL Server 2000 15

פרק 2: ADO.NET – Data Providers 79

פרק 3: ADO.NET – DataSet 131

פרק 4: ADO.NET – DataAdapter 179

פרק 5: ADO.NET – תכונות וטכניקות נוספות..... 223

אינדקס 257

תוכן העניינים

הקדמה 11

למי מיועד ספר זה? 11
כיצד ספר זה מאורגן? 12
סביבת התוכנה 12
קוד מקור 12
שמור על קשר 14
אודות המחברים 14
אודות היועץ המקצועי 14
תודות 14

פרק 1: תחילת עבודה עם SQL Server 2000 15

מה בפרק זה? 15
הגדרה והפעלה של Microsoft SQL Server 2000 18
בדיקת דרישות התקנה עבור SQL Server 2000 19
התקנת SQL Server 2000 20
הפעלה והפסקת SQL Service באמצעות SQL Service Manager 23
קביעת אופן ההפעלה של SQL Server 24
יסודות העבודה ב- SQL Server 2000 25
הפעלת SQL Server Enterprise Manager 25
יצירת מסד נתונים באמצעות SQL Server Manager 27
יצירת טבלאות ב- SQL Server Database 30
יצירת טבלאות ב- SQL Server באמצעות Enterprise Manager 31
יצירת עמודת זיהוי לזיהוי ייחודי של רשומות 33
יצירת מפתחות ראשיים בשיטות נוספות 35
הגדרת עמודה כמפתח ראשי 36
גישה למסד נתונים באמצעות SQL Query Analyzer 37
הצגת כל האובייקטים במסד הנתונים באמצעות sp_help 39
הפעלת מסד נתונים קיים 41
שליחת פקודות SQL ל- SQL Query Analyzer 42
בקרת גישה לנתונים באמצעות תצוגות מסדי נתונים 42
יצירת תצוגות ב- SQL Server Enterprise Manager 43
שימוש בתצוגות ביישום מסחרי 45
יצירת תצוגות באמצעות SQL Query Analyzer Batch 46

יצירה והפעלת שגרות מאוחסנות 47
יצירת שגרות מאוחסנות באמצעות SQL Server Enterprise Manager 48
הפעלת שגרה מאוחסנת מתוך SQL Enterprise Manager 49
יצירת שגרות מאוחסנות באמצעות SQL Query Analyzer 50
הצגת טקסט של שגרה מאוחסנת או של תצוגה קיימת 52
יצירת מזניקים 53
תסריט עסקי 1.1 : יצירת מזניק לחיפוש מקורב 54
ניהול משתמשים ואבטחה באמצעות SQL Server Enterprise Manager 57
יצירה ותחזוקה של התחברויות ומשתמשים 58
ניהול תפקידים באמצעות SQL Server Enterprise Manager 61
בדיקת תכונות אבטחה על ידי SQL Query Analyzer 63
החלת תכונות אבטחה באמצעות SQL Query Analyzer 64
בדיקה מי מחובר באמצעות sp_who 66
סיום תהליך באמצעות הפקודה kill 67
הסרת אובייקטים ממסד הנתונים 67
תסריט עסקי 1.2 : כתיבת תסריט SQL ליצירת מסד נתונים 68
סיכום 77
שאלות ותשובות 77

פרק 2: ADO.NET – Data Providers 79

מה בפרק זה? 79
סקירת ADO.NET 80
מוטיבציה ופילוסופיה 80
תמיכה ביישומים מבוזרים ובמודל תכנות מנותק 81
תמיכה מקיפה ב-XML 81
שילוב עם NET Framework. 82
"מראה ותחושה" של ADO 82
השוואת ADO.NET ל-ADO קלאסי (ADO 2.X) 82
אובייקטי ADO.NET ב- NET Framework. 83
ממשקי יישומים 84
סקירת אובייקטים של ספקי נתונים ב-NET. 84
SqlClient 84
OLEDB 85
ODBC 85
אובייקטים עיקריים 86
האובייקט Connection 87
האובייקט Command 92
שימוש באובייקט Command עם פרמטרים ושגרות מאוחסנות 95
הפעלת הפקודות 101
האובייקט DataReader 112

שימוש ברכיבים Connection ו-Command במהלך תכנון 116
אובייקטים נוספים של ספקי נתונים 119
תסריט עסקי 2.1: כתיבת שגרה לגניזת הזמנות ישנות בארכיון, לפי שנים 120
סיכום 128
שאלות ותשובות 129

פרק 3: ADO.NET – DataSet 131

מה בפרק זה? 131
יישומים ורכיבים של DataSet 132
מילוי נתונים ב-DataSet ושינוי שלהם 134
הגדרת סכמות של DataTable 135
הוספת נתונים ל-DataTable 141
גישה לנתונים מתוך האובייקט DataTable 144
חיפוש, סינון ומיון רשומות 148
עדכון DataSet 153
מחיקת רשומה 153
RowStates ו-Versions 154
שגיאות בעמודות ורשומות 156
קשרים בין טבלאות 158
אילוצי טבלאות 165
שימוש ברכיב DataSet 171
סיכום 177
שאלות ותשובות 178

פרק 4: ADO.NET – DataAdapter 179

מה בפרק זה? 179
מילוי DataSet ממקור נתונים 181
עדכון מקור הנתונים 190
הגדרת פקודות Update 193
שימוש באובייקט CommandBuilder 193
פקודות עדכון מפורשות (explicit) 199
הטמעת לוגיקה עסקית בפקודות עדכון 207
שימוש ברכיב העיצוב DataAdapter 209
תסריט עסקי 4.1: שילוב מספר טבלאות עם קשרים 215
סיכום 221
שאלות ותשובות 221

פרק 5: ADO.NET – תכונות וטכניקות נוספות..... 223
מה בפרק זה? 223
גילוי התנגשויות 224
מיפויי טבלאות ועמודות 229
טבלאות מדומות – DataViews..... 233
תסריט עסקי 5.1: הצגת נתונים משולבים ממקורות שונים..... 244
Strongly Typed DataSets 250
סיכום 255
שאלות ותשובות 256

אינדקס..... 257

הקדמה

המטרה של ספר זה היא להראות לך כיצד לפתח יישום המבוסס על מסד נתונים בעזרת Visual Basic.NET, #C ו-ADO.NET.

הערה

ישנם מספר שמות למוצר Visual Basic.NET והם: VB.NET או "סתם" VB.

ישנם מספר שמות למוצר #C והם: C Sharp, Visual C Sharp, vcsharp, CS, Visual C# וגם Visual C#.NET.

במרבית הדוגמאות, ההתייחסות היא ל- Microsoft SQL Server, אבל זה לא צריך להפריע לך אם אתה משתמש במסד נתונים אחר. במקום שיש שוני ביישום, תמצא הסבר מה לעשות.

ההנחה היא שאתה עובד עם SQL Server, אבל גם אם אתה עובד עם מסד נתונים מסוג Access או Oracle זה יספיק. הקדשנו פרק שלם וגדול לנושא SQL. גם אם אתה מכיר נושא זה, מומלץ לעבור במהירות על החלקים שבפרק, אולי תלמד משהו חדש! אם אתה מרגיש שאתה זקוק לפירוט נוסף בנושא SQL, מומלץ לעיין בספר **בסיסי נתונים טבלאיים ושפת SQL עקרונות עיצוב** ובספר **המדריך ל- SQL Server 2000**, שניהם ב**הוצאת הוד-עמי**.

הספר כולל עשרות דוגמאות, תרגילים וקוד מקור. הדרך לשליטה בטכנולוגיה המתקדמת ולהטמעת הרעיונות היא להריץ את הדוגמאות, לעשות את התרגילים ו"לשחק" עם קוד המקור. במהלך הפרק תמצא קטעי קוד קצרים להמחשת הרעיונות. כל רעיון ממומש בקוד Visual Basic.NET ובקוד #C, ובהמשך הפרק תמצא דוגמה עסקית המורכבת מכל אותן פיסות קוד שהוסברו במהלכו.

למי מיועד ספר זה?

הספר מיועד למי שיודע Visual Basic.NET ו/או למי שיודע #C. ספר זה לא ילמד אותך Visual Basic.NET – לשם כך עליך להיעזר בספר **Visual Basic.NET סדנת לימוד** בהוצאת **הוד-עמי**. ספר זה לא ילמד אותך #C – לשם כך עליך להיעזר בספר **#C סדנת לימוד** בהוצאת **הוד-עמי**. מעבר לכך, ההנחה היא שאתה משתמש בסביבת העבודה VS.NET, ולכן לא תמצא בספר זה הסבר כיצד לפתוח פרויקט.

מרבית הדוגמאות הם פרויקטים מסוג Windows Application הנקראים גם Windows Forms. הסיבה היא שמרבית המתכנתים ב- Visual Basic.NET או #C עובדים ומכירים סוג פרויקט זה. גישה זו תאפשר לך להתרכז במסדי הנתונים. למי שמעוניין

לפתח Web Application או Web Services, מומלץ לעיין בספר **ASP.NET סדנת לימוד** בהוצאת **הוד-עמי**.

כיצד ספר זה מאורגן?

הספר מכיל 5 פרקים (בשונה מהספר המקורי):

פרק 1 – "מיישר קו" בנושא SQL Server 2000. מידע נוסף תמצא בספר **המדריך ל- SQL Server 2000** בהוצאת **הוד-עמי**.

פרקים 2-4 – הם לב ליבו של הספר ומכילים מידע מפורט ומעמיק אודות ADO.NET: אובייקטים, שיטות ותכונות.

פרק 5 – מכיל שיטות מתקדמות לעבודה עם ADO.NET.

סביבת התוכנה

ההנחה היא שהתקנת את אחת הגרסאות של Visual Studio.NET (Enterprise Developer, Enterprise Architect, Professional או Academic. על ההבדלים בין הגרסאות השונות ביחס לנושא מסדי נתונים קרא באתר Microsoft בכתובת: msdn.microsoft.com/vstudio).

בפרק 1 מתוארת התקנה של SQL Server 2000 בפירוט רב. אנא, צור העתק של מסד הנתונים pubs, מכיון שמרבית הדוגמאות בספר זה מתייחסות אליו.

מרבית הדוגמאות העסקיות שבספר זה משתמשות במסד נתונים בשם Novelty. הנחיות מפורטות לבניית מסד נתונים Novelty במערכת SQL Server 2000 ניתנות לאורך הספר. בכל פעם שיופיע הביטוי 'מסד נתונים' במהלך הספר, הכוונה היא למסד נתונים במערכת SQL Server 2000.

קוד מקור

כל קוד המקור שבספר נמצא באתר הוד-עמי בכתובת:

www.hod-ami.co.il/59389.html

בין קטעי קוד המקור של פרק 1 תמצא גם את קובץ NoveltyDB.sql שבעזרתו תוכל לבנות את מסד הנתונים. פעל לפי ההוראות הבאות:

1. פתח את **SQL Server Query Analyzer** ובצע נוהל כניסה לאחד השרתים.
2. בחר בתפריט **File** באפשרות **Open** ופתח את הקובץ **NoveltyDB.sql**.

3. הקוד מופיע על המסך.

4. בסרגל הכלים לחץ על **Excute Query**.

התסריט יבוצע ויצור את מסד הנתונים.

בקוד המקור של פרק 1 תוכל למצוא תסריטים נוספים. למשל, הקובץ OrdersData.sql מכיל תסריט המכניס נתונים למסד הנתונים. אם אתה רוצה להפעיל תסריט, פעל לפי ההוראות שלעיל.

אם חסר לך רכיב כמו ODBC או מנהל התקנים (Data Provider) אחר, תוכל להוריד אותו מאתר Microsoft בכתובת msdn.microsoft.com/downloads.

הערה

הפיתוח של ספק נתונים עבור ODBC השתהה מעט מאחורי חלקים אחרים של NET Framework. או Visual Studio.NET. לכן, הוא לא נכלל במהדורת Visual Studio.NET הראשונה, אך ניתן להוריד אותו מאתר האינטרנט של מיקרוסופט. בדוק מדי פעם והורד ספקי נתונים נוספים של NET. אשר יהיו זמינים.

ספר זה נבדק בעזרת Visual Studio.NET 2003. בגירסה זו ספק הנתונים עבור ODBC כלול במרחב השמות System.Data.Odbc. במידה והינך משתמש בגירסה קודמת, יהיה עליך לשנות אותו ל- Microsoft.Data.Odbc בכל מקום בספר שמצוין System.Data.Odbc.

אם הינך עובד עם Visual Studio.NET 2002, ייתכן שעורך Visual Studio לא יזהה את מרחב השמות **Microsoft.Data.Odbc**, מכיון שמדובר בתוספת למוצר הבסיסי.

במקרה זה, בצע את הפעולות שלהלן:

1. כדי להתקין את הספק במחשב, הורד את קובץ ההתקנה של ספק הנתונים ODBC מאתר מיקרוסופט באינטרנט ופעל על פי ההנחיות המוצגות בו.
2. ב- Solution Explorer, לחץ לחיצה ימנית על הרכיב **References** של הפרויקט DataProviderObjects.
3. בחר **Add Reference** מהתפריט הנפתח.
4. בכרטיסיה NET. של תיבת הדו-שיח Add Reference, גלול את רשימת הרכיבים עד שתגיע ל- Microsoft.Data.Odbc.dll.
5. לחץ לחיצה כפולה על פריט הרשימה **Microsoft.Data.Odbc.dll**, כדי להוסיף אותו לרשימה Selected Components שבתחתית תיבת הדו-שיח.
6. לחץ **OK** כדי לסגור את תיבת הדו-שיח.

שמור על קשר

אם יש לך הערות, שאלות או רעיונות הנוגעים לספר זה של ההוצאה, אנא שלח אותם ישירות למחבר: ג'קי גולדשטיין jackie@renaissance.co.il (תוכל לכתוב בעברית או באנגלית, כרצונך). כמו כן, אתה מוזמן לבקר באתר המחבר בכתובת **www.renaissance.co.il**.

תוכל גם לשלוח דואר אלקטרוני לכתובת **support@hod-ami.co.il** או דואר רגיל לכתובת: הוצאת הוד-עמי, ת.ד. 6108, הרצליה 46160.

אודות המחברים

ג'קי גולדשטיין (Jackie Goldstein) מייסד חברת **רנסאנס מערכות מחשוב בע"מ**, המתמחה בייעוץ ובפיתוח בטכנולוגיות ובכלים של Microsoft. לג'קי ניסיון של למעלה מ- 20 שנה בפיתוח בארה"ב ובישראל. ג'קי הוא ה- Microsoft Regional Director היחיד הנושא תואר זה בישראל. פרטים נוספים באתר **www.renaissance.co.il**.

ג'פרי פ. מקמנוס (Jeffrey P. McManus) מפתח יישומים מקוונים בסביבת שרת/לקוח ואינטרנט בכלי Microsoft. עד כה הוא כתב 4 ספרים בנושאי מסדי נתונים. ג'פרי הינו מרצה מבוקש בכנסים וימי עיון בטכנולוגיות Microsoft.

אודות היועץ המקצועי

מאיר קלטר מתמחה בפיתוח אתרים עתירי טכנולוגיות וסביבות פיתוח: Java, JSP, ASP ו- Visual Studio.NET. עוסק בהדרכת ובהנחיית פרויקטי גמר לסטודנטים במדעי המחשב.

פרק 1

תחילת עבודה עם SQL Server 2000

מה בפרק זה?

- הגדרה והפעלה של Microsoft SQL Server 2000
- יסודות העבודה ב- SQL Server 2000

בעבר, רבים ממתכנתי Visual Basic נחשפו לתכנות מסד נתונים באמצעות מנוע מסד הנתונים Jet שהיה בשימוש Visual Basic ו- Microsoft Access. לאחר שנפחי מסדי נתונים גדלו מעבר לכמה מאות רשומות או מספר קטן של משתמשים, המתכנתים החלו להיתקל במגבלות. התנגשות בין משתמשים בעת גישה לנתונים, ביצועים גרועים, חוסר בתכונות נתונים מתקדמות וניהול שרתים לקוי, גרמו למתכנתים רבים לפנות לארכיטקטורה חילופית אשר פתרה את בעיותיהם במסדי הנתונים הגדולים. ארכיטקטורה זו נקראת "מחשוב שרת-לקוח" (client-server programming או distributed programming).

אין להחליף בין המושג "מחשוב שרת-לקוח" לבין "מחשוב מרובה-משתמשים" (multiuser computing), אשר נתמך על ידי Jet. בארכיטקטורה מרובת משתמשים, מספר משתמשים חולקים את אותם הנתונים באמצעות הרשת. כלומר, קובץ או קבצי מסד הנתונים נמצאים בשרת מרכזי אשר לכל משתמשי תחנות העבודה יש גישה אליו. השוני הוא בכך שבארכיטקטורה כגון Jet, שאינה ארכיטקטורת שרת-לקוח, כל העיבוד מתבצע בתחנות העבודה של הלקוחות. כלומר, כדי לאחזר שורה יחידה על פי הצהרת SELECT של SQL מטבלה המכילה 50,000 שורות, יש להעביר תחילה את כל השורות (או לפחות את האינדקסים שלהם) לתחנת העבודה של הלקוח. בצד השני של הרשת אין התקן חכם שמסוגל לקבל דרישות ולהחזיר להן נתונים.

לעומת זאת, לארכיטקטורת שרת-לקוח יש מעין **קצה חכם** – אין מדובר במתכנת אשר יושב בכסאו במשך היום, אלא בתוכנה אשר אחראית לאחזור והטמנת נתונים, יצירת חיץ בין משתמשים שונים וגם לטיפול באבטחה. תוכנה זו, כמו Microsoft SQL Server, מקבלת דרישות מתחנות עבודה של לקוחות, מפעילה דרישות אלו במחשב השרת ומחזירה את התוצאות בלבד אל מחשב הלקוח. לפיכך, במקרה של צורך לאחזר שורה יחידה מטבלה בת 50,000 שורות נתונים, הצהרת SELECT מועברת לשרת, תוכנת מסד הנתונים של השרת מבצעת את ההצהרה ומחזירה ללקוח את השורה האחת הרצויה. החיסכון בתעבורת הרשת ברור. יתרון נוסף שקשור לביצועים הוא שמחשבי שרת לרוב חזקים יותר (כלומר, מותקנים בהם מעבדים מהירים יותר וכמות גדולה יותר של זיכרון) מאשר במחשבי לקוח, כך שזמן ביצוע ההצהרה ואחזור הנתונים בפועל מתקצר.

אם אתה משתמש ב- Visual Basic.NET (VB.NET), הרי ש- Microsoft SQL Server אמורה להיות הבחירה הברורה שלך לתוכנת קצה של מסד הנתונים. זו אינה רק מערכת מסד נתונים רבת עוצמה וקלה לשימוש, אלא שעותק של SQL Server קיים גם בכל מהדורה של VB.NET ושל Visual Studio.NET (VS.NET). נבחן אילו מהדורות של SQL Server כלולות במהדורות VB.NET ו-VS.NET, לאחר שנתאר את ההבדל בין מהדורות SQL Server השונות.

טיפ

אין להשתמש במסדי נתונים מסוג Jet (MDB), אלא ביישומים הפשוטים ביותר או יישומים מוגבלים בזיכרון. התוכנה SQL Server 2000 Desktop Engine (MSDE) מבטלת את הצורך להשתמש במסדי נתוני Jet במערכות אבטיפוס ו/או במערכות זולות. עם זאת, השימוש מלכתחילה במסד נתונים חופשי תואם SQL Server מונע את הצורך לבצע שינויים בשאילתות, בקוד או בעיצוב במהלך שדרוג המערכת.

בפרק זה נתמקד בתחילת העבודה ב- SQL Server 2000. בכוונתנו להציג בפניך מבוא מלהיב להגדרה ושימוש ב- SQL Server כדי להכין אותך לחומר ולדוגמאות שתמצא בהמשך הספר. אם אינך מכיר את SQL Server, החומר בפרק זה יספיק לך כדי להתחיל ולהתוודע לשרת מיוחד זה. אנו ממליצים לעיין בספר **המדריך ל- SQL Server 2000** בהוצאת **הוד-עמי**. אם אתה מכיר את SQL Server, עדיין מומלץ לקרוא פרק זה לשם ריענון הידע. ייתכן אפילו שתלמד דבר או שניים. להלן תרחיש טיפוסי:

נניח שאתה חבר בצוות פיתוח של מערכת מבוזרת או מערכת שרת-לקוח. לרשותך שרת מסד נתונים אשר פועל ב-95%. כלומר, אינו פועל כלל. מוטל עליך לבצע את עבודתך, אך רכיב השרת ביישום אינו מוכן לעבודה שוטפת.

יתרה מכך, נניח שלרשותך מתכנת אחד או שניים בלבד בצד השרת. תכנות שרת היא מיומנות נדירה, ולכן מצב זה קורה פעמים רבות בארגוני פיתוח של יישומי שרת-לקוח. לארגונים קשה ביותר לשכור ולהשאיר בתפקידם מתכנים מסוג זה, ולכן סביר שהם האנשים קצרי הרוח ביותר שתפגוש. לכן, קשה ביותר להשתמש בהם במקרה של תקלה. כמו כן, פעמים רבות מתכנתי שרת-לקוח אינם יכולים לסיים את עבודתם עד שמתכנתי השרת יתקנו את התקלה בשרת.

אם היית מעורב בפרויקט פיתוח מבוזר אשר כלל יותר משני מפתחים, אתה כבר מכיר את התרחיש. פתרון אחד הוא ליצור אבטיפוס (prototype) של יישום הלקוח, באמצעות שימוש בתבנית קבועה למקור נתוני Jet, וכאשר השרת מוכן, עליך לחבר את היישום אל השרת. תכנון מקור נתוני ODBC או שימוש בקישור נתוני OLE DB הן שתי דרכים קלות לעשות זאת. שימוש ב- ODBC או OLE DB מאפשר ליצור ולהשתמש בקלות באבטיפוס של מסד הנתונים ביישום שלך, ולעבור למסד הנתונים האמיתי כאשר תחפוץ בכך.

הכנסת שכבה אחת או יותר של פישוט בין הלקוח לבין השרת מונעת גם ממתכנתי לקוח מלהעיק על מתכנת השרת. המשמעות עבור מתכנת השרת הינה חשיפה של תצוגות או שגרות מאוחסנות אשר מספקות שירותי נתונים ללקוחות; המשמעות עבור מתכנתי VB.NET היא יצירת רכיבי קוד שעושים את אותה הפעולה. למידע אודות אסטרטגיות שימוש בשגרות מאוחסנות ותצוגות, ראה סעיפים "יצירה והפעלת שגרות מאוחסנות" ו"בקרת גישה לנתונים באמצעות תצוגות מסדי נתונים" בהמשך פרק זה.

הגדרה והפעלה של
Microsoft SQL Server 2000

אנו ממליצים לגלוש לאתר http://www.microsoft.com/sql ולחפש תחת הכותרת System Requirements.

הפעלת שרת מסד נתונים אמיתי שונה באופן משמעותי משיתוף מסד נתוני Microsoft Jet. עליך ללמוד שתי תפיסות חדשות ולדאוג לנושאים חדשים. עם זאת, קל בהרבה להגדיר ולתחזק את SQL Server 2000 מאשר גרסאות קודמות, במיוחד גירסה 6.5 וקודמותיה.

בסעיף זה נפרט תחילה את הדרישות המינימליות להגדרה והפעלה של SQL Server 2000. כאמור, לתוכנת השרת יש גרסאות שונות:

- SQL Server 2000 Standard Edition – תוכנת שרת מסד נתונים בסיסית, מתאימה לקבוצת עבודה או למחלקה.
- SQL Server 2000 Enterprise Edition – התוכנה מכילה את כל התכונות של Standard Edition ומציעה תכונות וביצועים נוספים הדרושים לתמיכה בארגונים גדולים, אתרי אינטרנט ויישומי data warehousing.
- SQL Server 2000 Personal Edition – מתאימה למשתמשים ניידים, אשר מנותקים פעמים רבות מהרשת שלהם, אך זקוקים ל- SQL Server כמחסן נתונים מקומי להפעלת יישומים עצמאיים במחשב לקוח עם SQL Server. שלא כמו מהדורות Standard ו-Enterprise, אשר דורשות גרסת שרת של Windows NT, Windows 2000 או Windows Server 2003, מהדורת Personal יכולה לפעול גם על Windows XP Home Edition, Windows XP Professional, Windows 2000 Professional, NT 4.0 Workstation ו- Windows 98/ME. גירסה זו מגבילה את ביצועי השרת כאשר מבוצעות יותר מחמש פקודות בו-זמנית.
- SQL Server 2000 Developer Edition – כוללת את כל התכונות של Enterprise Edition, אך מורשית לשימוש על ידי מפתחים בלבד, אשר מפתחים ובודקים יישומי SQL Server. מוצר זה אינו מורשה לשימוש בשרת מסחרי.
- SQL Server 2000 Desktop Engine (MSDE) – מספקת את כל התכונות של Standard Edition. ניתן להפיץ גירסה זו באופן חופשי כחלק מיישומים קטנים או גרסאות הדגמה. גודל מסד הנתונים של Desktop Engine מוגבל ל-2GB, ובדומה ל- Personal Edition, ביצועיה מוגבלים כאשר מבוצעות יותר מחמש אצוות בו-זמנית. עם זאת, גירסה זו אינה כוללת כלי ניהול או כלי פיתוח גרפיים.

הערה

כל מהדורה של VB.NET או VS.NET כוללת את גרסת MSDE של SQL Server 2000. מהדורות Enterprise Developer ו- Enterprise Architect של Visual Studio כוללות גם את Developer Edition של SQL Server 2000.

זכור את הנקודות החשובות שלהלן:

- MSDE אינה כוללת את הכלים הגרפיים של SQL Server המתוארים בפרק זה. לכן, לא תוכל לבצע את ההדגמות והדוגמאות המתוארות בו (עם זאת, יהיו ברשותך מספר כלי נתונים גרפיים לגישה ל-MSDE מתוך סביבת הפיתוח של VS.NET).
- Developer Edition של SQL Server מורשית לשימוש בפיתוח בלבד. לפיתוח יישום מסחרי ב- SQL Server עליך לרכוש רשיונות מתאימים לשרת ולגישת לקוח עבור SQL Server 2000.

- SQL Server 2000 Windows CE Edition – משמשת כמחסן נתונים בהתקני Windows CE. גירסה זו יכולה להחליף נתונים עם כל אחת מגרסאות SQL Server 2000 האחרות.

בדיקת דרישות התקנה עבור SQL Server 2000

על פי מיקרוסופט, להתקנת SQL Server 2000 יש להשתמש במעבד פנטיום (או תואם) במהירות מינימלית של 166MHz; 95-270GB מקום פנוי בכונן הקשיח (250MB להתקנה רגילה, 44MB ל- Desktop Engine); כונן תקליטורים; Internet Explorer 5.0 או עדכני יותר; ומערכת הפעלה נתמכת. להלן דרישות זיכרון RAM:

- Standard Edition – 64MB לפחות.
- Enterprise Edition – 64MB לפחות, מומלץ 128MB.
- Personal Edition – 64MB לפחות ב- Windows 2000; 32MB לפחות במערכות הפעלה אחרות.
- Developer Edition – 64MB לפחות.
- Desktop Engine – 64MB לפחות ב- Windows 2000; 32MB לפחות במערכות הפעלה אחרות.

אם כבר ניסית להפעיל את SQL Server במחשב עם מעבד 166MHz בעל זיכרון של 64MB ואתה רואה שהתוכנה פועלת כראוי, המתן עם ההתלהבות והמשך לקרוא. אלו הן דרישות **המינימום**. התוכנה SQL Server עשויה לפעול במחשב ישן מעין זה, אך ספק רב אם אתה, המשתמש, תוכל לראות משהו מעבר ללוגו שלה. בעולם האמיתי, עליך להשתמש במחשב החזק ביותר שתוכל להרשות לעצמך. זה אמור להיות המחשב שעליו יפעל העסק כולו; חיסכון בחומרה יגרום לך להצטער מאוחר יותר. עליך להרחיב את הזיכרון ללא היסוס. מעשית, אם התקציב שלך מוגבל, מומלץ להשקיע

בזיכרון נוסף מאשר בתוספת מהירות למעבד. שדרוג צנוע של זיכרון עשוי לשפר מאוד את ביצועי המערכת.

הערה

ספר זה סוקר פתרונות מונחי מסד נתונים ב-VB.NET, ולכן איננו חוקרים את כל התכונות של SQL Server. תיעוד SQL Books Online אשר מגיע עם SQL Server הינו מקור טוב לקבלת מידע מפורט מעין זה. אם אתה מחפש ספר לימוד, מומלץ לעיין בספר **המדריך ל- SQL Server 2000** בהוצאת **הוד-עמי**.

התקנת SQL Server 2000

לאחר שהכנת מחשב להתקנת SQL Server, תוכל להתחיל בהתקנה.

ההתקנה ישירה למדי, עם מספר פעולות התערבות במהלכה.

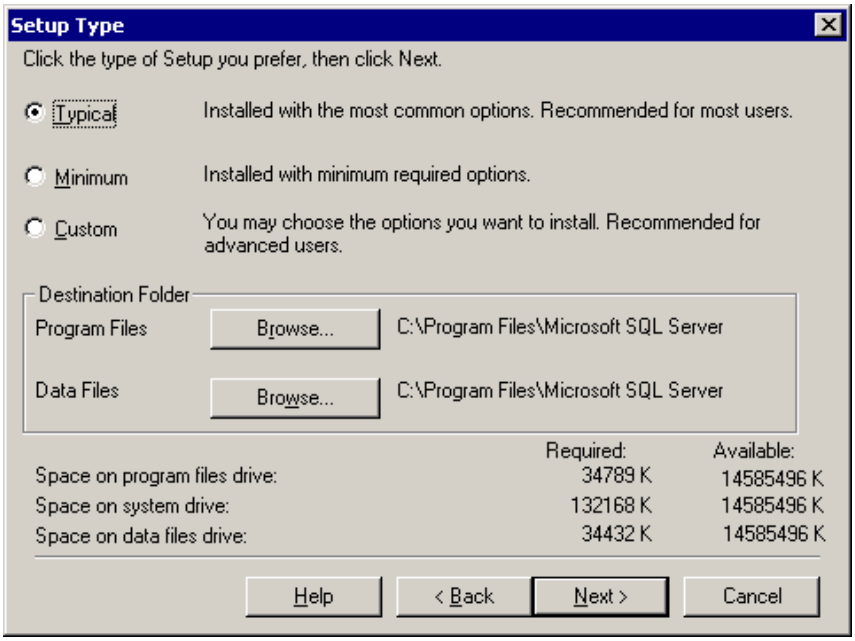
- ההתקנה נמשכת זמן רב.
- במהלך ההתקנה תישאל שאלות רבות ושונות שאינן מופיעות בדרך כלל בהתקנת יישומים רגילים.

מבחינת זמן ההתקנה לא נוכל לצערנו לעזור, אך נוכל לספק לך מידע אודות השאלות שמוצגות על ידי תוכנית ההתקנה של SQL Server.

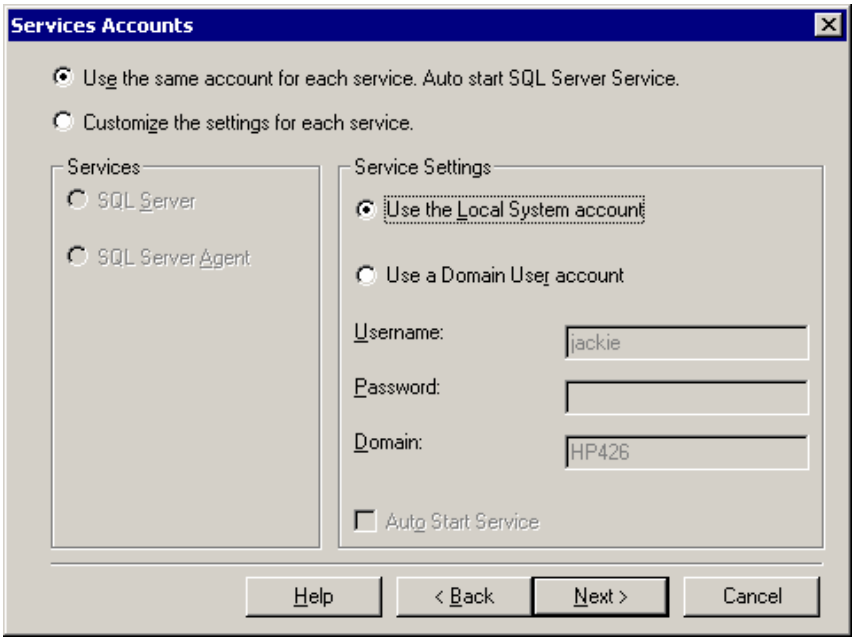
בעיקרון, ובמיוחד בתצורות פיתוח פשוטות, מומלץ לקבל את אפשרויות ברירת המחדל המוצעות על ידי דפי הדו-שיח של אשף ההתקנה. ההערות שלהלן מתייחסות לתיבות הדו-שיח אשר דורשות מחשבה מעמיקה יותר.

בתיבת הדו-שיח Setup Type המוצגת בתרשים 1.1, תוכל לבחור התקנה רגילה (typical), מינימלית (minimum) ומותאמת (custom). תוכל גם לבחור בנתיב ההתקנה של תוכנית SQL Server וקבצי הנתונים. ודא שיש מספיק שטח פנוי בכונן הקשיח המיועד לאחסנת קבצי הנתונים ורצוי לוודא שהם נמצאים בנתיב שמגובה באופן קבוע על ידי מערכת הגיבוי הרגילה שלך.

בתיבה Services Accounts המוצגת בתרשים 1.2, ברירת המחדל היא חשבון Domain User, אך תוכל להשתמש בחשבון Local System אם אינך נמצא ב-domain או שלרשותך מחשב שרת פיתוח נפרד. בדף דו-שיח זה תוכל לקבוע אם SQL Server יופעל באופן אוטומטי עם הפעלת Windows. אם תבחר באפשרות זו, קח בחשבון כי SQL Server יופעל כשירות מ-Windows. חשבונות שירותים (Services Accounts) פועלים כאילו היו חלק ממערכת ההפעלה; הם אינם מוצגים במנהל המשימות ולא ניתן לכבותם בדומה ליישומים רגילים. בסעיף הבא נספק מעט מידע אודות ניהול של שירות הפועל בסביבת Windows, אך מומלץ לקרוא גם את הסעיף "קביעת אופן ההפעלה של SQL Server" בהמשך פרק זה.

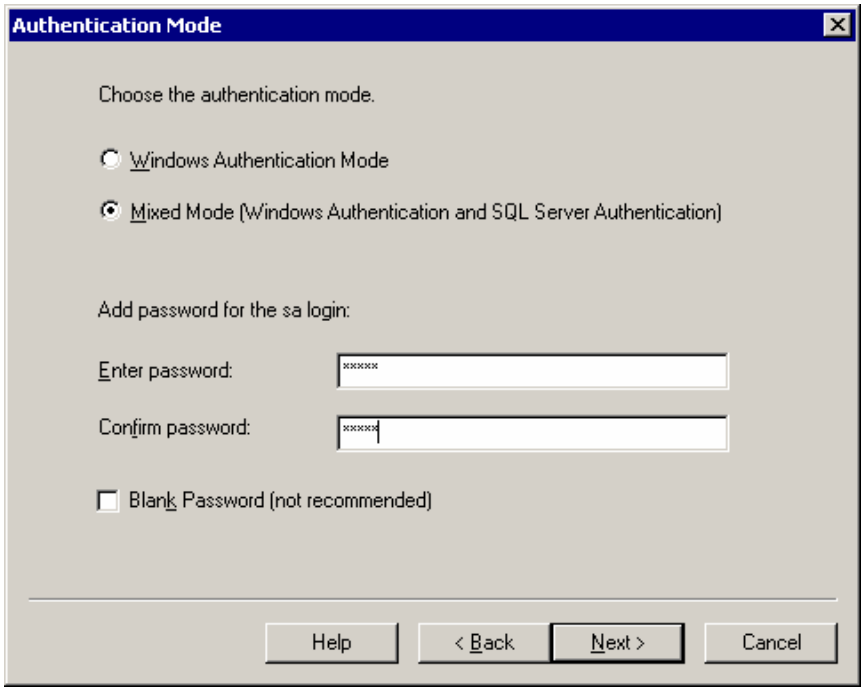


תרשים 1.1 תיבת הדו-שיח Setup Type של אשף ההתקנה SQL Server.



תרשים 1.2 תיבת הדו-שיח Services Accounts של אשף ההתקנה SQL Server.

בשרת מסחרי מומלץ להשתמש ב- Windows Authentication Mode. זוהי ברירת המחדל. מצב זה מנצל חשבון משתמש קיים ב- Windows NT/2000 וגם מנגנוני אבטחה קיימים. כאשר מתבצע ניסיון להתחבר ל- SQL Server, האימות מתבצע באמצעות חשבון המשתמש. אם המשתמש (או הקבוצה) קיבלו הרשאת גישה ל- SQL Server, הכניסה תאושר. גישה זו פשוטה ובה מתבצע ניהול חשבונות משתמשים וקבוצות ממיקום יחיד.



תרשים 1.3 תיבת הדו-שיח Authentication Mode של אשף ההתקנה של SQL Server.

במקרים מסוימים, ייתכן שתצטרך להשתמש ב- Mixed Mode כפי שמסומן בתרשים 1.3. בנוסף לאפשור Windows Authentication, Mixed Mode מאפשר גם את SQL Server Authentication. שיטת האימות השנייה מחייבת הגדרת חשבונות משתמשים בתוך SQL Server, התואמים לנסיונות ההתחברות. היתרון העיקרי של מצב זה הוא בכך שאין צורך בחיבור מהימן (trusted connection) בין השרת לתחנת העבודה שממנה מתבצע החיבור, ולכן צורת חיבור זו מועדפת בלקוחות UNIX או Web אשר ניגשים למסד הנתונים. עם זאת, יש צורך בעבודה נוספת של ניהול חשבונות (חשבונות Windows וחשבונות SQL Server).

הערה

לעיתים קרובות, יהיה נוח יותר להגדיר מחשב פיתוח ב- Mixed Mode, כך שתוכל פשוט להשתמש בחשבון sa (system administrator) המותקן מראש. עם זאת, עליך לפתח גישה מתוחכמת ומאובטחת יותר במחשב המסחרי – הקצה לפחות סיסמה מתאימה לחשבון sa!

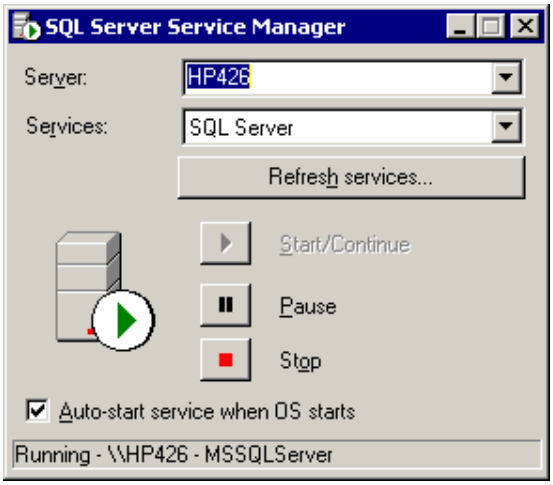
הפעלה והפסקת SQL Service באמצעות SQL Service Manager

באמצעות SQL Service Manager ניתן להפעיל ולהפסיק את SQL Server במקרים בהם ברצונך להפסיק את פעולת השרת כדי לבצע משימות מסוימות, או אם אינך רוצה להפעיל את SQL Server במחשב מסוים (לדוגמה, במחשב פיתוח).

במצב רגיל לא תיתקל בצורך להפסיק את פעולת SQL Server, אשר תפקידו – לשרת כמערכת מסד נתונים ארגונית (enterprise database system). הרעיון הוא שעליך להשאיר את השירות פעיל ברציפות. עם זאת, במקרים נדירים, עליך להפסיק את השירות כדי לבצע משימות מסוימות, כגון שינוי אפשרויות תצורה בשרת או ביצוע שדרוג חומרה במחשב בו נמצא השרת. במקרה כזה, השתמש ב- SQL Service Manager והפסק בעזרתו את SQL Server. השתמש בשירות זה גם כדי להחזיר את SQL Server לפעולה לאחר מכן.

SQL Server Manager אינו חייב להיות פעיל לצורך הפעולה השוטפת של SQL Server, אלא רק כדי לאפשר שליטה בהפעלה וכיבוי של השרת. לאחר שהשרת נמצא במצב עבודה מסחרית, סביר להניח שלא תרצה להשתמש באופן קבוע בתוכנה SQL Service Manager.

כאשר תפעיל את SQL Service Manager באמצעות בחירת הסמל המתאים מקבוצת התוכניות SQL Server, יוצג חלון כמו זה שמופיע בתרשים 1.4.



תרשים 1.4 SQL Service Manager במצבו הראשוני, כאשר SQL Server פעיל.

אם SQL Server פעיל, מחוון המצב (status indicator) ייראה כחץ ירוק; אם השרת אינו פעיל, המחוון ייראה כריבוע אדום. להפעלת SQL Server, לחץ Start/Continue; להפסקת פעולת השרת, לחץ Stop. אין פשוט מזה, אבל עליך להמתין מעט עד שהמצב ישתנה.

קביעת אופן ההפעלה של SQL Server

לאחר הגדרת SQL Server, מערכת ההפעלה תפעיל את השרת באופן אוטומטי במהלך אתחול המחשב שתוכנת השרת מותקנת בו. בלוח הבקרה Services, תוכל לקבוע אם SQL Server יופעל תמיד עם הפעלת המחשב. להצגת המצב הנוכחי של SQL Server וכדי לקבוע את אופן הפעלתו באתחול המחשב, בצע את הפעולות שלהלן.

1. הפעל את לוח הבקרה (Control Panel) של Windows. בחר **Administrative Tools** מלוח הבקרה (Control Panel).
2. בחר **Services** מ- Administrative Tools.
3. כעת יוצג לוח הבקרה Services. גלול את רשימת השירותים עד שתגיע ל-MSSQLServer.

אם התקנת זה עתה את SQL Server במחשב, מצב השירות MSSQLServer יהיה Started והפעלתו תהיה אוטומטית (Automatic). להפסקת השירות MSSQLServer מלוח הבקרה Services:

1. לחץ **Stop**.
2. לאחר מספר שניות, SQL Server יופסק.
3. לאתחול SQL Server, לחץ **Start** בלוח הבקרה Services.

הערה

ההפעלה והכיבוי של SQL Server מתוך לוח הבקרה Services נעשים באופן דומה להפעלה וכיבוי של השרת על ידי SQL Service Manager. ההבדל הוא בממשק המשתמש בלבד.

יסודות העבודה ב- SQL Server 2000

לאחר התקנת השרת, עליך לבצע מספר משימות כדי להתחיל באחסון נתונים ובאחזורם מ- SQL Server:

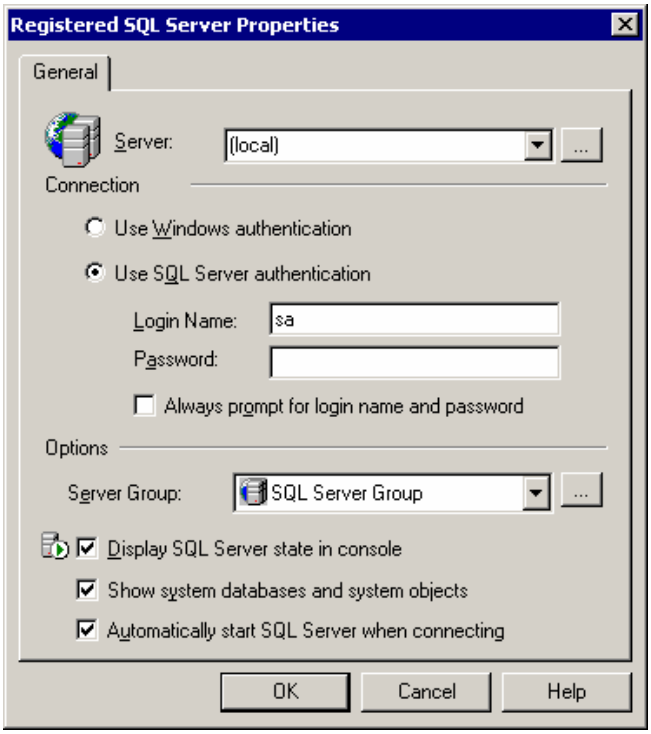
- הגדרת חשבונות משתמשים וקבוצות אבטחה.
- יצירת מסד נתונים.
- יצירת טבלאות במסד נתונים.
- יצירת תצוגות ושגרות מאוחסנות אשר קובעות כיצד יאוחזרו הנתונים ממסד הנתונים (כדאי, אך לא חובה).

כל המשימות שעליך לבצע מתוארות בסעיף זה. ניתן לבצע את רובן מבלי לכתוב קוד, באמצעות תוכנית השירות SQL Server Enterprise Manager שב- SQL Server 2000.

הפעלת SQL Server Enterprise Manager

תוכל לבצע פעולות רבות הקשורות בהגדרת תצורת מסדי נתונים שכיחים ב- SQL Server, באמצעות שימוש בתוכנית השירות SQL Server Enterprise Manager. העוצמה וקלות השימוש ב- SQL Server Enterprise Manager הופכים אותו לאחד מהכלים החשובים ביותר ב- Microsoft SQL Server 2000. תוכנית שירות זו מפשטת את משימות הניהול של מסד נתונים, ומציעה ממשק גרפי קל ונוח במקום להשתמש בפקודות SQL מסובכות.

ניתן להפעיל את Enterprise Manager מהסמל המתאים בקבוצת התוכניות Microsoft SQL Server. לאחר הפעלת היישום, תוכל לגשת לתוכנת SQL Server הפועלת ברשת שלך. הסעיפים שלהלן מתארים מספר משימות שכיחות שיש לבצע ב- Enterprise Manager המותקן ביישום מסחרי.



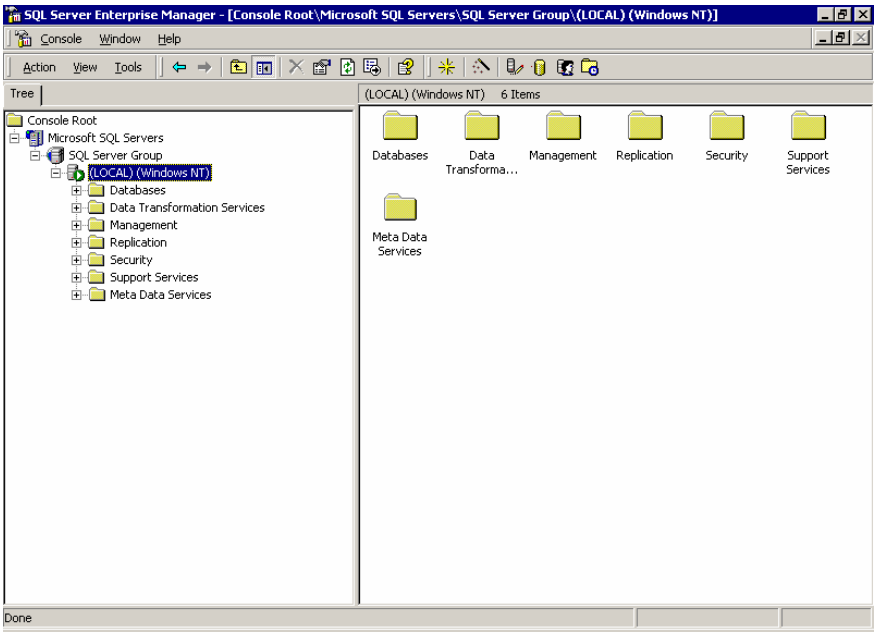
תרשים 1.5 תיבת הדו-שיח Registered SQL Server Properties של Enterprise Manager.

בפעם הראשונה שתפעיל את Enterprise Manager, עליך לרשום את התקנת SQL Server. פעולה זו תאפשר ל- Enterprise Manager לדעת איזו תוכנת SQL Server ברצונך להפעיל; פעולה זו גם תאפשר לנהל יותר מהתקנה יחידה של SQL Server. רישום SQL Server מתבצע מתיבת הדו-שיח Registered SQL Server Properties המוצגת בתרשים 1.5.

אם אתה מנסה לרשום את התוכנה SQL Server הפועלת במחשב שאתה עובד בו, הדרך הקלה לעשות זאת היא להשתמש בשם השרת (local), כולל סוגריים. אם אתה מנסה להתחבר ל-SQL Server דרך רשת LAN, קל לעשות זאת באמצעות הלחצן (...) Server browse ולאתר שרתים זמינים ברשת.

טיפ

תיבת הדו-שיח Registered SQL Server Properties מכילה תיבת סימון חשובה: Show system databases and system objects, שפעמים רבות קשה למצוא אותה. אם תיבה זו אינה מסומנת, מסדי הנתונים והאובייקטים יישארו חבויים בחלונות Enterprise Manager השונים. בצורה זו ניתן לארגן את התצוגה בצורה נוחה יותר ולהסיר פריטים מיותרים כאשר אתה עובד עם טבלאות וקבצים של היישום שלך בלבד. עם זאת, אם תרצה לצפות באובייקטי מערכת אלה, חזור לתיבת דו-שיח זו וסמן את התיבה.



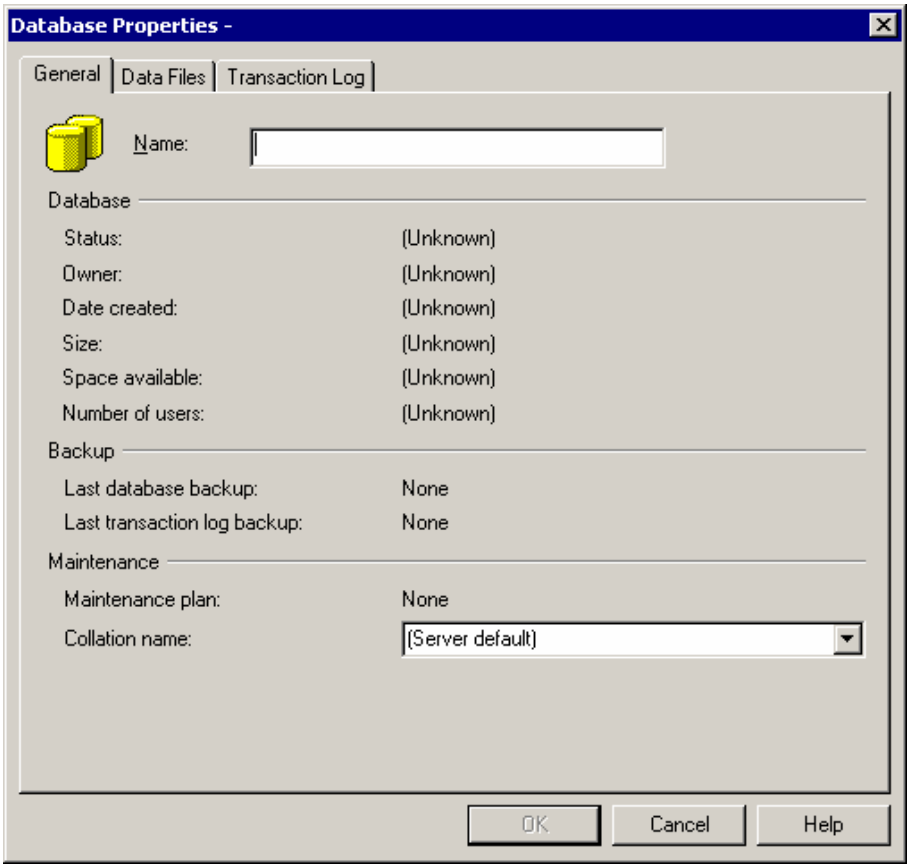
תרשים 1.6 חלון Microsoft SQL Servers MMC עם SQL Server מקומי ב- Enterprise Manager.

לאחר רישום השרת שברצונך להפעיל, עליך ללחוץ OK בתיבת הדו-שיח Registered SQL Server Properties. עליך לעשות זאת פעם אחת בלבד. לאחר מכן, SQL Enterprise Manager יזכור כיצד להתחבר לשרת הרצוי. שם השרת שלך יופיע גם בחלון Microsoft SQL Servers MMC, יחד עם שרתים אחרים שרשמת. במחשב עם חיבור ל- SQL Server מקומי, חלון Microsoft SQL Servers MMC ייראה כפי שמוצג בתרשים 1.6.

יצירת מסד נתונים באמצעות SQL Server Manager

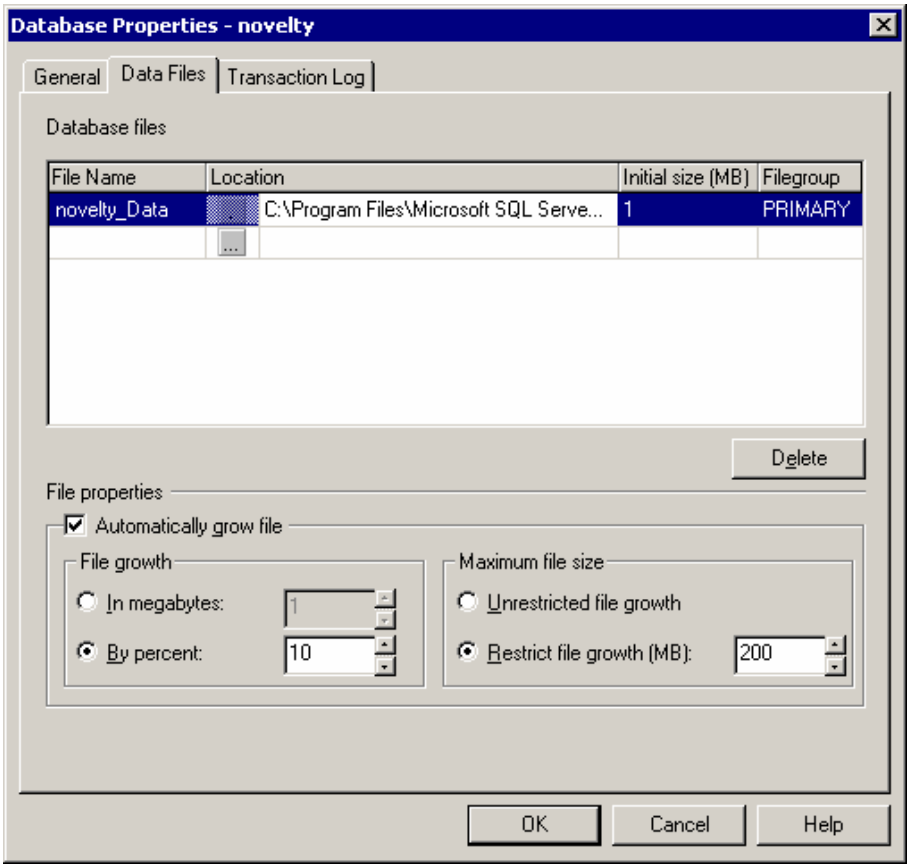
לאחר רישום השרת שלך תוכל להתחיל לעבוד. השלב הבא הוא ליצור מסד נתונים ולהתחיל למלא אותו באובייקטי מסד נתונים, כגון טבלאות, תצוגות ושגרות מאוחסנות.

ניתן ליצור מסדי נתונים על ידי כתיבת קוד SQL, אך קל בהרבה לעשות זאת בעזרת Enterprise Manager. הסיבה היא ש- Enterprise Manager מאפשר לעצב את רוב אובייקטי מסד הנתונים בצורה גרפית, ומונע ממך את הצורך לעסוק בקוד SQL מורכב. בצע את הפעולות שלהלן כדי ליצור מסד נתונים חדש על ידי SQL Server Enterprise Manager.



תרשים 1.7 תיבת הדו-שיח Database Properties של Enterprise Manager.

1. לחץ לחיצה ימנית על התיקיה **Database** בחלון Microsoft SQL Servers MMC של Enterprise Manager.
2. מהתפריט הנפתח, בחר **New Database**. תוצג תיבת הדו-שיח Database Properties, כפי שמוצג בתרשים 1.7.
3. הקלד את שם מסד הנתונים החדש בתיבת הטקסט Name (בדוגמאות שבספר זה נשתמש בשם Novelty).
4. כברירת מחדל, הנתונים מאוחסנים בקובץ *database-name*_Log.ldf. ניתן לשנות שמות ברירת מחדל אלה וגם את הנתיב שהקבצים מאוחסנים בו, על ידי שינוי שם הקובץ ו/או המיקום בכרטיסיות Datafiles ו- Transaction Log. הכרטיסיה Data Files מוצגת בתרשים 1.8.



תרשים 1.8 בכרטיסיה Data Files של תיבת הדו-שיח Database Properties ניתן להגדיר את מיקום קובץ כמו גם פרטים נוספים.

הערה

שלא כמו בגרסאות קודמות של MS SQL Server, אין צורך לקבוע מראש ולהקצות את גודל הנתונים וקבצי היומן. SQL Server 2000 מאפשר גידול אוטומטי בגודל הקבצים על פי הדרוש, בקפיצות של מספר קבוע של מגה-בתים או באחוזים מגודל הקובץ הנוכחי. עליך לציין גודל מרבי לקובץ, כך שהקובץ לא יגדל ללא שליטה עד למילוי הכונן הקשיח כולו.

5. לחץ **OK**. בכונן הקשיח ייווצרו שני קבצים חדשים: Novelty_Data.mdf ו- Novelty_Log.ldf. הגודל ההתחלתי של כל קובץ הוא 1MB.

6. מסד הנתונים החדש ייווצר ותיבת הדו-שיח Database Properties תיסגר. מסד הנתונים החדש יופיע ברשימת התיקיה Databases של החלון Microsoft SQL Servers MMC.

הערה

בכרטיסיה General תוכל לציין גם אוסף ברירת מחדל עבור מסד הנתונים. אוסף (collation) קובע את ערכת התווים ואת הכללים על פיהם מאוחסנים ומושווים תווים. הגדרה זו חשובה במיוחד כאשר אתה עוסק בנתונים המכילים שפות שאינן אנגלית. בהגדרה זו משתמשים גם כדי לציין מיון והשוואה לפי אותיות רישיות.

יצירת טבלאות ב- SQL Server Database

ב- Microsoft SQL Server ניתן ליצור טבלאות באחת משתי דרכים:

1. שימוש ב- SQL Data Definition Language (DDL).
2. שימוש בתכונות הגרפיות ליצירת טבלאות על ידי SQL Server Enterprise Manager.

לשתי טכניקות אלו יש חסרונות ויתרונות. השימוש בפקודות SQL DDL מורכב יותר מאשר בנייה גרפית של טבלאות, במיוחד אם לא עבדת ב-SQL בלבד בעבר; השימוש ב-SQL גמיש יותר ומאלץ אותך לכתוב ולתחזק קוד ליצירת מסד הנתונים. עם זאת, פקודות DDL הן מטבען פקודות אוטומטיות, בדומה ליצירת תסריט לבניית מסד נתונים בלחיצה אחת; הן מאפשרות גם פונקציונליות ואפשרויות שאינן מוצגות בממשק המשתמש של Enterprise Manager, ולכן הן מהוות שיטה (גולמית למדי) לתיעוד סכמת מסד הנתונים.

כאשר נבחן את השימוש ב- Enterprise Manager של SQL Server, נראה שהוא מאפשר ליצור מבנה מסד נתונים במהירות ובקלות תוך ניצול כל היתרונות הטמונים בממשק משתמש גרפי. עם זאת, לא קל ליצור פעולות אוטומטיות ב- Enterprise Manager.

מתכנתי מסד נתונים מסוימים מעדיפים להשתמש בקוד SQL וליצור בעזרתו את מסד הנתונים שלהם, ולכן יש להם תמיד תיעוד כתוב (בצורת קוד SQL DDL) של מה שהתבצע בעת יצירת מסד הנתונים. הטכניקה שתשתמש בה היא פונקציה של ההעדפה האישית שלך, תקני הפיתוח של הארגון וסוגי היישומים של מסד הנתונים שתיצור. בסעיפים הבאים נציג את שתי הטכניקות.

טיפ

ל- SQL Server 2000 יש אפשרות ליצור קוד SQL DDL עבור אובייקטים במסדי נתונים קיימים. תוכל לצפות בהם על ידי לחיצה ימנית על מסד נתונים בתיקיה Databases שבחלון Microsoft SQL Servers MMC, בחירה באפשרות All Tasks ולאחר מכן בחירה באפשרות Generate SQL Script כדי להציג את החלון Generate SQL Scripts.

יצירת טבלאות ב- SQL Server באמצעות Enterprise Manager

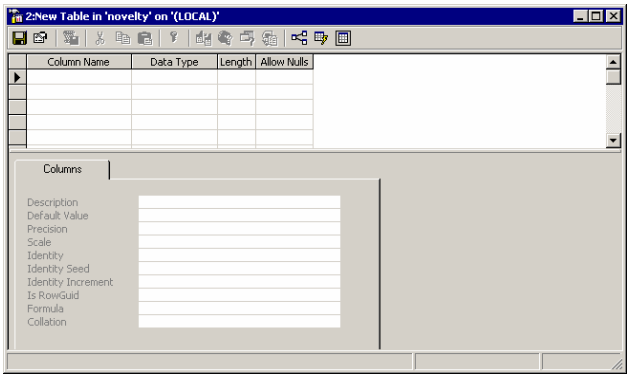
לאחר יצירת מסד נתונים ב- SQL Server, תוכל להשתמש באפשרות SQL Server Enterprise Manager כדי ליצור טבלאות במסד הנתונים. בצע את הפעולות שלהלן כדי ליצור טבלה במסד נתונים.

1. בחלון Enterprise Manager של Microsoft SQL Servers MMC, הרחב את הרכיב המייצג את מסד הנתונים שברצונך ליצור בו טבלה.
2. לחץ לחיצה ימנית על הרכיב **Tables**.
3. מהתפריט הנפתח, בחר **New Table**. כעת תוצג תיבת הדו-שיח Design Table, כפי שמוצג בתרשים 1.9.

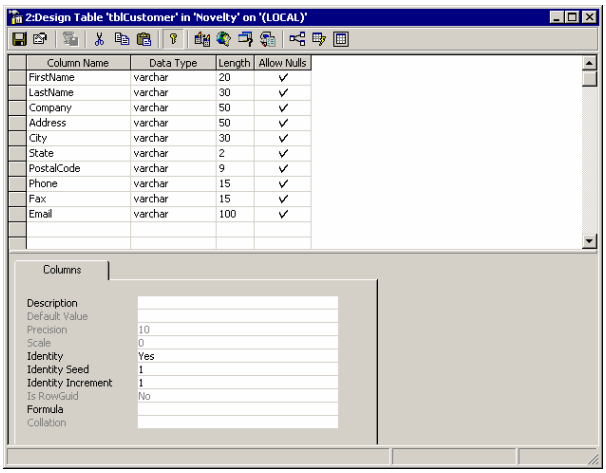
הערה

כאשר תציג תיבת דו-שיח זו בפעם הראשונה, תיאור תיבת הדו-שיח Design Table יתחיל ב- New Table ולא ב- Design Table.

4. התחל ביצירת טבלה לאחסון לקוחות. לשם כך, לחץ בתיבת הדו-שיח **Design Table** על העמודה Column Name. לאחר מכן, כתוב את שם השדה הראשון בטבלה FirstName.
5. הקש **Tab** כדי לעבור לעמודה הבאה – Datatype. בעמודה זו קבע את סוג הנתונים כשדה טקסט בעל אורך משתנה, או varchar, על ידי בחירת varchar בתיבה המשולבת הנפתחת. סוג הנתונים varchar משמש בעיקרון ב- SQL Server לאחסון קטעים קטנים יחסית של נתוני מחרוזות.
6. בעמודה הבאה, הקלד את המספר **20** כדי להגביל את מספר התווים בשדה FirstName ל-20 תווים.



תרשים 1.9 תיבת הדו-שיח Design Table של Enterprise Manager, שניתן לרשום בה את הגדרות עיצוב טבלאות מסד הנתונים.



תרשים 1.10 תיבת הדו-שיח Design Table ובה הגדרות שדות לטבלה החדשה.

7. העמודה Allow Nulls קובעת אם שדה יציג ערכים ריקים (null). אם התיבה מסומנת, ניתן להזין ערכים ריקים לשדה. תיבה זו צריכה להיות **מסומנת** בשדה FirstName.

8. הזן הגדרות וסוגי נתונים עבור השדות הנוספים בטבלה זו. בסיום הגדרת הטבלה, המסך אמור להיראות בדומה לתרשים 1.10.

הערה

בנקודה זו, ייתכן שתרצה ליצור שדה המשמש מזהה ייחודי עבור כל רשומה בטבלה. ב- SQL Server, שדה זה נקרא **עמודת זיהוי** (identity column). כעת זהו הזמן לבצע זאת, מכיון שלא תוכל ליצור עמודת זיהוי עבור הטבלה לאחר שתיצור אותה ותוסיף בה נתונים. הסיבה לכך היא ששדות מפתח אינם יכולים להכיל ערכים ריקים. ניתן להגדיר שדות שאינם מכילים ערכים ריקים בטבלה רק לפני שהם מכילים נתונים. SQL Server אינו גמיש במובן זה כמו Microsoft Access, אך זה המחיר שעליך לשלם עבור ביצועים משופרים ומדרגיות (scalability) טובה יותר שקיימים ב- SQL Server. בסעיף הבא נספק מידע נוסף אודות יצירת עמודות זיהוי בעת יצירת הטבלה.

9. כאשר תסיים לעצב את הטבלה, לחץ **Save** בסרגל הכלים שבראש תיבת הדו-שיח.

10. כעת תוצג תיבת הדו-שיח Choose Name. הקלד את שם הטבלה בתיבה ולחץ **OK**. תוכל להשתמש כמעט בכל שם רצוי, אך בדוגמאות המובאות בספר זה נשתמש בשם tblCustomer.

11. הטבלה החדשה תוצג כעת בחלון Microsoft SQL Servers MMC.

יצירת עמודת זיהוי לזיהוי ייחודי של רשומות

נוח (אך לא חובה) להקצות לכל רשומה מידע אשר יזהה אותה באופן ייחודי. לעיתים, מזהה ייחודי זה אינו קשור לנתונים המיוצגים על ידי הרשומה. ב- SQL Server ניתן להגדיר עמודה שתהיה עמודת זיהוי (מקביל לסוג שדה AutoNumber ב-Jet). בזמן יצירת הרשומה, עמודת הזיהוי מקצה ערך מספרי ייחודי באופן אוטומטי לעמודה בכל רשומה.

אם אתה מכיר את שדה AutoNumber של Jet, תוכל להשתמש בידע זה להשוואה עם עמודת זיהוי ב- SQL Server, אשר שונה ובמובנים מסוימים מתוחכמת יותר משדה Autonumber של Jet. התכונות של עמודת זיהוי ב- SQL Server הם:

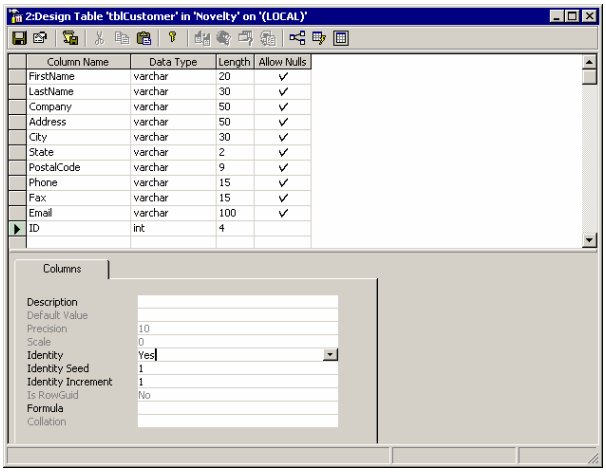
- ניתן להגדירה באמצעות כל סוג נתונים מספרי (ב-Jet ניתן להגדיר שדות אלה כ- long integer בלבד).
- ניתן לקבוע עבורה הגדלה בכל סכום שתציין (ב-Jet ניתן להגדיל שדות אלה ב-1 או במספר אקראי).
- ניתן להתחיל מספור בכל ערך רצוי (ב-Jet, הספירה מתחילה תמיד ב-1).
- ניתן לשכתב אותה. תכונה זו מאפשרת להכניס מספרים מסוימים לעמודת זיהוי. לדוגמה, כדי לבנות מחדש רשומה שנמחקה בטעות (ב-Jet, עמודות הזיהוי הן מסוג קריאה בלבד).

עמודת זיהוי של SQL Server גמישה פחות משדה Autonumber של Jet במובן אחד: אם תיצור עמודת זיהוי בטבלה, עליך לעשות זאת כאשר תיצור את הטבלה ולפני הוספת נתונים. הסיבה לכך היא שקיימת דרישה ב- SQL Server שכל שדה אשר נוצר לאחר מכן חייב לאפשר ערכים ריקים; אם כך, שדות אשר מכילים ערכים לא ריקים בלבד ניתן ליצור רק לפני שהטבלה מכילה נתונים כלשהם.

בצע את הפעולות שלהלן כדי ליצור עמודת זיהוי על ידי SQL Server Enterprise Manager.

1. בתיבת הדו-שיח Design Table, צור שדה חדש בשם **ID**. הגדר סוג נתונים **int** עבור שדה זה. זכור כי סוג הנתונים int של SQL Server הוא באורך של ארבעה בתים, בדומה לסוג הנתונים Integer של Visual Basic.NET.
2. בטל את סימון התיבה Allow Nulls. בצורה זו תבטיח שלא ניתן יהיה להכניס ערכים ריקים לעמודה זו; כך תוכל להשתמש בעמודה זו גם כעמודת זיהוי.
3. בחלק התחתון של תיבת הדו-שיח Design Table, מוצג דף תכונות עבור העמודה שנבחרה בטבלה. עבור לשדה **Identity** שבדף התכונות באמצעות העכבר או על ידי הקשת **tab**.
4. בתיבת הרשימה Identity, בחר **Yes**. ניתן לקבוע ערכים בתיבות Identity Seed ו- Identity Increment; תיבות אלו קובעות את המקום שבו יחל המספור האוטומטי ואת הערך שיוקצה לכל מספר עוקב.

לאחר הגדרת העמודה Identity, תיבת הדו-שיח Design Table תיראה בדומה לזו שבתרשים 1.11.



תרשים 1.11 יצירת עמודת זיהוי בתיבת דו-שיח Design Table ב- SQL Server Enterprise Manager.

זכור שכאשר אתה משתמש בעמודות זיהוי ב- SQL Server, הן לא תהיינה בהכרח רציפות. לדוגמה, אם משה מנסה ליצור רשומה עם מספר זיהוי 101, רחל תיצור את הרשומה הבאה (מספר זיהוי 102), פעולת הכנסת הרשומה של משה תיכשל ולעולם לא תיווצר רשומה עם מספר זיהוי 101.

עובדה זו אינה אמורה להפריע, במיוחד ביישום מסד נתונים שאינו חושף לעולם את ערך המפתח הראשי בפני המשתמש (זו תפיסת תכנון שיש לשמר). עם זאת, זכור כי יכולים להיות ערכי זיהוי "אבודים". אם אתה משתמש בשדה זיהוי לזיהוי ייחודי של רשומות, אל תופתע אם תעיין יום אחד בנתונים ותגלה למשל שחשבונית מספר 101 אינה קיימת.

יצירת מפתחות ראשיים בשיטות נוספות

טבלאות מסד הנתונים שתיצור אינן חייבות להכיל מפתחות ראשיים. עם זאת, סביר שתחליט להשתמש במפתחות ראשיים – גם אם מדובר בערך המוקצה על ידי השרת בזמן יצירת הרשומה. כזכור לך, דרושים מפתחות ראשיים לביצוע פעולות כגון איחוד מספר טבלאות בשאילתות. השימוש במפתחות ראשיים הוא גם דרך נוחה לפנות לרשומות בממשק המשתמש. במקום להעביר את הרשומה כולה משגרה אחת לאחרת, תוכל להעביר רק את נתוני המפתח הראשיים.

דרך חילופית אחת ליצירת מפתחות ראשיים היא ליצור ערך אקראי ייחודי בשדה המפתח הראשי עבור כל רשומה שנוצרת. דרך זו נמצאת בשימוש בטבלאות המכילות שדות Auto Number אשר הומרו, או **הוגדלו**. שיטה זו משמשת, החל מ- Microsoft Access וגם ב- SQL Server, למניעת התנגשויות בין רשומות במסדי נתונים משוכפלים של Access אשר מוזנים על ידי משתמשים מרוחקים ומנותקים.

שיטה נוספת היא לאחסן ערך מונה בטבלה זמנית אשר משתמשת בו לקביעת עמודת המפתח הראשי של כל רשומה חדשה בזמן יצירתה. בשיטה זו משתמשים בביצוע תנועה (Transaction) אשר קוראת את הערך הנוכחי של טבלת המונים, ממלאת באמצעותו את המפתח העיקרי של הרשומה החדשה ולאחר מכן מגדילה את ערך המונה בטבלה ביחידה אחת. לטכניקה זו יש יתרון בכך שהיא מספקת מערכת מספור רציפה שיש לך עליה שליטה מלאה. החיסרון, בהשוואה לטכניקה הפשוטה יותר של יצירת עמודת זיהוי, הוא בכך שעליך לכתוב שגרה מאוחסנת ולהוסיף טבלאות למסד הנתונים שלך כדי שתוכל ליישם אותה. אחסון ערך מונה יכול גם ליצור התנגשויות בטבלה יחידה, אשר עלול לגרום לבעיית ביצועים במערכת עמוסה.

השיטה האחרונה היא ליצור מפתח מתוך הנתונים; לדוגמה, המפתח עבור רשומה של אדם בשם Vito Polito עשויה להיות VP001. עם יתווסף אדם נוסף שראשי התיבות של שמו הם גם כן VP, הרשומה תקבל את המפתח VP002 וכן הלאה. לשיטה זו יתרון בכך שהיא מספקת מפתח שאינו מנותק לחלוטין מהנתונים, אך השימוש בה מחייב כתיבת קוד רב בצורת שגרה מאוחסנת המופעלת כמזניק (trigger), כמוסבר בהמשך הפרק.

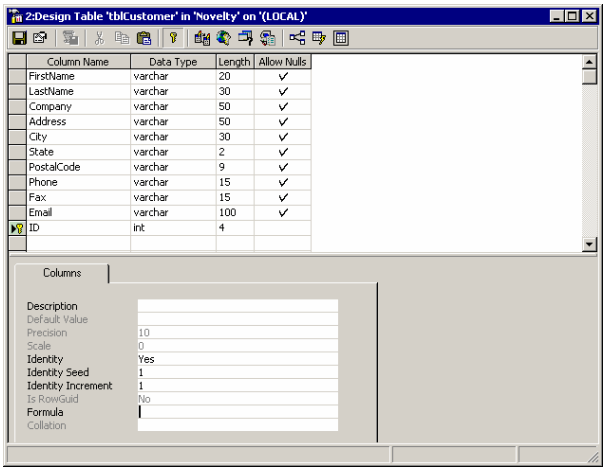
הגדרת עמודה כמפתח ראשי

כאשר אתה יוצר עמודת זיהוי, תרצה בוודאי להגדיר אותה כמפתח הראשי של הטבלה. תוכל לעשות זאת בתיבת הדו-שיח Design Table של SQL Server Enterprise Manager, בצורה הבאה:

1. בחר בשורה המכילה את העמודה (שדה) שברצונך להפוך למפתח הראשי של הטבלה.
2. לחץ על **Set Primary Key** (סמל מפתח) בסרגל הכלים. אינדקס המפתח העיקרי יתווסף להגדרת הטבלה, ותיבת הדו-שיח Design Table תיראה בדומה לזו שבתרשים 1.12. עמודות המכילות את סמל המפתח בעמודה הראשונה של הרשת הן שדות אשר מכילים את המפתח הראשי של הטבלה. כמו כן, שים לב שכל שדה יכול לשמש כמפתח ראשי של הטבלה, ולא רק עמודת זיהוי.

הערה

ניתן להגדיר מספר שדות כמפתח העיקרי של הטבלה; שיטה זו נקראת **מפתח משורשר**. לדוגמה, תוכל לעשות זאת במצבים שבהם ברצונך להגדיר כי השם הפרטי ושם המשפחה של כל אדם בטבלה יהיו ייחודיים. בשיטה זו תמנע הזנה כפולה של שם כגון Amy Rosenthal, אך הדבר לא ימנע הזנה של אנשים נוספים בשם Amy לטבלה. להגדרת מספר שדות עבור המפתח העיקרי, בחר בשורות הרצויות בעת לחיצה ממושכת על <Ctrl>.



תרשים 1.12 הגדרת עמודה כמפתח העיקרי של הטבלה tblCustome באמצעות SQL Enterprise Manager.

גישה למסד נתונים באמצעות SQL Query Analyzer

תוכל לשלוח פקודות SQL ל- SQL Server באמצעות תוכנית השירות SQL Query Analyzer (שנקראה בעבר ISQLW). באמצעות תוכנית שירות זו, תוכל להפעיל שאילתות SQL ואף לבצע פעולות נוספות ברשומות, כגון עדכון, מחיקה והוספה. תוכל גם לבצע באמצעות SQL Query Analyzer משימות מתוחכמות לניהול השרת ומסדי נתונים, כגון יצירת מסדי נתונים, טבלאות, תצוגות ושגרות מאוחסנות.

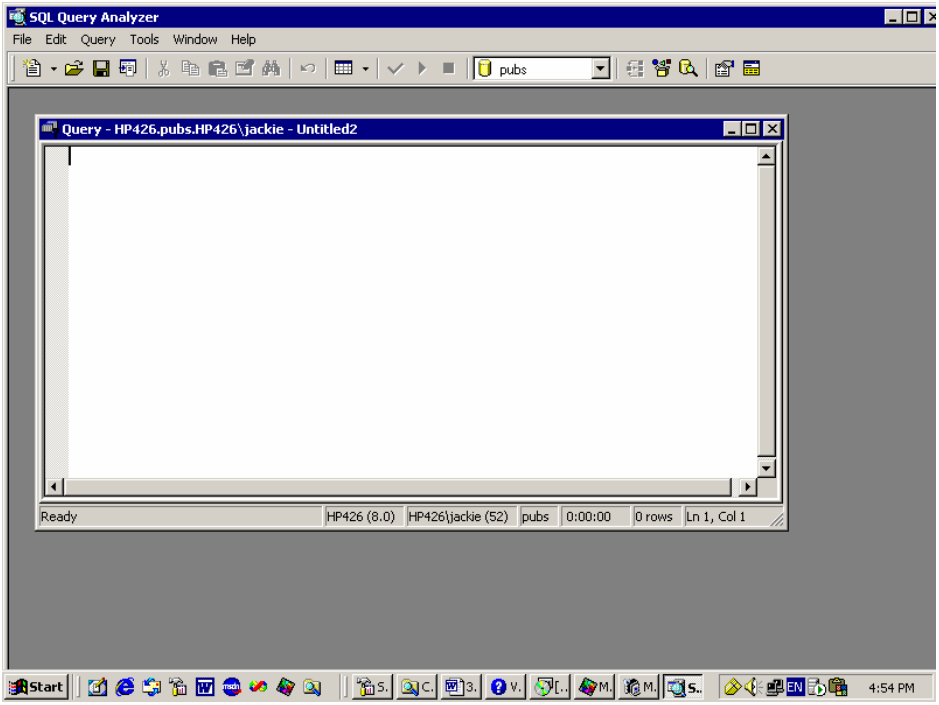
אם אתה מכיר את תחביר SQL, יהיה לך קל להשתמש ב- SQL Query Analyzer. עם זאת, שליטה מעמיקה בתוכנית שירות זו עשויה להיות קשה, ולכן דוגמאות מורכבות רבות בפרק זה מבוצעות באמצעות התוכנית הגרפית המקבילה של SQL Query Analyzer, הלא היא SQL Enterprise Manager.

לשליחת פקודות למסד הנתונים באמצעות SQL Query Analyzer, הפעל את SQL Query Analyzer מקבוצת התוכניות Microsoft SQL Server, או מתפריט Tools של SQL Server Enterprise Manager. התוכנה SQL Query Analyzer תציג את תיבת הדו-שיח Connect to SQL Server. לאחר מכן, בחר את השרת שברצונך להתחבר אליו, הקלד שם משתמש וסיסמה ולחץ Connect. כעת יוצג החלון הראשי של SQL Query Analyzer, כפי שמוצג בתרשים 1.13. כאשר מופעלת פקודה, נוספת חלונית מרובת כרטיסיות לתוצאות התצוגה, הודעות, סטטיסטיקות ותוכניות שונות. נציג את כרטיסיית ההודעות והתוצאות בדוגמאות נוספות בספר.

לאחר הפעלת SQL Query Analyzer, תוכל לשלוח פקודות למסד הנתונים ב-SQL. כדי לוודא שהיישום פועל כשורה, בדוק את חיבורי מסד הנתונים לפני שתנסה לעשות פעולה כלשהי אחרת. תוכל לעשות זאת באמצעות שימוש במסד הנתונים pubs אשר מגיע עם SQL Server, בדרך זו:

1. הורה ל- SQL Server באיזה מסד נתונים ברצונך להשתמש על ידי ביצוע פקודת SQL USE, והקלד את שם מסד הנתונים הרצוי. בחלון Query של SQL Query Analyzer, הקלד:

```
USE pubs
```



תרשים 1.13 החלון הראשי של SQL Query Analyzer.

הערה

ניתן להזין פקודות SQL באותיות רישיות או באותיות קטנות, אך נהוג להזין אותן באותיות רישיות. בפרק זה נשתמש בנוהג זה.

2. הפעל את הפקודה באמצעות הקשה על **F5** או באמצעות לחיצה על הלחצן Executy Query הירוק שבסרגל המשימות. SQL Query Analyzer יעבור לכרטיסיה Messages וכך תוכל לצפות בתגובה של SQL Server. אם הפעולה מתבצעת כשורה, תופיע ההודעה:

   The command(s) completed successfully.

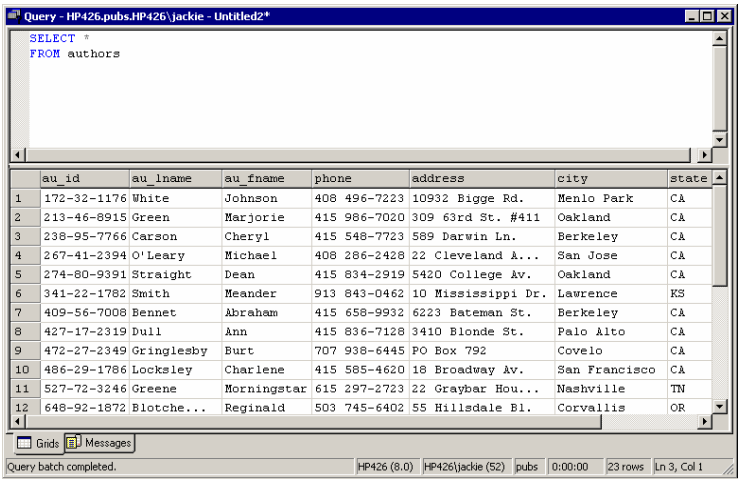
3. מחק את חלוניות החלון על ידי בחירה ב-**Edit**, **Clear Windows** (או באמצעות הקיצור **Ctrl+Shift+Delete**) בכל חלונית.

4. כעת בצע שאילתה פשוטה על מסד הנתונים pubs. לאחר מכן, ודא שמוחזרים נתונים. הקלד את קוד SQL שלהלן בחלון Query:

```
SELECT *
FROM authors
```

5. הפעל את השאילתה באמצעות הקשה על **F5**, או באמצעות לחיצה על הלחצן **Execute Query** המסומן בירוק בסרגל הכלים. אם הפעולה התבצעה כשורה, SQL Query Analyzer יחזיר את תוצאות השאילתה בכרטיסיה Grids, כפי שמוצג בתרשים 1.14. הכרטיסיה Messages מציגה את מספר השורות שהחזירה השאילתה, כמו לדוגמה,

(23 row(s) affected)



	au_id	au_lname	au_fname	phone	address	city	state
1	172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA
2	213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA
3	238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA
4	267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland A...	San Jose	CA
5	274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA
6	341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS
7	409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA
8	427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA
9	472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA
10	486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA
11	527-72-3246	Greene	Morningstar	615 297-2723	22 Graybar Hou...	Nashville	TN
12	648-92-1872	Blotche...	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR

תרשים 1.14 תוצאות שאילתת בדיקה שבוצעה על מסד הנתונים pubs, כפי שמוצג בחלון SQL Query Analyzer.

טיפ

בנוסף להפעלת כל פקודות התסריט המוצגות כרגע בחלון Query, תוכל להפעיל שורה אחת או יותר על ידי בחירה (סימון) של השורות הרצויות והקשה על F5 או לחיצה על Execute Query הירוק שבסרגל הכלים. בצורה זו תוכל לחזור בקלות על חלקי פקודות אם שינית אותן.

הצגת כל האובייקטים במסד הנתונים באמצעות sp_help

ב- SQL Server, ניתן לצפות בכל האובייקטים הזמינים בכל מסד נתונים. המערכת מספקת יכולת זו באמצעות **שגרה מאוחסנת** (stored procedure) – קטע קוד המאוחסן ומופעל בשרת.

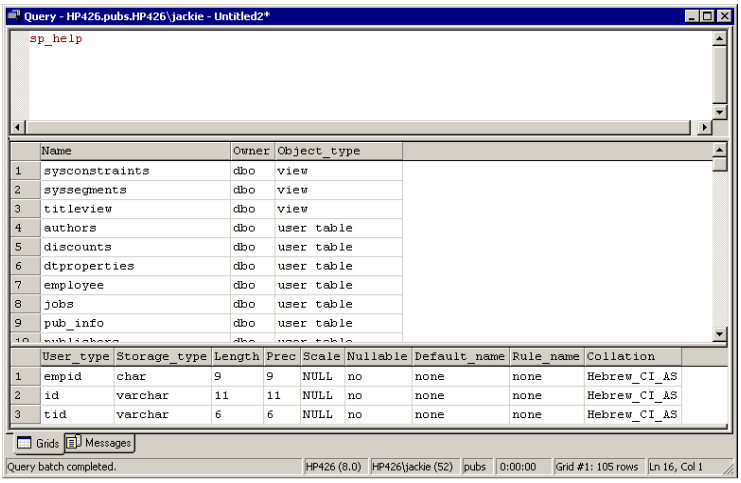
השגרה המאוחסנת sp_help מאפשרת לעיין במסדי נתונים. תוכל להפעיל את sp_help באותה הדרך שבה אתה מפעיל כל שאילתת SQL – על ידי הקלדתה בחלון Query של SQL Query Analyzer.

בצע את הפעולות שלהלן כדי לקבל את רשימת האובייקטים במסד הנתונים באמצעות שימוש ב- sp_help:

1. עבור לחלון Query ב- SQL Query Analyzer; במידת הצורך, מחק את תיבת השאילתה באמצעות הקשה על **Ctrl+Shift+Delete**.
2. בתיבת השאילתה, הקלד:

```
sp_help
```

3. הפעל את הפקודה. SQL Server יגיב ויספק רשימת אובייקטי מסד נתונים הדומה לרשימה המוצגת בתרשים 1.15.



תרשים 1.15 תגובה טיפוסית של sp_help למסד הנתונים pubs.

הערה

ב- SQL Server תוכל ליצור שגרות מאוחסנות משלך. למידע נוסף אודות נושא זה, ראה סעיף "יצירה והפעלת שגרות מאוחסנות" בהמשך פרק זה. בנוסף, השגרות המאוחסנות שאתה יוצר הן לרוב מקומיות ונמצאות במסדי נתונים נפרדים, אך המערכת מספקת שגרות מאוחסנות נוספות הזמינות עבור כל מסד נתונים ב-SQL Server. לקבלת דוגמה נוספת של שגרת מערכת מאוחסנת, ראה סעיף "הצגת טקסט של שגרה מאוחסנת או של תצוגה קיימת" בהמשך פרק זה.

הפעלת מסד נתונים קיים

תוכל להפעיל מסד נתונים מסוים באמצעות הפקודה USE ב- SQL Query Analyzer. כאשר אתה מגדיר מסד נתונים מסוים על ידי הפקודה USE, כל פקודות SQL שתפעיל לאחר מכן יבוצעו על מסד נתונים זה. חשוב לזכור עובדה זו, מכיון שאתה עלול להפעיל בטעות פקודות לא נכונות על מסד נתונים שאינו נכון.

הערה

הפעלה בטעות של פקודות על מסד נתונים לא נכון היא סיבה אחת לכך שעליך להגדיר כברירת מחדל מסד נתונים כלשהו, ולא להתיר לשרת להשתמש במסד הנתונים "master" המוגדר כברירת מחדל. כלומר, למסד הנתונים master יש לשלוח אך ורק נתוני תצורת שרת בלבד. ניתן לקבוע ברירת מחדל למסד נתונים עבור כל התחברות לשרת. תוכל לשנות מסד נתונים זה בעת ההתחברות או לאחר מכן; הדרך לעשות זאת מתוארת בסעיף "יצירה ותחזוקה של התחברויות ומשתמשים" בהמשך פרק זה.

לדוגמה, בצע את הפעולות שלהלן כדי לעבור ממסד הנתונים "master" למסד הנתונים Novelty.

1. בחלון Query של SQL Query Analyzer, הקלד:

```
USE novelty
```

   או בחר בו מתיבת הרשימה של מסדי הנתונים אשר נמצאת בסרגל הכלים של חלון SQL Query Analyzer.

2. אם קיים מסד נתונים בשם Novelty, תופיע ההודעה הזו:

```
The command(s) completed successfully.
```

3. אם מסד הנתונים אינו קיים, תופיע ההודעה הזו:

```
Server: Msg 911, Level 16, State 1, Line 1
Could not locate entry in sysdatabases for database 'novelty'.
No entry found with that name. Make sure that the name is entered correctly.
```

זכור, אם תשכח את שם מסד הנתונים או את איות שמו, תוכל לאתרו ב- SQL Enterprise Manager או להציג רשימת שרתים זמינים באמצעות השגרה המאוחסנת sp_helpdp, אשר מחזירה מידע אודות מסד נתונים מסוים או אודות כל מסדי הנתונים.

שליחת פקודות SQL
ל- SQL Query Analyzer

באמצעות SQL Query Analyzer, ניתן להפעיל כל פקודת SQL על מסד נתונים מסוים. להפעלה באמצעות SQL Query Analyzer יש מספר יתרונות בהשוואה לשיטות אחרות לשליחת פקודות ל- SQL Server. זכור שהשימוש בתחביר SQL עשוי להיות קשה יותר מהשימוש ב- SQL Enterprise Manager – במיוחד אם אתה משתמש בפקודות חד-פעמיות, כגון פקודות ליצירת מסדי נתונים – אך היתרון בשימוש ב- SQL Query Analyzer טמון ביכולת האינטראקטיבית שלו. תוכנית השירות מגיבה במהירות לפקודות שאתה שולח. ל- SQL Query Analyzer יש מספר תכונות שאינן קיימות ב- SQL Enterprise Manager, כגון יכולת להפעיל שאילתות ושגרות מאוחסנות.

בקרת גישה לנתונים
באמצעות תצוגות מסדי נתונים

תצוגה (view) היא הגדרת שאילתה המאוחסנת במסד נתונים, באופן דומה להגדרת שאילתות במנוע מסד הנתונים Jet של מיקרוסופט, בכך שהיא מאחסנת הגדרה שנמצאת במסד הנתונים ומאפשרת ליישומי לקוח גישה לנתונים.

עליך להשתמש בתצוגות במצבים שבהם תרצה להעניק למשתמשים גישה למסדי נתונים, אך לא תרצה להעניק להם גישה ישירה לטבלאות. עם זאת, לאחר הגדרת תצוגה, היא מציגה למעשה טבלה וירטואלית שניתן להשתמש בה כבכל טבלה של מסד נתונים. לעובדה שתצוגה נראית בדיוק כמו טבלה עבור יישום לקוח יש מספר יתרונות. נבחן מה קורה כאשר משתמשים ניגשים לנתונים באמצעות תצוגות ולא בגישה ישירה דרך הטבלאות:

- ניתן לשנות את עיצוב הטבלה מבלי לשנות את התצוגות הקשורות אליה.
- ניתן להגביל את מספר השורות או העמודות המוחזרות על ידי התצוגה.
- ניתן לספק גישה פשוטה לנתונים המאוחזרים ממספר טבלאות על ידי שימוש בצירופים (joins) הנמצאים בתצוגה.

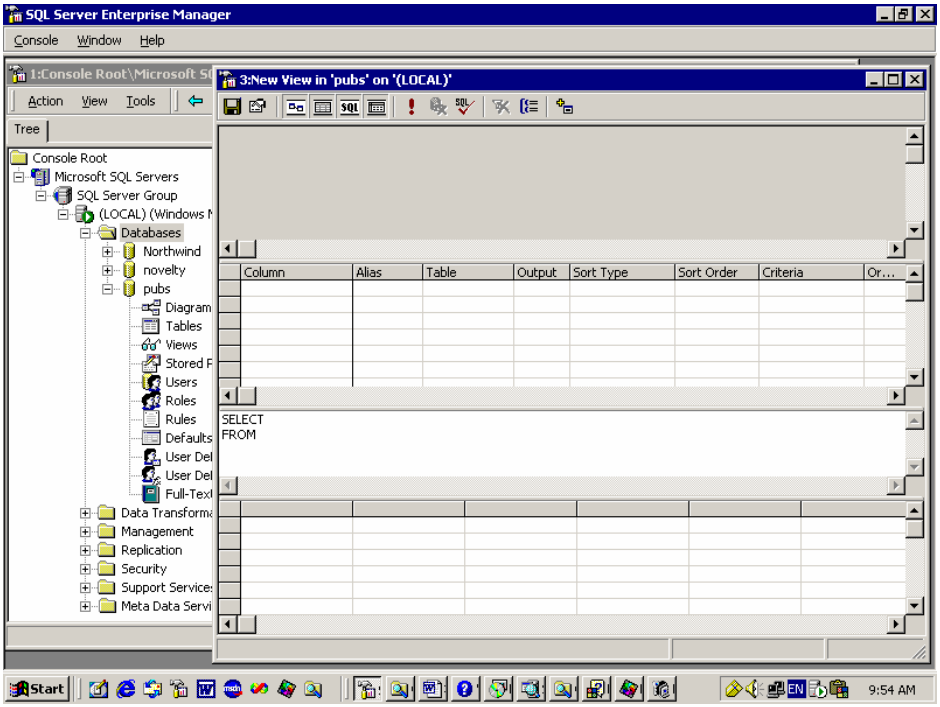
כדי לנצל את מלוא היתרונות של השימוש בתצוגות, עליך לנקוט באסטרטגיית אבטחה מתאימה עבור מסד הנתונים. פעולה זו תאפשר להצמיד הרשאות אבטחה לתצוגות במקום לטבלאות, כדי להקל על הענקת הרשאות לסוגים שונים של משתמשים. בנושא האבטחה נדון בסעיף "ניהול משתמשים ואבטחה באמצעות SQL Enterprise Manager" בהמשך הפרק.

יצירת תצוגות
ב- SQL Server Enterprise Manager

בדומה לאובייקטים רבים של מסד נתונים שניתן ליצור ב- SQL Server, תוכל ליצור תצוגות גם ב- SQL Query Analyzer או ב- SQL Server Enterprise Manager. בעיקרון, שתי הטכניקות דומות; ב- SQL Server Enterprise Manager הדבר מתבצע בצורה קצת יותר גרפית, ואילו ב- SQL Query Analyzer הדבר מתבצע בצורה אינטראקטיבית, כך שתוכל לבדוק בקלות את התצוגה לאחר שיצרת אותה.

בצע את הפעולות שלהלן כדי ליצור תצוגה ב- SQL Server Enterprise Manager.

1. מחלון Server Manager, לחץ לחיצה ימנית על הלחצן **Views** שבמסד הנתונים שברצונך ליצור בו תצוגה. למטרות דוגמה זו, השתמש במסד הנתונים pubs.
2. בחר **New View** מתפריט הקיצור. כעת יופיע החלון Design View, כפי שתראה בתרשים 1.16.



תרשים 1.16 יצירת תצוגה חדשה ב- SQL Server Enterprise Manager.

הערה

כאשר תציג תיבת דו-שיח זו בפעם הראשונה, תיאור תיבת הדו-שיח Design View יתחיל ב- New View ולא ב- Design View.

טיפ

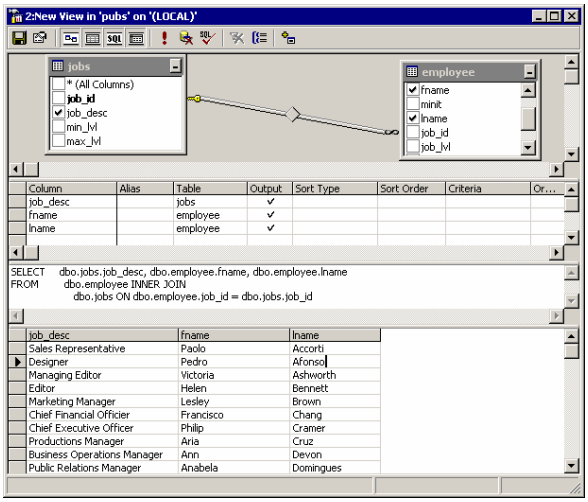
ניתן להשתמש בכלי הגרפי לעיצוב תצורות גם לצורך עיצוב שאילתות. ניתן לגשת לכלי זה על ידי לחיצה ימנית על טבלת מסד נתונים, בחירת האפשרות Open Table מתפריט הקיצור ובחירת Query. עם זאת, לא תוכל לשמור את השאילתה, מכיון ששאילתה נפרדת אינה מהווה אובייקט מסד נתונים ב- SQL Server, אך תוכל להשתמש ב- Query Designer ולפתח ולבדוק שגרות מאוחסנות וגם לאחסן ערכה מסוימת של נתונים ממסד הנתונים.

3. הצג את תיבת הדו-שיח Add Table באמצעות לחיצה על **Add Table** בסרגל הכלים Design View, או באמצעות לחיצה ימנית על החלונית Diagram (עליונה) ובחירת האפשרות **Add Table** מתפריט הקיצור.
4. בחר בטבלה jobs ולחץ **Add** (או לחץ לחיצה כפולה על הטבלה jobs), כדי להוסיף את הטבלה jobs לתצוגה.
5. בחר בטבלה employee ולחץ **Add** (או לחץ לחיצה כפולה על הטבלה employee), כדי להוסיף את הטבלה employee לתצוגה.
6. לחץ **Close** כדי לסגור את תיבת הדו-שיח Add Table.
7. בחר שדות מכל טבלה. סמן את השדה **job_desc** בטבלה jobs ואת השדות **fname** ו-**lname** בטבלה employee.
8. בדוק את התצוגה באמצעות לחיצה על **Run** (סימן קריאה) בסרגל הכלים Design View, או באמצעות לחיצה ימנית על אחת מחלוניות החלון ובחירת האפשרות **Run** מהתפריט הנפתח. התוצאות מוצגות בתרשים 1.17.
9. שמור את התצוגה החדשה באמצעות לחיצה על **Save** בסרגל הכלים Design View, או באמצעות לחיצה ימנית על אחת מחלוניות החלון ובחירת האפשרות **Save** מהתפריט הנפתח.

הערה

תוכל גם להקצות את הסיומת "_view" לשם של תצוגה (לדוגמה, SpecialCustomers_view). בצורה זו יהיה ברור כי אתה פועל על תצוגה ולא על טבלה. תוכל כמובן להקצות כל סיומת או להשתמש בכל שיטה אחרת לזיהוי תצוגות, או לא להשתמש באף שיטה כזו.

10. כאשר תוצג תיבת הדו-שיח Save As, הקלד בה את שם התצוגה ולחץ **OK**. תוכל להשתמש כמעט בכל שם רצוי, אך בדוגמה המופיעה בפרק זה נשתמש בשם התצוגה EmployeeJobs_view.



תרשים 1.17 תוצאות יצירה והפעלת תצוגה חדשה בחלון Design View של SQL Server Enterprise Manager.

לאחר יצירת התצוגה EmployeeJobs_view, יש ברשותך אובייקט שניתן לטפל בו כאילו היה טבלה פשוטה – אי אפשר להבחין כי זהו למעשה מיזוג בין שתי טבלאות. לכן, נוכל להשתמש בהצהרות SQL קצרות ופשוטות יותר המבוססות על התצוגה הזו, ונקבל עדיין מסד נתונים עם עיצוב תקין (מנורמל).

בדומה לכך, נוכל ליצור תצוגה המבוססת על חישובים או טיפול בנתוני הטבלה. לדוגמה, אם נרצה לאחזר שמות עובדים כשדה יחיד המשלב את השם הפרטי ואת שם המשפחה במבנה lname ו-fname, נוכל ליצור את התצוגה באמצעות הקוד שלהלן:

```
CREATE VIEW EmployeeNames_view AS
    SELECT lname + ', ' + fname AS Name
    FROM employee
```

שימוש בתצוגות ביישום מסחרי

תצוגה היא מבנה המספק שליטה טובה יותר באחזור נתונים ממסד נתונים של SQL Server. שליטה זו מוכיחה את עצמה במספר דרכים. על ידי הגבלת מספר שורות או עמודות המאוחזרות מהתצוגה, תוכל לשלוט בנתונים שמשתמש מסוים יכול לאחזר. בצורה זו תוכל לבצע מספר פעולות שימושיות, כגון יצירת קריטריון בחירה הידוע לך בלבד, או נעילת משתמשים מפני ערכות משנה מסוימות של נתונים על פי הרשאות האבטחה שלהם. תוכל לבצע פעולות כאלו מכיון שניתן לשייך כל אובייקט

במסד הנתונים – כולל טבלאות, תצוגות ושגרות מאוחסנות – עם משתמשים או קבוצות אבטחה מסוימות. במסד נתונים אשר מנצל תצוגות ושגרות מאוחסנות, הגישה הישירה לטבלאות מוגבלת בעיקרון למנהל מסד הנתונים; יישומי הלקוח מוגבלים ויכולים לגשת לתצוגות או לשגרות מאוחסנות בלבד, אשר אחראיות על אחזור נתונים מטבלאות מסדי נתונים.

העמודה Hide היא דוגמה ליישום המשתמש בטכניקה כזו. אם ערך העמודה Hide של רשומה נקבע ל-True, שורה זו לא תוחזר למשתמש בשום מקרה; השורה תסונן על ידי התצוגה האחראית על אחזור הנתונים ממסד הנתונים. ליישומי לקוח אין דרך לדעת שמשהו השתנה, מכיון שהם שולחים תמיד את הדרישות לאותה התצוגה.

הגישה למסדי נתונים באמצעות תצוגות ולא באמצעות גישה ישירה לטבלאות היא רכיב חשוב בכל התקנה מסחרית גדולה של מסד נתונים. בנוסף לאפשרות להגביל את מספר השורות והעמודות המאוחזרות, ההגנה על טבלאות מסד הנתונים באמצעות תצוגות מאפשרת לבצע שינויים בטבלאות מבלי לשנות את יישומי הלקוח.

את התהליך של הגנה על עיצוב מסד הנתונים מפני שינויים, עקב שינויים עסקיים, אפשר להרחיב באמצעות **רכיבי השכבה האמצעית** (middle-tier components). מבחינת התפיסה, רכיבים אלה דומים לתצוגות ושגרות מאוחסנות, בכך שהם מגינים על עיצוב מסד הנתונים מפני שינויים ביישום התוכנה, אך יש להם יתרון בהשוואה לתצוגות ושגרות המאוחסנות ב- SQL Server: קל יותר לתכנת אותם; הם מחזירים נתונים בצורת אובייקטים ולא כשורות ועמודות; והם אינם מוגבלים למערכת ניהול מסד נתונים אחת או לשפת תכנות אחת כלשהי.

יצירת תצוגות באמצעות SQL Query Analyzer Batch

ניתן ליצור תצוגות באמצעות SQL Query Analyzer. התהליך אינו מבוצע בצורה גרפית, אך הוא גמיש יותר מאשר יצירת תצוגות ב- Enterprise Manager. בצע את הפעולות שלהלן כדי ליצור תצוגה בטבלה tblEmployee אשר אינה מכילה את השדה (החסוי) Salary, באמצעות שימוש ב- SQL Query Analyzer.

1. הקלד את הקוד שלהלן בחלון Query Analyzer (אצווה זו כתובה בצורה שתיצור תצוגה גם אם אינה קיימת).

```
USE novelty
GO
DROP VIEW Employee_view
GO
CREATE VIEW Employee_view AS
   SELECT ID, FirstName, LastName, DepartmentID FROM tblEmployee
GO
SELECT * FROM Employee_view
```

2. הפעל את האצווה באמצעות הקשה על **F5**, או על ידי לחיצה על הלחצן **Execute Query** הירוק שבסרגל הכלים. התצוגה תיווצר ותופעל; התוצאות מוצגות בכרטיסיה Grids (או Results).

3. ודא שהתצוגה נוצרה. עבור ל- SQL Server Enterprise Manager ובחר בכרטיסיה **Views** עבור מסד הנתונים Novelty.

הערה

בחלק קודם של פרק זה, הראינו כיצד לעצב תצוגה באופן גרפי על ידי SQL Server Enterprise Manager – באמצעות לחיצה ימנית על **תצוגה** ובחירת האפשרות **Design View** מהתפריט הנפתח. תוכל גם לערוך את טקסט התצוגה באמצעות לחיצה ימנית על **תצוגה** ושינוי הטקסט המופיע בתיבת הדו-שיח.

הקוד הקודם, המהווה גם דוגמה ליצירת תצוגה ב- SQL Query Analyzer, מהווה גם דוגמה להפעלת אצווה (batch) ב- SQL Query Analyzer. אצווה זו אינה יוצרת רק את התצוגה, אלא גם עוברת למסד הנתונים הנכון ומפעילה את התצוגה בסיום יצירתה. תוצאה זו מאשרת כי התצוגה פועלת כצפוי.

תוכל ליצור אצוות כדי לפשט את תהליך היצירה של אובייקטים של מסד נתונים באמצעות שימוש ב- SQL Query Analyzer; ברוב המקרים, כאשר אתה יוצר אובייקטים של מסד נתונים, תרצה לבצע יותר פעולות. שימוש טיפוסי באצוות SQL הוא השמטת טבלה, יצירת טבלה והכנסת נתוני דגימה; שימוש נוסף הוא בדיקה האם קיים חשבון משתמש ולאחר מכן יצירת חשבון זה באמצעות שימוש בסיסמת ברירת המחדל; ויש שימושים רבים נוספים.

יצירה והפעלת שגרות מאוחסנות

תצוגות מעניקות שליטה גדולה יותר על הצורה בה נתונים מאוחזרים ממסד נתונים של SQL Server, אך קיימת טכניקה טובה אף יותר על ידי שימוש ב**שגרות מאוחסנות** (stored procedures). שגרה מאוחסנת דומה לתצוגה, אלא שהיא גם מאפשרת לבצע פעולות מורכבות יותר על נתונים. לדוגמה, שגרה מאוחסנת מאפשרת:

- לבצע חישובים בנתונים.
- לקבל או להחזיר פרמטרים.
- להטמיע לוגיקת יישום אשר דורשת מספר שלבים או שאילתות, באמצעות שימוש בשפת תכנות מוכוונת מסד נתונים.
- להחזיר נתונים בצורה קלה ויעילה יותר לתוכניות מצד הלקוח.

אף לא אחת מתכונות אלו זמינה בתצוגות (Views).

מבחינה מסוימת, עדיף להתייחס לשגרה מאוחסנת כאל סוג מיוחד של שגרה – ומכך שמה. שמה של השגרה המאוחסנת ניתן לה מכיון שהיא מאוחסנת במסד הנתונים עצמו ולא כחלק מהיישום הפועל במחשב הלקוח או בשרת היישום. על פי הרשימה הקודמת, שגרה מאוחסנת יכולה להיות שאילתה פשוטה בת שורה אחת, או שגרה אשר מבצעת סדרה של שאילתות ופעולות מורכבות לפני החזרת התוצאה.

שגרות מאוחסנות נכתבות בשפת תכנות מוכוונת מסדי נתונים המותאמת להן. שפה זו כוללת את רוב וכמעט כל מבני התכנות הדרושים, אם כי לא תמיד, למרות שלפעמים התחביר מיושן. שפת התכנות של SQL Server נקראת Transact-SQL או בקיצור T-SQL.

הערה

על פי הצהרת מיקרוסופט, גרסאות עתידיות של SQL Server תאפשרנה כתיבת שגרות מאוחסנות בכל שפה הנתמכת על ידי פלטפורמת .NET (כגון, VB.NET), ולא רק ב- Transact-SQL. עובדה זו תקל על מפתחים לעבור מצורה אחת של פיתוח יישום לצורה אחרת, מבלי ללמוד תחביר של שפת תכנות חדשה.

סעיף זה אינו מתאר בפירוט את כל הפקודות הזמינות בתכנות שגרות מאוחסנות, אך הוא מספק מידע בסיס המתאר כיצד הן פועלות, מדוע הן שימושיות וכיצד תוכל לשלב אותן ביישומים המבוססים על SQL Server.

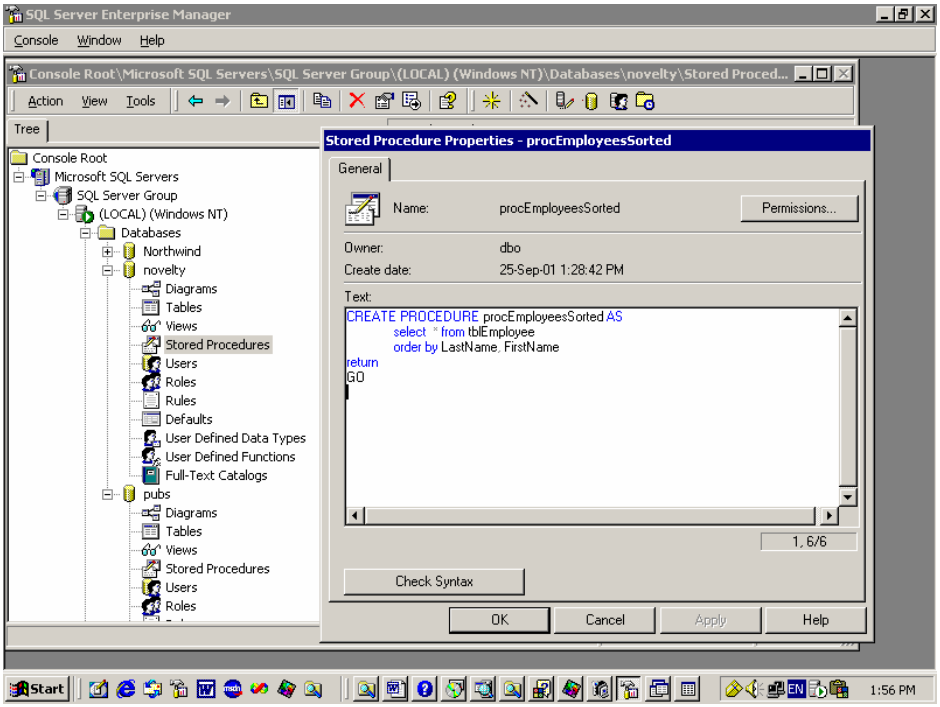
יצירת שגרות מאוחסנות באמצעות SQL Server Enterprise Manager

ניתן ליצור שגרות מאוחסנות באמצעות Enterprise Manager של SQL Server על ידי ביצוע הפעולות שלהלן.

1. בחלון Microsoft SQL Server MMC של SQL Server Enterprise Manager, לחץ לחיצה ימנית על הרכיב **Stored Procedures** במסד הנתונים בו אתה עובד. למטרות דוגמה זו, השתמש במסד הנתונים pubs.
2. מהתפריט הנפתח, בחר **New Stored Procedure**. כעת יוצג החלון Stored Procedure Properties.

הערה

נראה כי לחלון Stored Procedure Properties יש גודל קבוע, אך ניתן לשנות את גודלו מהשוליים או מהפינות שלו, בדומה לחלונות אחרים שניתן לשנות את גודלם. לפיכך, תוכל לשנות את גודל החלון על פי רצונך לפי כמות הטקסט המוצג בו.



תרשים 1.18 יצירת שגרה מאוחסנת באמצעות SQL Server Enterprise Manager.

3. כתוב את טקסט השגרה, כפי שמודגם בתרשים 1.18.
4. לאחר סיום כתיבת השגרה, לחץ **OK** בתחתית החלון Stored Procedure Properties.

הפעלת שגרה מאוחסנת מתוך SQL Enterprise Manager

ניתן להפעיל שגרות מאוחסנות (וכן תצוגות ופקודות SQL נוספות) מתוך SQL Server Enterprise Manager. פעולה זו שימושית כאשר תרצה לבדוק שגרות או תצוגות שיצרת. בצע את הפעולות שלהלן כדי לבדוק שגרה מאוחסנת:

1. בחר באפשרות **SQL Query Analyzer** מתפריט Tools של SQL Server Enterprise Manager. כעת יופעל היישום SQL Query Analyzer.
2. בחלון Query, הקלד את שם השגרה המאוחסנת שברצונך להפעיל. לדוגמה, כדי להפעיל את השגרה המאוחסנת אותה יצרת בדוגמה הקודמת, הקלד:

ProcEmployeesSorted

3. הפעל את השאילתה באמצעות הקשה על **F5**, או באמצעות לחיצה על הלחצן **Execute Query** הירוק שבסרגל הכלים SQL Query Analyzer. השגרה תבוצע ותחזיר ערכת תוצאות (אם קיימים נתונים בטבלה) בכרטיסיה Grids (או Results).
4. בחר ברכיב **Stored Procedure** בחלון SQL Servers MMC של Enterprise Manager. ודא שהשגרה המאוחסנת החדשה נוצרה. ייתכן שתצטרך ללחוץ Refresh בסרגל הכלים כדי לאלץ עדכון של חלון Servers MMC.

ניתן להפעיל את השגרה המאוחסנת גם בצורה ישירה, באמצעות הפעלת SQL Query Analyzer – **אינך חייב להפעיל אותה מתוך SQL Server Enterprise Manager**.

יצירת שגרות מאוחסנות באמצעות SQL Query Analyzer

הפעולות ליצירת שגרות מאוחסנות ב- SQL Query Analyzer כמעט זהות לחלוטין לפעולות המבוצעות ב- SQL Enterprise Manager.

הערה

ודא שאתה יוצר את השגרה המאוחסנת במסד הנתונים Novelty. קל לשכוח לעבור למסד הנתונים הנכון (באמצעות הפקודה USE או תיבת הרשימה ב- SQL Query Analyzer) לפני הפעלת פקודות על מסד הנתונים. יצירת שגרות מאוחסנות ב- SQL Server Enterprise Manager מפחיתה את הסיכוי לטעות מעין זו.

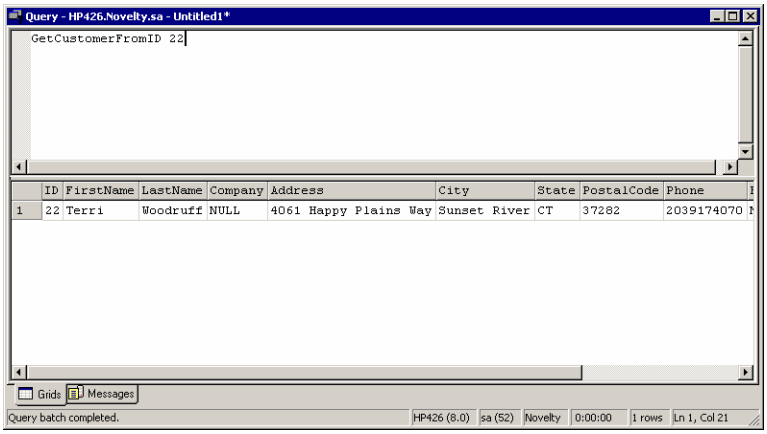
ליצירת שגרה מאוחסנת ב- SQL Query Analyzer, בצע את הפקודה Create Procedure.

1. ב- SQL Query Analyzer, הקלד את הקוד שלהלן בחלון Query:

```
CREATE PROCEDURE GetCustomerFromID
    @custID int
    AS
    SELECT * FROM tblCustomer
    WHERE ID = @custID
```

2. קוד זה יוצר שגרה מאוחסנת בשם GetCustomerFromID. השגרה מקבלת פרמטר – @custID, ומחזירה רשומת לקוח אשר מתאימה לארגומנט @custID. השדה ID הוא המפתח הראשי של tblCustomer, ולכן השגרה תחזיר תמיד אפס רשומות או רשומה אחת.
3. הפעל את הפקודה ליצירת שגרה מאוחסנת.
4. חזור לחלון Query והפעל את השגרה המאוחסנת כדי לבדוק אותה. להפעלת השגרה, נסה לאחזר רשומה מהטבלה על ידי הקלדת הקוד:

```
GetCustomerFromID 22
```



תרשים 1.19 רשומת לקוח יחיד המוחזרת על ידי השגרה המאוחסנת GetCustomerFromID ב- SQL Query Analyzer.

SQL Server מגיב בהחזרת הרשומה עבור לקוח עם ID 22, כפי שמוצג בתרשים 1.19. אם תזין זיהוי ID של לקוח שונה כערך פרמטר בשגרה המאוחסנת, תקבל רשומה שונה.

כמובן שהשגרה מחזירה נתונים רק אם קיימים נתונים מתאימים בטבלה.

הערה

כעת מומלץ לטעון נתונים למסד הנתונים. ניתן להוריד תסריט בשם CustomerData.sql מהכתובת http://www.awprofessional.com/titles/0672323435. תסריט זה יטען נתוני לקוח למסד הנתונים Novelty. ניתן להוריד גם תסריטים נוספים אשר ימלאו נתונים בטבלאות מסדי נתונים אחרים.

הערה

הקובץ ReadMe.txt בספריה Chapter1\Database Scripts\ מכיל את רשימת התסריטים הדרושים ליצירת כל הטבלאות ולמילוי נתונים בטבלאות. יש להקפיד על הסדר בהפעלתם.

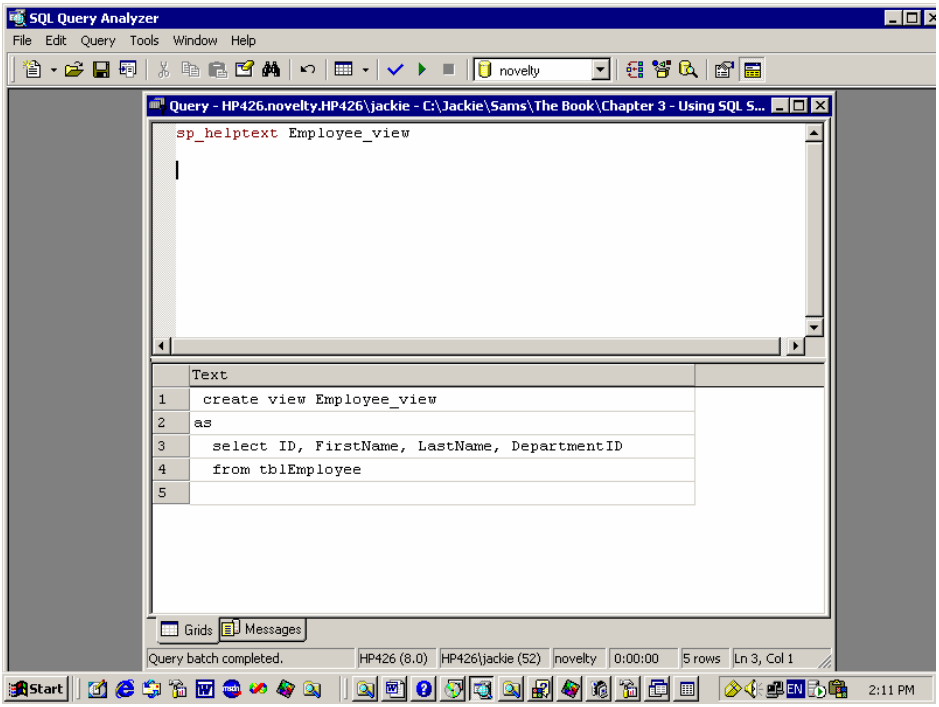
הצגת טקסט של שגרה מאוחסנת או של תצוגה קיימת

על ידי השגרה המאוחסנת sp_helptext, תוכל להציג קוד תצוגה או שגרה מאוחסנת. להצגת נתונים אלה, הקלד את הפקודה **sp_helptext** ואחריה את שם אובייקט מסד הנתונים הרצוי. מעבד SQL Server יחזיר את הטקסט המלא של התצוגה או השגרה המאוחסנת. לדוגמה, כדי לצפות בקוד התצוגה Employee_view שיצרת בתצוגות על פי הסעיף הקודם, בצע את הפעולות האלו:

1. בחלונית SQL Query Analyzer, הקלד:

```
sp_helptext Employee_view
```

2. הפעל את השגרה המאוחסנת באמצעות הקשה על **F5**, או באמצעות לחיצה על הלחצן **Execute Query** הירוק שבסרגל הכלים SQL Query Analyzer. קוד השגרה המאוחסנת יוחזר כתוצאה בכרטיסיה Grids, כפי שמוצג בתרשים 1.20.



תרשים 1.20 הצגת טקסט תצוגה באמצעות השגרה המאוחסנת sp_helptext.

יצירת מזניקים

מזניק (trigger) הוא סוג מיוחד של שגרה מאוחסנת אשר מופעלת בכל פעם שמתבצעת גישה לטבלה מסוימת. ניתן לדמות מזניקים לשגרות אירוע (event procedures) אשר מופעלות בכל פעם שהנתונים מתעדכנים, נמחקים או מתווספים לטבלה.

בעיקרון, השימוש במזניקים מיועד למקרים שעליך לבצע משהו מסובך בנתונים בתגובה לגישה כלשהי אליהם. לדוגמה, תוכל להשתמש במזניק כדי להוסיף רשומה ליומן בכל פעם שנתונים מסוימים משתנים, או ליצור ערך ברירת מחדל מורכב עבור שדה ברשומה חדשה, על פי שאילתות בטבלה אחת או יותר.

אל תשתמש במזניקים להגדרות ברירת מחדל פשוטות; לשם כך, השתמש בפקודות ברירת המחדל או בתכונות. אל תשתמש בהם לשמירה על שלמות הנתונים; לשם כך, השתמש ב- SQL Server's inherent referential intergrity constraint. אם עליך לבצע פעולה שאינה אפשרית באמצעות התכונות של SQL Server, עליך לשקול שימוש במזניק (trigger).

לדוגמה, תוכל להשתמש במזניקים כדי לספק ערך ייחודי בעמודה אשר ישמש כמפתח העיקרי של הרשומה; טקטיקה זו של שימוש ב- Microsoft Access Upsizing Tools מכילה מזניק ליצירת מפתח עיקרי אקראי עבור כל רשומה. תוכל גם להשתמש למטרה זו בעמודות זיהוי, כפי שתואר קודם לכן בפרק זה. להלן דוגמה למזניק מעין זה:

```
CREATE TRIGGER tblCustomer_ITrig ON dbo.tblCustomer
FOR INSERT
AS
DECLARE @randc int, @newc int

SELECT @randc = (SELECT convert(int, rand() * power(2, 30)))
SELECT @newc = (SELECT ID FROM inserted)
UPDATE tblCustomer SET ID = @randc WHERE ID = @newc
```

הערה

כדי שכל אחד מהמזניקים האלה יפעל כשורה ויוכל לעדכן את העמודה ID, עליך לאפס את העמודה ID באופן שלא תצוין כעמודת זיהוי. לשם כך, עליך לחזור לתיבת הדו-שיח Design Table ולקבוע את ערך התכונה Identity של העמודה ID ל-"No".

הדרך ליצירת מספר אקראי לזיהוי ייחודי של רשומה היא הטכניקה הפשוטה ביותר ליצירת מפתח עיקרי. עם זאת, יש לה שני חסרונות. המפתחות העיקריים נוצרים ללא סדר ברור, דבר שעשוי להיראות כבעיה חזותית בלבד. אולם, אם תנסה ליצור מערכת חשבוניות, רצוי לדעת בבירור כי חשבונית 20010 עוקבת לחשבונית 20009.

הבעיה החמורה יותר היא העובדה, שאין הבטחה כי המפתח העיקרי שנוצר באופן אקראי יהיה אכן ייחודי. הסיבה לכך היא שהמזניק אינו בודק אם המספר האקראי שנוצר היה בשימוש ברשומה אחרת במסד הנתונים. עם זאת, הסיכוי ליצירה של אותו מספר שלם (integer) פעמיים הינו קטן. זכור כי סוג הנתונים integer של SQL Server הוא מספר שלם באורך ארבעה בתים, אשר יכול להכיל מספרים בין מינוס 2.1 מיליארד לפלוס 2.1 מיליארד.

תסריט עסקי 1.1:
יצירת מזניק לחיפוש מקורב

בראד ג'ונס – נשיא Jones Novelties, Incorporated – אישר את עבודתה של מפתחת מסדי הנתונים שלו. כעת היא מוכנה להתמודד עם הסוג הבא של בעיות הקשורות למסדי נתונים: שאילתות על שמות אנשים אשר כוללות שגיאות כתיב והומופונים (מילים שנשמעות דומה, אך נכתבות בצורה שונה), המהוות משימה קשה. האם חיפוש השם Smith הוא למעשה חיפוש אחר אדם ששמו האמיתי הוא Smythe? האם מדובר ב-McManus או ב-MacManus? כל אדם אשר שם המשפחה שלו קשה לביטוי, מכיר את הבעיה.

מפתחת מסדי הנתונים של ג'ונס הבינה כי היא עשויה להיתקל בבעיה מעין זו, ולכן החליטה לנצל פונקציית SQL Server לפתרון הבעיה. פונקציה זו, ()soundex, ממירה מילה לערך אלפאנומרי אשר מייצג את הצליל הבסיסי. אם ערך soundex של השם יישמר בזמן יצירתו, ניתן יהיה לבצע חיפוש בשאילתה לפי ערך soundex של השם. השאילתה תחזיר רשומות נוספות אשר יתאימו לקריטיון המבוקש.

כדי ליישם תכונה זו במסד הנתונים של Jones Novelities, יש לבצע מספר פעולות:

- שינוי הטבלה tblCustomer, כך שתכיל שדה חדש בשם LastNameSoundex.
- הפעלת שאילתת עדכון אשר תספק ערכי soundex לרשומות קיימות בטבלה tblCustomer.
- יצירת מזניק אשר ימלא את השדה LastNameSoundex בעת יצירה או שינוי של הרשומה.
- יצירת שגרה מאוחסנת אשר מחזירה את כל הלקוחות אשר שם המשפחה שלהם דומה לערך מסוים.

מפתחת מסד הנתונים תתחיל בשינוי הטבלה tblCustomer, באופן שתכיל ערך soundex עבור כל רשומה במסד הנתונים; היא מפעילה את הפקודה שלהלן מתוך SQL Query Analyzer:

```
ALTER TABLE tblCustomer ADD
    LastNameSoundex varchar(4) NULL
```

לאחר מכן, המפתחת מפעילה פקודת עדכון אשר מלאה בערכי soundex ברשומות קיימות. פעולה זו יש לבצע פעם אחת בלבד. העדכון מבוצע באמצעות פקודות SQL הבאות שב- SQL Query Analyzer:

```
UPDATE tblCustomer

SET LastNameSoundex = soundex(LastName)

GO

SELECT LastName, LastNameSoundex

FROM tblCustomer

GO
```

אין צורך לכלול באצווה הצהרת SELECT לאחר העדכון, אך ההצהרה קיימת במקרה שמפתח מסדי הנתונים ירצה לוודא שהפעולה בוצעה ולצפות בנתונים שהשתנו.

כעת ניתן ליצור את המזניק שיוסיף ערך soundex לכל לקוח בעת הוספתו למסד הנתונים. צריך להקליד ב- SQL Query Analyzer את הקוד המזניק שלהלן:

```
CREATE TRIGGER trCustomerI

ON

tblCustomer

FOR insert, update

AS

    UPDATE tblCustomer

    SET tblCustomer.LastNameSoundex = soundex(tblCustomer.LastName)

    FROM inserted

    WHERE tblCustomer.ID = inserted.ID
```

הערה

SQL Server 2000 מאפשר להגדיר מספר מזניקים מאותו הסוג (Insert, Update, Delete) עבור טבלה אחת, אך סדר ההפעלה אינו בשליטתך המלאה. תוכל לציין איזה מזניק להפעיל ראשון ואיזה אחרון. כדי לוודא שהמזניק הקודם הופעל לאחר הוספת כל המזניקים האחרים ל-tblCustomer (כגון מזניק אשר מקצה ערך לעמודה ID), מבוצע ב- Query Analyzer הקוד שלהלן, לאחר יצירת המזניק:

```
sp_settriggerorder @triggername= 'trCustomerI', _

@order='last', @stmttype = 'INSERT'
```

מזניק זה נראה מורכב מעט יותר מהצפוי בשל הדרך בה מופעלים המזניקים. ככלל, מזניק מופעל פעם אחת בלבד, גם אם פעולת ההוספה, העדכון או המחיקה גרמו לו

להפעיל תהליך אצווה גדול מאוד אשר מעבד אלפי רשומות. כתוצאה מכך, המזניקים שכתבה מפתחת מסדי הנתונים חייבים להיות מסוגלים לטפל במספר בלתי מוגבל של רשומות.

המפתח לטיפול במספר מסוים של רשומות באמצעות מזניק הוא לבצע עדכון המבוסס על כל הרשומות האפשריות אשר השתנו על ידי השגרה שגרמה מלכתחילה להפעלת המזניק. כיצד יידע המזניק אילו רשומות הושפעו משגרה זו? למזניקים יש גישה למידע זה באמצעות הטבלאות הווירטואליות inserted ו-deleted. הטבלה הווירטואלית inserted מכילה את הרשומה או הרשומות שהוכנסו (או עודכנו) על ידי השגרה שהפעילה את המזניק; הטבלה הווירטואלית deleted מכילה נתונים שנמחקו על ידי השגרה שהפעילה את מזניק המחיקה.

מפתחת מסדי הנתונים של ג'ונס יצרה את המזניקים insert ו-update, ולכן ההפניה לרשומות בטבלה הווירטואלית inserted מתייחסת לכל הרשומות שהוספו או עודכנו, ללא תלות בדרך שבה נעשה הדבר. כל רשומה תקבל כמובן ערך soundex אשר נוצר עבורה על ידי המזניק.

לאחר כתיבת המזניק אשר יוצר ערך soundex עבור כל רשומה ב-tblCustomer, מפתחת היישומים בדקה את המזניק על ידי הוספת רשומה שלא תוחזר בשאילתה רגילה. בהנחה שקיימים כבר מספר אנשים בשם Smith במסד הנתונים, מספיק להוסיף את הפקודה הזו:

```
INSERT INTO tblCustomer (FirstName, LastName)
    VALUES ('Avigail', 'Smythe')
```

המפתחת יכלה לוודא שהמזניק יצר ערך soundex עבור רשומה זו על ידי ביצוע שאילתה מיידית לערך זה במסד הנתונים:

```
SELECT LastNameSoundex, LastName
FROM tblCustomer
WHERE LastName = 'Smythe'
```

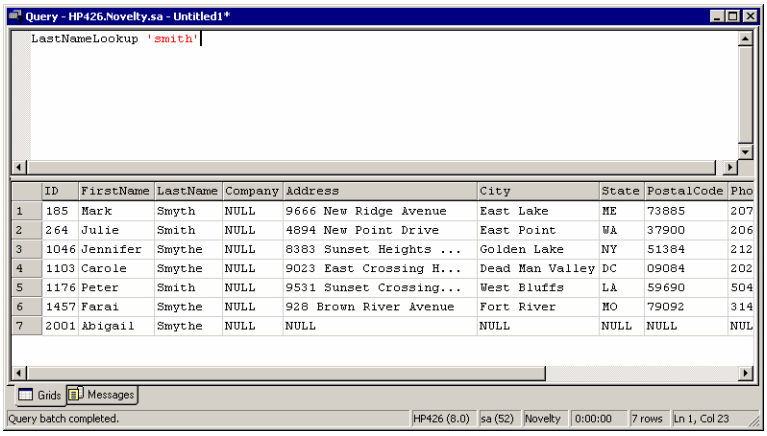
כעת, לאחר שווידאה כי המזניק פועל כשורה, ניתן היה ליצור שגרה מאוחסנת אשר מנצלת את העמודה LastNameSoundex. שגרה זו נוטלת פרמטר – שם המשפחה של האדם המבוקש – ומחזירה את כל האנשים הקיימים ב-tblCustomer אשר שמם דומה לשם המבוקש. להלן קוד השגרה המאוחסנת:

```
CREATE PROCEDURE LastNameLookup
    @name varchar(40)
AS
SELECT * FROM tblCustomer
WHERE soundex(@name) = LastNameSoundex
```

כעת אפשר לאחזר רשומות ממסד הנתונים על פי ערכי ה-soundex שלהם. לשם כך, המפתחת צריכה להפעיל את השגרה המאוחסנת LastNameLookup:

```
LastNameLookup 'smith'
```

לאחר ביצוע שגרה זו, SQL Query Analyzer מחזיר ערכת תוצאות המכילה כל אדם הנמצא במסד הנתונים, אשר שם המשפחה שלו דומה ל-Smith, כולל אנשים ששמם Smythes, כפי שמוצג בתרשים 1.21.



	ID	FirstName	LastName	Company	Address	City	State	PostalCode	Pho
1	185	Mark	Smyth	NULL	9666 New Ridge Avenue	East Lake	ME	73885	207
2	264	Julie	Smith	NULL	4894 New Point Drive	East Point	WA	37900	206
3	1046	Jennifer	Smythe	NULL	8383 Sunset Heights ...	Golden Lake	NY	51384	212
4	1103	Carole	Smythe	NULL	9023 East Crossing H...	Dead Man Valley	DC	09084	202
5	1176	Peter	Smith	NULL	9531 Sunset Crossing...	West Bluffs	LA	59690	504
6	1457	Farai	Smythe	NULL	928 Brown River Avenue	Fort River	MO	79092	314
7	2001	Abigail	Smythe	NULL	NULL	NULL	NULL	NULL	NUL

תרשים 1.21 ערכת התוצאות שהוחזרה על ידי השגרה המאוחסנת LastNameLookup.

ניהול משתמשים ואבטחה באמצעות SQL Server Enterprise Manager

אחת הסיבות החשובות לשימוש ב- SQL Server היא היכולת לטפל במספר משתמשים אשר מנסים לגשת לאותם הנתונים באותו הזמן. במצב זה קיימות מספר בעיות, כמו במקרים שבהם משתמשים ניגשים למידע בלעדי ושניים מהם או יותר מנסים לעדכן את אותה הרשומה באותו הזמן. ניתן לפתור בעיות רבות מסוג זה ואחרות על ידי הגדרות שרת.

תכונות האבטחה של SQL Server מאפשרות גמישות רבה בהקצאת הגישה לנתונים. ניתן להגדיר רשימת משתמשים לכל מסד נתונים שכל אחד מהם יכול לקבל ערכת הרשאות משלו. ערכת הרשאות (permission set) מאפשרת למשתמש לגשת לנתונים ולשנות אותם (אם תבחר בכך), ואף ליצור ולמחוק אובייקטים של מסד הנתונים.

תכונות האבטחה של SQL Server אף מאפשרות להקצות תפקידים למשתמשים מסוימים כדי לפשט את תהליך הקצאת ההרשאות. לדוגמה, תוכל ליצור תפקיד של מפתח עם הרשאת גישה לכל האובייקטים במסד הנתונים, ליצור תפקיד מנהל אשר רשאי לגשת למידע רגיש כגון נתוני משכורות ומכירות של החברה, או ליצור תפקיד של משתמש עבור משתמשים רגילים שלא יקבלו הרשאות מיוחדות לגישה למסד

הנתונים. אתה מחליט כיצד להקצות תפקידים למשתמשים, אך זכור שעליך להשתמש בנוהל התפקידים גם במסד נתונים פשוט, כדי לפשט את הטיפול בהרשאות גישה.

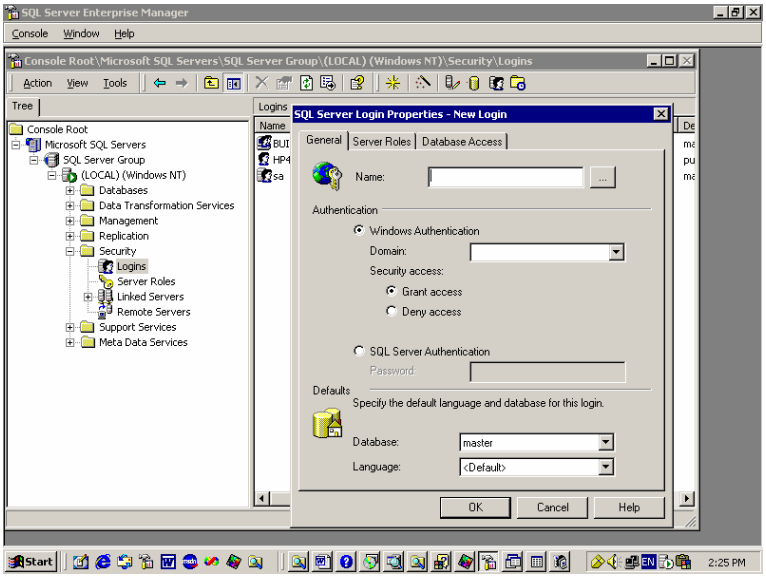
יצירה ותחזוקה של התחברויות ומשתמשים

כדי לקיים מערכת אבטחה מתוחכמת וגמישה יש להקצות למשתמשים זהויות נפרדות. SQL Server מאפשר זאת על ידי הגדרת התחברויות ומשתמשים. **התחברות** (login) מייצגת חשבון (או אדם) אשר ניגש ל- SQL Server. התחברויות משמשות ליצירת **משתמשים** (users). יצירה של משתמש מאפשרת להעניק הרשאות ייחודיות להתחברות אל מסד נתונים מסוים. תוכל גם להוסיף משתמשים לתפקידים ולהקצות להם קבוצה גדולה של הרשאות בו-זמנית.

אם עדיין לא הבנת מהו ההבדל בין התחברות (login) לבין משתמש (user), חשוב על כך בצורה הבאה: התחברות נוצרת ברמת השרת; משתמש נוצר ברמת מסד הנתונים. במילים אחרות, עליך לקבל הקצאת התחברות לפני שתוכל להפוך למשתמש של מסד נתונים מסוים.

כדי ליצור משתמש, צור התחברות עבור אדם.

1. ב- SQL Server Enterprise Manager, הרחב את התיקיה Security ב- SQL Server שברצונך לבצע בו את הפעולות. זכור שהתחברויות אינן שייכות למסד נתונים מסוים אלא לשרת עצמו.
2. לחץ לחיצה ימנית על הרכיב **Logins** ומתפריט הקיצור בחר **New Login**.
3. כעת תוצג תיבת הדו-שיח SQL Server Login Properties, כפי שמוצג בתרשים 1.22.
4. אם אתה יוצר התחברות SQL חדשה, סמן את התיבה **SQL Server Authentication** והקלד בתיבה **Login Name** את השם שיש להשתמש בו. תוכל אף להזין סיסמה עבור התחברות זו.



תרשים 1.22 תיבת הדו-שיח SQL Server Login Properties להתחברות חדשה.

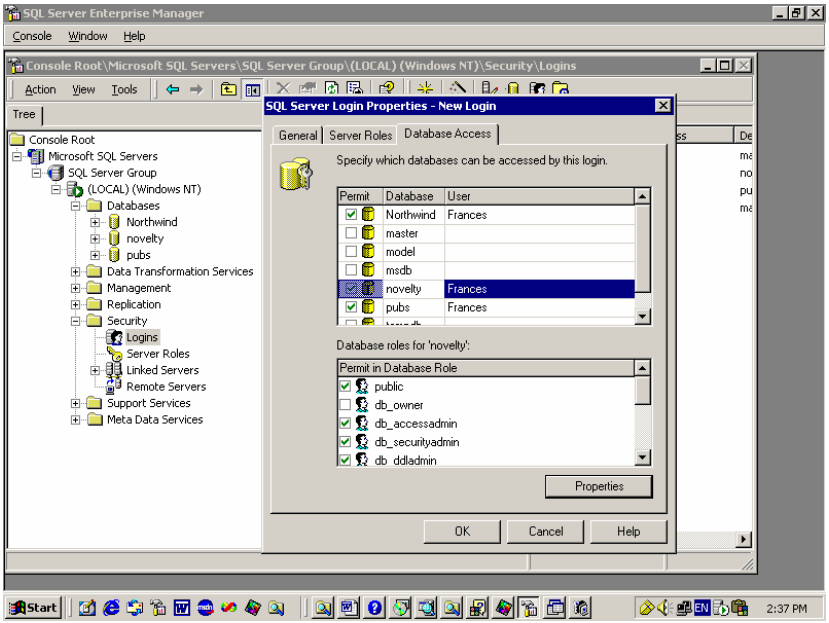
5. אם אתה ממפה חשבון Windows NT/2000 לחשבון SQL Server, סמן את התיבה **Windows Authentication** ולאחר מכן לחץ על לחצן ... שבצד שדה Name כדי לבחור בחשבון **Windows NT/2000** קיים.

הערה

כאשר אתה משתמש ב- SQL Server Authentication, שקול יצירה של שגרה ביישומי הלקוח שתיצור. השגרה תאפשר למשתמש ליצור ולשנות את סיסמתו. תוכל ליישם נוהל זה על ידי השגרה המאוחסנת sp_password. הדבר אינו נחוץ אם אתה משתמש ב- Integrated Security, מכיון שבמקרה כזה נעשה שימוש במנגנון האימות והסיסמה הסטנדרטי של Windows NT/2000.

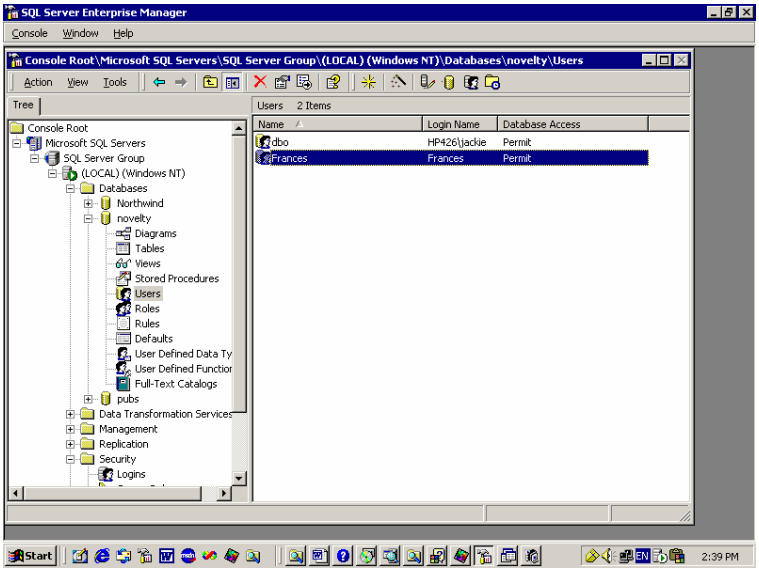
6. קבע מסד נתוני ברירת מחדל עבור התחברות זו על ידי בחירת האפשרות Novelty מתיבת הרשימה Database.

7. כעת תוכל להוסיף התחברות זו כמשתמש למסד נתונים מסוים על ידי מעבר לכרטיסיה Database Access בתיבת הדו-שיח SQL Server Login Properties. סמן את מסדי הנתונים שברצונך להקצות להם גישה עבור התחברות זו, וסמן את התפקידים המוקצים להתחברות בזמן הגישה למסד הנתונים, כפי שמוצג בתרשים 1.23.



תרשים 1.23 יצירת התחברות חדשה והוספת גישה למסד הנתונים.

8. לחץ **OK** בסיום הקצאת ההתחברות למשתמשים. כעת תיווצר ההתחברות וכל המשתמשים שיצרת עבור התחברות זו יתווספו למסד או למסדי הנתונים המתאימים. פעולה זו מוצגת מיידית בחלון Servers של Enterprise Manager, כפי שמוצג בתרשים 1.24.



תרשים 1.24 התחברות ומשתמש חדשים מוצגים בחלון SQL Server Enterprise Manager.

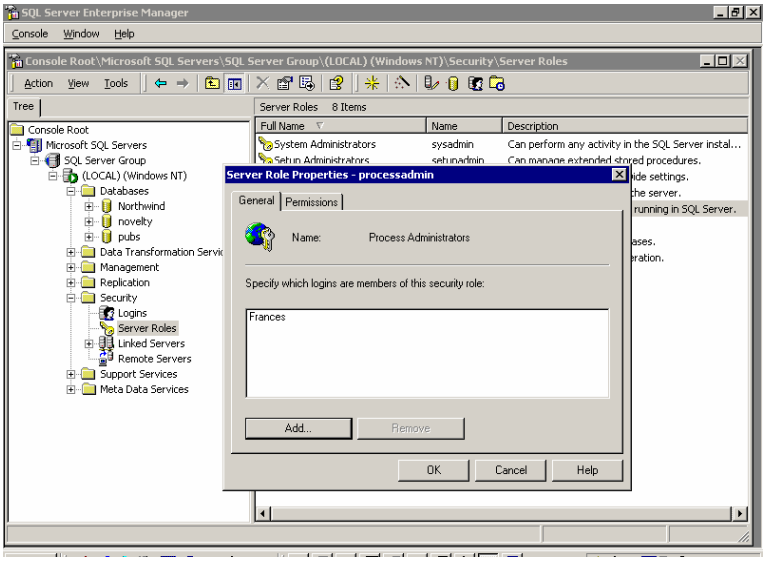
ניהול תפקידים באמצעות SQL Server Enterprise Manager

SQL Server 2000 משתמש בתפקידים (roles) ומקבץ בעזרתם משתמשים בעלי הרשאות דומות. כל משתמש אשר מתווסף לתפקיד, יורש את ההרשאות שניתנו לתפקיד זה. שינוי ההרשאות של תפקיד מסוים משנה גם את ההרשאות של כל המשתמשים המוקצים לו. בצורה זו, אם ברצונך להוסיף או לשלול מספר גדול של הרשאות ממשתמש מסוים, תוכל לעשות זאת בצורה קלה על ידי שינוי הקצאת התפקיד של המשתמש.

ל- SQL Server 2000 יש שני סוגי תפקידים – תפקידי שרת (Server roles) ותפקידי מסדי נתונים (Database roles). תפקידי שרת שולטים על הגישה לפעולות המשפיעות על SQL Server כולו. הן כוללות הפעלה והפסקה של השרת, הגדרת תכונות מתקדמות כגון שכפול, ניהול אבטחה ויצירת מסדי נתונים. תפקידי מסדי נתונים שולטים בגישה לפעולות ולנתונים במסד נתונים מסוים.

בצע את הפעולות שלהלן כדי להוסיף משתמש לתפקיד שרת ב- Enterprise Manager.

1. ב- Micorosft SQL Servers MMC של Enterprise Manager, הרחב את התיקיה Security של השרת שברצונך לבצע בו את השינוי ובחר ברכיב **Server Roles**. פעולה זו תציג את הערכה הקבועה של תפקידי השרת.
2. לחץ לחיצה ימנית על התפקיד שברצונך לשנות ומהתפריט הנפתח בחר **Properties**. לחילופין, תוכל ללחוץ לחיצה כפולה על התפקיד שברצונך לשנות. כעת תוצג תיבת הדו-שיח Server Role Properties, כפי שמוצג בתרשים 1.25.



תרשים 1.25 תיבת הדו-שיח Server Role Properties של Process Administrators Server Role.

3. להוספת התחברות לתפקיד זה, לחץ **Add** ובחר בהתחברויות הרצויות מתוך רשימת ההתחברויות הזמינות.

4. להסרת התחברות מתפקיד זה, בחר בהתחברות או בהתחברויות הרצויות מרשימת ההתחברויות החברות בתפקיד זה, ולחץ **Remove**.

5. לחץ **OK** כדי לסגור את תיבת הדו-שיח Server Role Properties.

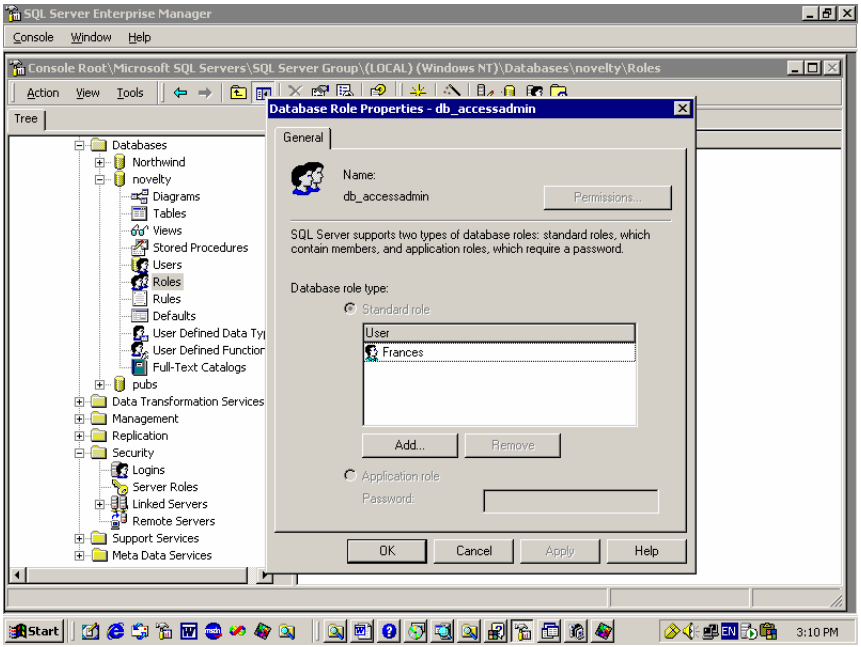
טיפ

ניתן להוסיף או להסיר משתמש מסוים מתפקיד שרת, באמצעות שימוש בכרטיסיה Server Roles של תיבת הדו-שיח SQL Server Login Properties, אשר תוארה קודם בסעיף "יצירה ותחזוקה של התחברויות ומשתמשים".

בצע את הפעולות שלהלן כדי להוסיף משתמש לתפקיד מסד נתונים ב- SQL Server Enterprise Manager.

1. ב- Microsoft SQL Servers MMC, בחר ברכיב Roles עבור מסד הנתונים שברצונך לערוך בו את השינוי. פעולה זו תציג את תפקידי מסד הנתונים הזמינים.

2. לחץ לחיצה ימנית על התפקיד שברצונך לשנות, ובתפריט הקיצור בחר **Properties**. לחילופין, תוכל ללחוץ לחיצה כפולה על התפקיד הרצוי. כעת תוצג תיבת הדו-שיח Database Role Properties, כפי שמוצג בתרשים 1.26.



תרשים 1.26 תיבת הדו-שיח Database Role Properties של התפקיד db_accessadmin.

3. להוספת משתמש לתפקיד זה, לחץ **Add** ובחר במשתמש או במשתמשים הרצויים מתוך רשימת המשתמשים הזמינים עבור מסד נתונים זה.

4. להסרת משתמש מתפקיד, בחר במשתמש או במשתמשים הרצויים מרשימת המשתמשים החברים בתפקיד, ולאחר מכן לחץ **Remove**.

5. לחץ **OK** כדי לסגור את תיבת הדו-שיח Database Role Properties.

הערה

SQL Server 2000 תומך אף בתפקידי מסד נתונים המוגדרים על ידי המשתמש, בנוסף לתפקידי מסד הנתונים שתיארנו. באמצעות תפקידים אלה, ניתן להתאים גישה לנתונים ופעולות במסד הנתונים. שים לב שהלחצן Permission אמור להיות מאופשר. ניתן לקבל מידע נוסף אודות תפקידי מסד נתונים המוגדרים על ידי המשתמש ב- SQL Server Books Online.

בדיקת תכונות אבטחה על ידי SQL Query Analyzer

הינך בוודאי סקרן לראות מה יקרה אם משתמש ינסה להשתמש באובייקטים של מסד נתונים שאין לו הרשאות גישה אליהם. SQL Query Analyzer מאפשר התחברות בכל שם משתמש, ולכן תוכל לבדוק את תכונות האבטחה על ידי שימוש בתיבת הדו-שיח Query, בצורה שלהלן.

1. ודא שלא סימנת את התיבה db_owner של המשתמש שיצרת במסד הנתונים Novelty שבתרשים 1.23.

2. התנתק מ- SQL Query Analyzer על ידי בחירת **Disconnect** או **Disconnect All** בתפריט **File**.

3. בחר **Connect** בתפריט File כדי להתחבר שוב ל- SQL Server באמצעות Query Analyzer. הפעם, אל תתחבר כ-sa, אלא בשם המשתמש שיצרת בדוגמה הקודמת. כעת התחל במסד הנתונים Novelty, או במסד הנתונים המוגדר כברירת מחדל עבור ההתחברות, ולא במסד ברירת המחדל בעת התחברות כ-sa.

4. נסה להפעיל שאילתה על טבלה שאין לך הרשאות גישה אליה. עשה זאת על ידי הפעלת הפקודה הבאה:

```
SELECT * FROM tblCustomer
```

SQL Server יגיב בהודעה זו:

```
Msg 229, Level 14, State 1

SELECT permission denied on object tblCustomer, database Novelty, owner dbo
```

הערה

השורות הקודמות וגם השורה המופיעה בשלב 5, מוצגות בהנחה שאתה משתמש במסד הנתונים Novelty שאנו משתמשים בו בהדגמות. אם אתה משתמש במסד נתונים שונה, ייתכן שלא תוכל לבצע את השינויים המתאימים.

5. כעת נסה להפעיל שגרה מאוחסנת על ידי הפעלת הפקודה שלהלן:

```
LastNameLookup 'smith'
```

SQL Server יגיב באחזור כל השמות אשר נשמעים כמו Smith ונמצאים ב-tblCustomer.

החלת תכונות אבטחה באמצעות SQL Query Analyzer

ניתן לבצע פעולות הקשורות לאבטחת מסדי נתונים על ידי SQL Query Analyzer. בעיקרון, עליך לעשות זאת אם ברצונך ליצור תכונות אבטחה של מסדי נתונים על ידי אותה אצוות SQL אשר יוצרת גם את מסד הנתונים עצמו. אם אתה מוסיף תמיד את הפקודות הקשורות לאבטחה אל אותה אצוות SQL אשר יוצרת את אובייקט מסד הנתונים, אתה עלול לשכוח להחיל תכונות אבטחה על אובייקטים של מסד הנתונים החדש.

עליך להשתמש בשגרה המאוחסנת sp_addlogin כדי ליצור התחברות SQL Server Authentication חדשה על ידי Query Analyzer. לדוגמה, כדי ליצור את ההתחברות Frances, בצע את הפקודה הזו:

```
sp_addlogin 'Frances'
```

כדי להקצות להתחברות Frances את הסיסמה 'stairmaster', הוסף את הסיסמה כארגומנט נוסף לשגרה sp_addlogin:

```
sp_addlogin 'Frances', 'stairmaster'
```

אם תרצה להוסיף התחברות Windows לחשבון Windows קיים במקום להוסיף התחברות SQL Server, השתמש בשגרה המאוחסנת sp_grantlogin בצורה דומה. עם זאת, שים לב שלא ניתן לציין סיסמה, מכיון שסיסמאות אינן חלק מהגדרת התחברות Windows של SQL Server, כי סיסמאות מטופלות באמצעות מנגנוני הסיסמה הסטנדרטיים של Windows.

השתמש בשגרה sp_adduser כדי להפוך את ההתחברות Frances למשתמש מסד הנתונים Novelty:

```
USE novelty
GO
sp_adduser 'Frances', 'Frannie'
GO
```

להצגת כל ההתחברויות ומסדי ברירת המחדל בהתקנת SQL Server באמצעות SQL Query Analyzer, בצע את פקודת SQL הבאה:

```
USE master
GO
SELECT name, dbname
FROM syslogins
```

להוספת משתמש לתפקיד Database, השתמש בשגרה המאוחסנת sp_addrolemember:

```
sp_addrolemember 'db_datawriter', 'Frannie'
```

ניתן להציג רשימה של כל תפקידי Database במסד נתונים מסוים, באמצעות השגרה המאוחסנת sp_helprole.

ניתן להחיל ולהסיר הרשאות לאובייקט מסוים במסד נתונים באמצעות פקודות SQL אלו: GRANT ו-REVOKE. הפקודה GRANT מעניקה למשתמש מסוים תפקיד אשר מאפשר לו גישה לאובייקט של מסד נתונים, והפקודה REVOKE מסירה את ההרשאה שניתנה. לדוגמה, כדי להעניק לחברי התפקיד public גישה מלאה ל-tblCustomer, בצע את פקודת SQL הבאה:

```
GRANT ALL
ON tblCustomer
TO public
```

לחילופין, תוכל להגביל את משתמשי אותה קבוצה מלבחור נתונים מ-tblCustomer. לשם כך, בצע את הפקודה GRANT עם האפשרות SELECT:

```
GRANT SELECT
ON tblCustomer
TO public
```

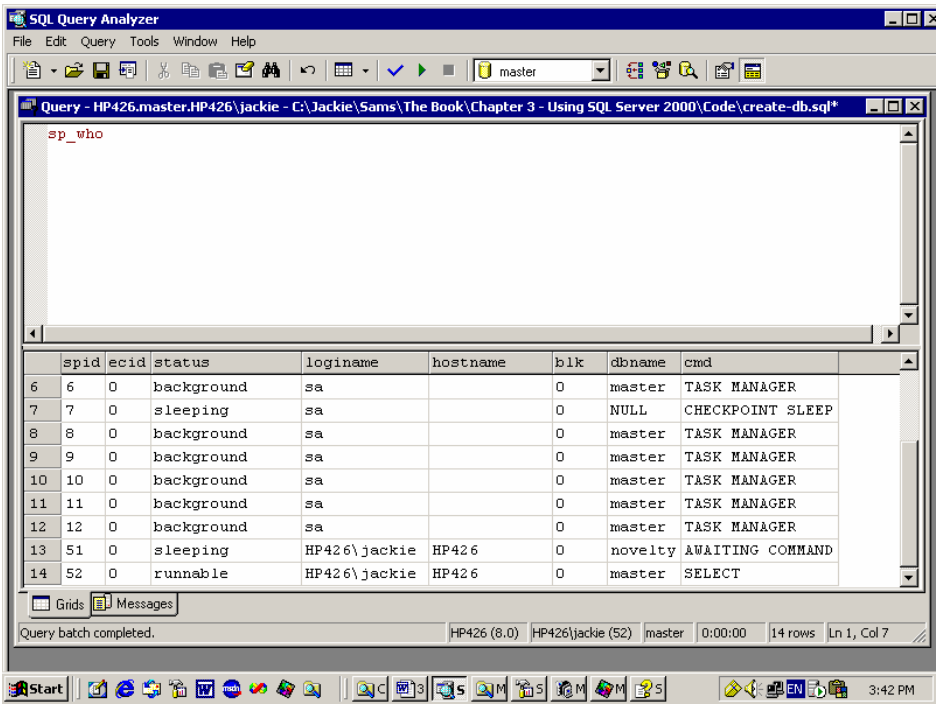
כדי למנוע הרשאות באובייקט מסד נתונים, השתמש בפקודה Revoke. לדוגמה, כדי למנוע הרשאת גישה ל-tblCustomer ממשתמשים אשר קיבלו בעבר את התפקיד public, השתמש בפקודת SQL הבאה:

```
REVOKE ALL
ON tblCustomer
TO public
```

תוכל גם להעניק או למנוע הרשאות לעדכון, בחירה, מחיקה והוספה של טבלאות ותצוגות. כמו כן, תוכל להעניק או למנוע הרשאות להפעלת שגרות מאוחסנות.

בדיקה מי מחובר באמצעות sp_who

ניתן לבדוק איזה משתמשים מחוברים למסד נתונים באמצעות השגרה המאוחסנת sp_who. השגרה מחזירה מידע אודות המשתמשים המחוברים למערכת ולאיזה מסדי נתונים הם מחוברים. כדי להשתמש ב-sp_who, פתח את החלון Query של SQL Query Analyzer. תקבל רשימת משתמשים המחוברים כעת, בדומה לזו המוצגת בתרשים 1.27.



תרשים 1.27 תוצאות הפעלת השגרה המאוחסנת sp_who.

ברשימת המשתמשים המחוברים ניתן לבצע מספר פעולות, כגון להוריד משתמש מהשרת (סיום session), כפי שמתואר בסעיף הבא.

סיום תהליך באמצעות הפקודה kill

הפקודה kill מאפשרת למנהל המערכת של SQL Server לסיים תהליך, כגון session של לקוח, או שבאפשרותה לשחרר מסד נתונים נעול. לעיתים עליך לעשות זאת כאשר החיבור של המשתמש הסתיים בצורה לא רגילה וברצונך להיפטר מה"שיירים" של התהליך ה"תקוע", או שברצונך לעשות זאת כאשר שגרת לקוח ביצעה נעילת נתונים שאינה משתחררת. אמנם, מצבים כאלה נדירים, אך הם מתרחשים מדי פעם ובמיוחד בסביבת פיתוח.

כדי להשתמש בפקודה KILL, עליך להפעיל תחילה את השגרה המאוחסנת sp_who (אם אתה מנסה לסיים חיבור של לקוח – session) או את השגרה sp_lock (אם אתה מנסה לשחרר נעילת מסד נתונים). שתי השגרות מחזירות את העמודה spid (שהיא הזיהוי, procedure ID, של התהליך). לאחר שידוע לך קוד הזיהוי, תוכל לסיים את התהליך באמצעות הפקודה KILL.

לדוגמה, נניח שאתה מפעיל את sp_who ומבחין בכך שישנו חיבור תקוע, עם קוד זיהוי (spid) מספר 10, של משתמש שידוע כי אינו עובד יותר בחברה. לסיום קוד זיהוי זה, בצע ב- SQL Query Analyzer את הפקודה הזו:

```
KILL 10
```

החיבור התקוע יסתיים מייד.

הערה

כדי לבדוק מה קורה במסד הנתונים, מומלץ להפעיל את sp_who מדי פעם, במיוחד בשלב הפיתוח.

הסרת אובייקטים ממסד הנתונים

המונח המקובל ב- SQL Server להסרת אובייקט ממסד הנתונים הוא **להשמיט** (drop) את האובייקט. כאשר אתה משמיט אובייקט ממסד הנתונים, הוא נמחק לצמיתות. אם ברצונך להשמיט טבלה המכילה נתונים, הן מבנה הטבלה והן הנתונים שמופיעים בה יימחקו לצמיתות.

להשמטת אובייקט מסד נתונים (כמו טבלה) ב- SQL Server Enterprise Manager, עליך ללחוץ על האובייקט לחיצה ימנית ולבחור באפשרות Delete מהתפריט הנפתח. האובייקט יימחק.

להשמטת אובייקט מסד נתונים מ- SQL Query Analyzer, השתמש בפקודה DROP. לדוגמה, בצע את הפקודה שלהלן כדי להשמיט את הטבלה tblCustomer:

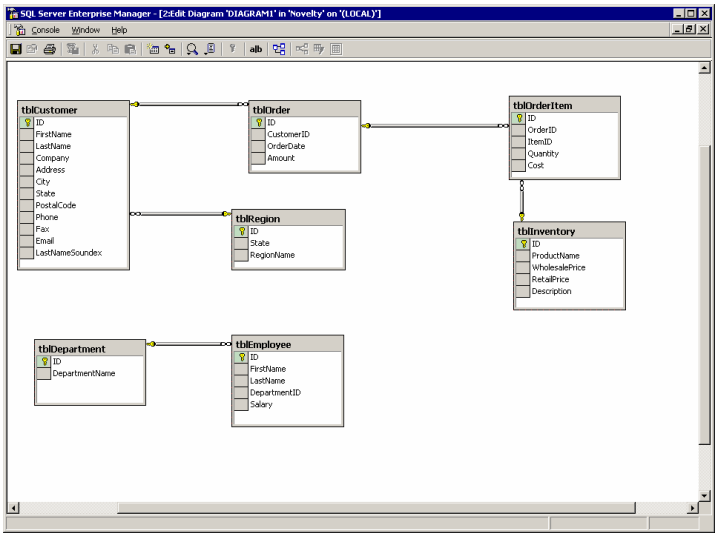
```
DROP tblCustomer
```

תסריט עסקי 1.2:
כתיבת תסריט SQL ליצירת מסד נתונים

כדי להפעיל את קוד הדוגמה שבספר זה, עליך להפעיל תחילה את התסריט NoveltyDB.sql אשר יוצר ומגדיר את מסד הנתונים Novelty במחשב שלך (האם עיינת בתסריט זה? האם אתה תוהה כיצד נכתב?).

מבנה מסד הנתונים עשוי להשתנות בצורה משמעותית משלב הפיתוח, ולכן על המפתח ליצור מדי פעם תסריט אשר מתעד את מסד הנתונים וגם מאפשר ליצור (או ליצור מחדש) את מסד הנתונים באופן אוטומטי. המפתח יכול לכתוב את התסריט באופן ידני, אך מומלץ ליצור אותו באמצעות SQL Server Enterprise Manager, לפחות כבסיס לתסריט שיעבור שינויים בהמשך. גישה זו מהירה בהרבה וחשופה פחות לשגיאות.

מפתחת מסדי הנתונים בחברה Jones Novelties, Incorporated החליטה לנקוט בגישה זו ליצירת מסד הנתונים של החברה. התסריט שנוצר מאפשר התקנה פשוטה של אובייקטים של מסד הנתונים במחשב שבו מסד הנתונים אינו מוגדר. במילים אחרות, התסריט מבצע את כל הפעולות הדרושות כדי ליצור ולהתקין את מסד הנתונים ואת האובייקטים שלו במחשב "נקי", פעולה שדרושה לרוב בהתקנת המערכת באתר חדש. סכמת מסד הנתונים אשר מרכיבה את מסד הנתונים Novelty מוצגת בתרשים 1.28.



תרשים 1.28 סכמת מסד הנתונים Novelty של חברת Jones Novelties, Incorporated.

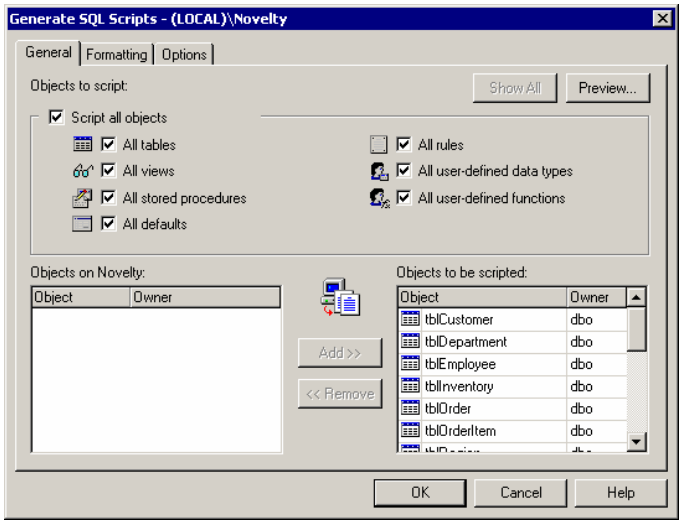
התסריט אשר יוצר את מסד הנתונים יופעל ב- SQL Query Analyzer. כדי לצמצם את מספר הודעות השגיאה, התסריט מוחק אובייקטי מסדי נתונים ישנים בכל הפעלה. המשמעות היא, שבכל פעם שמבוצע שינוי בעיצוב מסד הנתונים במצב עיצוב (design mode), ניתן ליצור שנית את מסד הנתונים מתחילתו על ידי הפעלה חוזרת

של התסריט. גישה זו מבטיחה שכל השינויים המבוצעים בעיצוב מסד הנתונים יחולו על מסד הנתונים בכל פעם שהמפתח יוצר אותו מחדש.

חיסרון השיטה הוא כמובן בעובדה שבמקרה של חוסר תשומת לב מצד המפתח, האצווה עלולה למחוק את כל הטבלאות במסד הנתונים, וגם את הנתונים. לכן, המפתח צריך לשקול חסימה או מחיקה של תסריטים מעין אלה במערכת לאחר הכנסת שרת מסד הנתונים לפעולה מסחרית. כמו תמיד, חשוב לגבות את מסד הנתונים לפני ביצוע שינויים, למקרה שמשהו ישתבש.

כעת, הבה נניח שאתה מפתח מסדי הנתונים של Jones Novelties, Incorporated. בצע את הפעולות שלהלן כדי ליצור את התסריטים.

1. פתח את **SQL Server Enterprise Manager** ולחץ לחיצה ימנית על מסד הנתונים **Novelty**.
2. מהתפריט הנפתח, בחר באפשרות **All Tasks** ולאחר מכן בחר **Generate SQL Scripts** מהתפריט שמוצג. כעת תקבל את תיבת הדו-שיח SQL Scripts.
3. לחץ **Show All** כדי להציג את כל האובייקטים הזמינים עבור מסד הנתונים שנבחר.
4. כעת תוכל לבחור אובייקט אחד או יותר אשר יוגדר בתסריט. סמן את התיבה **Script all objects** כדי לבחור בכל אובייקטי מסד הנתונים, כפי שמוצג בתרשים 1.29.



תרשים 1.29 רשימת כל האובייקטים בתיבת הדו-שיח Generate SQL Scripts.

5. בחר בכרטיסיה **Formatting**. בנוסף להגדרות ברירת המחדל, סמן גם את התיבה **Include descriptive headers**.

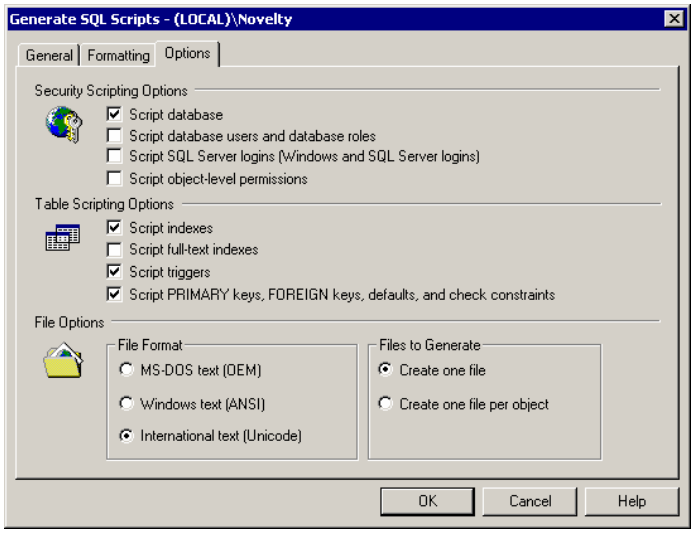
טיפ

אל תשכח לסמן את התיבה Include descriptive headers עבור Script Files שבכרטיסיה Formatting. כך תוסיף באופן אוטומטי שורת כותרת המכילה את התאריך והשעה בה נוצר התסריט. דבר זה שימושי למדי כאשר תרצה לקבוע איזה תסריט הוא המעודכן ביותר מתוך מספר גרסאות.

6. אם רצית רק ליצור תסריט עבור אובייקטי מסד הנתונים, עצור בנקודה זו. עם זאת, כדי ליצור את התסריט ליצירת מסד הנתונים בפועל, המשך ובחר בכרטיסיה Options.

7. ב- Security Scripting Options סמן את התיבה **Script database**. ניתן לבחור ביצירת תסריטים עבור משתמשים, תפקידים והתחברויות של מסד הנתונים, אך כרגע נניח שקיימת משימת ניהול נפרדת לטיפול בכך.

8. ב- Table Scripting Options סמן את התיבות **Script indexes**, **Script triggers** ו- **Script PRIMARY keys, FOREIGN keys, defaults, and check constriants**. ראה תרשים 1.30.



תרשים 1.30 הכרטיסיה Options בתיבת הדו-שיח Generate SQL Scripts.

9. לחץ **OK** כדי להתחיל בתהליך. פעולה זו תציג את תיבת הדו-שיח הסטנדרטית Save As, אשר תמתין להזנת שם ונתיב עבור הקובץ (עם סיומת SQL) שבו יישמר התסריט. הזן נתיב ושם קובץ לשמירת התסריט ולחץ **Save**. בסיום משימת יצירת התסריטים, לחץ **OK** כדי לסגור את תיבת הדו-שיח.

הערה

ייתכן שהתסריט שנוצר במחשב שברשותך לא יהיה זהה לתסריט המוצג כאן, מכיון שהדבר תלוי בסוג המחשב, השרת או בהגדרות מסד הנתונים הספציפיות במחשב. עם זאת, לא תתקשה להבין את הרעיון הכללי.

התסריט המוצג מכיל הצהרות GO אשר הוכנסו כדי לוודא כי הפקודות שלפניהן מבוצעות לפני שהתסריט ממשיך לפעול. פעמים רבות תראה גם בלוקים של פקודות כגון אלו:

```
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

וגם

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO
```

לפני ההפעלה של פקודות מסוימות. בלוקים אלה מבטיחים שמסד הנתונים מוגדר כשורה (באופן זמני) לפעולה הנדרשת. בסיום הפעולה, מבוצע קוד לאיפוס מצב מסד הנתונים.

בסעיף זה אנו מציגים את התסריט בחלקים, אך תסריט SQL מיועד להפעלה כשגרה גדולה אחת. בהתאם לכך, כאשר אתה יוצר או כותב תסריטים בעולם האמיתי, הכנס את התסריט כולו לחלון Query של SQL Query Analyzer (על ידי טעינת התסריט מקובץ טקסט או על ידי חיתוך והדבקה) ולחץ **Execute** כדי להפעיל אותו. לחילופין, תוכל להדגיש חלקים מסוימים או ליצור מספר תסריטים ולהפעילם בנפרד. כאשר אתה מפתח תסריטי SQL באמצעות SQL Query Analyzer, תוכל לערוך את הפקודות בחלון Query, לבדוק אותן על ידי הפעלתן ובדיקת התוצאות ולאחר מכן לשמור אותן לקובץ.

הדבר הראשון שעליך לעשות הוא ליצור מסד נתונים פיסי. קוד 1.1 מציג תסריט שעושה פעולה זו.

קוד 1.1 תסריט ליצירת מסד הנתונים Novelty

```
/****** Object:  Database Novelty    Script Date: ............ ******/
IF EXISTS (SELECT name FROM master.dbo.sysdatabases WHERE name = N'Novelty')
    DROP DATABASE [Novelty]
GO

CREATE DATABASE [Novelty]  ON (NAME = N'novelty_Data',
    FILENAME = N'c:\program files\microsoft sql
server\mssql\data\Novelty_Data.mdf', SIZE = 3, FILEGROWTH = 10%)
    LOG ON (NAME = N'novelty_Log',
FILENAME = N'c:\program files\microsoft sql server\mssql\data\Novelty_Log.LDF',
    SIZE = 3, FILEGROWTH = 10%)
COLLATE Latin1_General_CI_AI
GO
```

לפני הניסיון ליצור תסריט חדש, התסריט שנוצר יבדוק תמיד אם האובייקט קיים ולאחר מכן ישמיט (ימחק) אותו אם הוא קיים. לאחר בדיקת/השמטת מסד הנתונים Novelty, התסריט ייצור מסד נתונים חדש.

הערה

אם אתה כותב או מתאים את תסריט מסד הנתונים, תוכל לשנות את הבדיקה EXISTS כדי למנוע מהתסריט למחוק טבלה אשר מכילה נתונים. הדבר נעשה בדרך כלל במסד נתונים מסחרי, מכיון שאינך רוצה לאפשר מחיקה בטעות של טבלה עם נתונים. עם זאת, במסדי נתונים שנמצאים בשלבי הפיתוח, ניתן להשמיט מסדי נתונים ללא התנייה, אם הם קיימים.

שורת הקוד בה מצוינים הנתונים הפיסיים וקבצי היומן, יחד עם הגודל המקורי שלהם, ערכי הצמיחה ונתיבי הקבצים בכונן הקשיח, היא השורה שסביר להניח כי תרצה לשנות לפני הפעלת התסריט בהתקנה חדשה.

קוד 1.2 מגדיר אפשרויות שונות למסד הנתונים. תוכל לקרוא אודות המשמעות של כל אפשרות על ידי חיפוש הערך המתאים בתיעוד SQL Server (Books Online).

קוד 1.2 תסריט לקביעת אפשרויות מסד הנתונים Novelty

```
exec sp_dboption N'Novelty', N'autoclose', N'false'
GO

exec sp_dboption N'Novelty', N'bulkcopy', N'false'
GO
```

```
exec sp_dboption N'Novelty', N'trunc. log', N'false'
GO

exec sp_dboption N'Novelty', N'torn page detection', N'true'
GO

exec sp_dboption N'Novelty', N'read only', N'false'
GO

exec sp_dboption N'Novelty', N'dbo use', N'false'
GO

exec sp_dboption N'Novelty', N'single', N'false'
GO

exec sp_dboption N'Novelty', N'autoshrink', N'false'
GO

exec sp_dboption N'Novelty', N'ANSI null default', N'false'
GO

exec sp_dboption N'Novelty', N'recursive triggers', N'false'
GO

exec sp_dboption N'Novelty', N'ANSI nulls', N'false'
GO

exec sp_dboption N'Novelty', N'concat null yields null', N'false'
GO

exec sp_dboption N'Novelty', N'cursor close on commit', N'false'
GO

exec sp_dboption N'Novelty', N'default to local cursor', N'false'
GO

exec sp_dboption N'Novelty', N'quoted identifier', N'false'
GO

exec sp_dboption N'Novelty', N'ANSI warnings', N'false'
GO

exec sp_dboption N'Novelty', N'auto create statistics', N'true'
GO
```

```
exec sp_dboption N'Novelty', N'auto update statistics', N'true'
GO
```

כעת, לאחר יצירת מסד הנתונים, תוכל להמשיך ולהשתמש בו. כלומר, להפעיל עליו פקודות. אם תרצה להפעיל את הפקודות הנותרות ליצירת הסכמה על מסד נתונים אחר (לדוגמה, כדי לבצע בדיקות באותו השרת), תוכל לציין בפקודה זו שם מסד נתונים שונה:

```
USE [NoveltyTest]
GO
```

לפני שהתסריט ממשיך לפקודות יצירת האובייקטים, הוא בודק ומוחק אובייקטים באותו שם, אם הם קיימים. אובייקטים אלה כוללים אילוצים (constraints), מזניקים, שגרות מאוחסנות, תצוגות וטבלאות. סדר זה חשוב, מכיון שלא ניתן להשמיט טבלה אם קיימים אובייקטים המשויכים אליה. הקוד לביצוע משימה זו מוצג בקוד 1.3.

קוד 1.3 תסריט למחיקת אובייקטים קיימים במסד הנתונים Novelty

```
if exists (select * from dbo.sysobjects where id =
    object_id(N'[dbo].[FK_tblOrder_tblCustomer]')
    and OBJECTPROPERTY(id, N'IsForeignKey') = 1)
ALTER TABLE [dbo].[tblOrder] DROP CONSTRAINT FK_tblOrder_tblCustomer
GO

if exists (select * from dbo.sysobjects where id =
    object_id(N'[dbo].[FK_tblEmployee_tblDepartment]') and OBJECTPROPERTY(id,
    N'IsForeignKey') = 1)
ALTER TABLE [dbo].[tblEmployee] DROP CONSTRAINT
    FK_tblEmployee_tblDepartment
GO

if exists (select * from dbo.sysobjects where id =
    object_id(N'[dbo].[FK_tblOrderItem_tblInventory]') and OBJECTPROPERTY(id,
    N'IsForeignKey') = 1)
ALTER TABLE [dbo].[tblOrderItem] DROP CONSTRAINT
    FK_tblOrderItem_tblInventory
GO
::
::
```

המשך הקוד נמצא באתר הספר באינטרנט

התסריט ממשיך ויוצר אובייקטים חדשים עבור מסד הנתונים. תחילה נוצרות הטבלאות, כפי שמוצג בקוד 1.4.

קוד 1.4 תסריט ליצירת טבלאות במסד הנתונים Novelty

```
/****** Object:  Table [dbo].[tblCustomer]    Script Date: ........... ******/
CREATE TABLE [dbo].[tblCustomer] (
    [ID] [int] IDENTITY (1, 1) NOT NULL ,
    [FirstName] [varchar] (20) COLLATE Latin1_General_CI_AI NULL ,
    [LastName] [varchar] (30) COLLATE Latin1_General_CI_AI NULL ,
    [Company] [varchar] (50) COLLATE Latin1_General_CI_AI NULL ,
    [Address] [varchar] (50) COLLATE Latin1_General_CI_AI NULL ,
    [City] [varchar] (30) COLLATE Latin1_General_CI_AI NULL ,
    [State] [char] (2) COLLATE Latin1_General_CI_AI NULL ,
    [PostalCode] [varchar] (9) COLLATE Latin1_General_CI_AI NULL ,
    [Phone] [varchar] (15) COLLATE Latin1_General_CI_AI NULL ,
    [Fax] [varchar] (15) COLLATE Latin1_General_CI_AI NULL ,
    [Email] [varchar] (100) COLLATE Latin1_General_CI_AI NULL ,
    [LastNameSoundex] [varchar] (4) COLLATE Latin1_General_CI_AI NULL
) ON [PRIMARY]
GO

/****** Object:  Table [dbo].[tblDepartment]    Script Date: ........... ******/
CREATE TABLE [dbo].[tblDepartment] (
    [ID] [int] IDENTITY (1, 1) NOT NULL ,
    [DepartmentName] [varchar] (75) COLLATE Latin1_General_CI_AI NOT NULL
) ON [PRIMARY]
GO
::
::
```

המשך הקוד נמצא באתר הספר באינטרנט

לאחר מכן נוצרים האילוצים, כפי שמוצג בקוד 1.5.

קוד 1.5 תסריט ליצירת אילוצים (constraints) במסד הנתונים Novelty

```
ALTER TABLE [dbo].[tblCustomer] WITH NOCHECK ADD
    CONSTRAINT [PK_tblCustomer] PRIMARY KEY  CLUSTERED
    (
        [ID]
    )  ON [PRIMARY]
GO

ALTER TABLE [dbo].[tblDepartment] WITH NOCHECK ADD
    CONSTRAINT [tblDepartment_IDPK] PRIMARY KEY  CLUSTERED
```

```
    (
        [ID]
    )  ON [PRIMARY]
GO
::
::
```

המשך הקוד נמצא באתר הספר באינטרנט

לבסוף נוצרות תצוגות, שגרות מאוחסנות ומזניקים, כפי שמוצג בקוד 1.6.

קוד 1.6 תסריט ליצירת תצוגות, שגרות מאוחסנות ומזניקים במסד הנתונים Novelty

```
/****** Object:  View dbo.EmployeeDepartment_view    Script Date: …… ******/

CREATE view EmployeeDepartment_view
as
  select e.ID, FirstName, LastName, DepartmentName
  from tblEmployee e, tblDepartment t
  where e.DepartmentID  = t.ID
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO
::
::
```

המשך הקוד נמצא באתר הספר באינטרנט

התסריט נוצר באופן אוטומטי על ידי Enterprise Manager וניתן להפעילו כמו שהוא, אך תוכל לשנות אותו לפי צרכיך. זכור כי אם אתה יוצר את התסריט מחדש, כל השינויים שבוצעו באופן ידני יאבדו.

אחד השינויים השימושיים הוא הכללת הפקודה Print במקומות אסטרטגיים בתסריט להצגת טקסט בחלון Messages של Query Analyzer. בדרך זו תקבל חיווי חזותי על התקדמות התסריט. תוכל גם להשתמש בפלט המוצג ככלי לניפוי שגיאות שיסייע לגלות שגיאות באצווה. אינך חייב להשתמש בפקודה Print, ולפקודה זו אין השפעה כלשהי על תהליך יצירת מסדי הנתונים.

זכור, כאשר אתה משתמש באצווה מעין זו, תוכל לעשות בה שימוש חוזר כרצונך. אצווה זו נכתבה בצורה שתהרוס לגמרי את מסד הנתונים הקיים באותו שם ותיצור אותו מחדש. אם, במהלך הבדיקות, תטען נתוני דוגמה למסד הנתונים, תוכל לדעת לבטח שנתונים אלה לא יישארו בו כאשר יעבור לפעילות מסחרית. בנוסף לכך, יצירת מסד נתונים מאצווה מאפשרת להעביר את עיצוב מסד הנתונים לשרתים נוספים ולהשתמש בשני שרתי מסד נתונים שונים – אחד לפיתוח ואחד לעבודת אמת.

סיכום

בפרק זה הצגנו את היסודות להתחלת העבודה עם יישומים מבוזרים ב- Microsoft SQL Server. זכור כי למרות שהתמקדנו בפרק זה באופן ההגדרה והשימוש ב- Microsoft SQL Server 2000, החומר המופיע בפרקים הבאים מתאים לכל מערכת מסד נתונים – Oracle, Sybase, Informix או כל מערכת אחרת. כל עוד קיים מנהל התקן ODBC או ספק OLE DB שיכול לגשת לנתוני הקצה שלך, תוכל להשתמש במסד הנתונים מתוך Visual Basic.NET.

שאלות ותשובות

ש: תמיד פחדתי להתעסק עם SQL Server. השרת נראה לי תמיד כדבר אפל ומסתורי. האם הנושאים המתוארים בפרק זה יאפשרו לי ליצור יישומי מסד נתונים רציניים בעזרת SQL Server מבלי להסתבך בקשיים?

ת: כן ולא. פרק זה אינו מיועד להסביר משימות שוטפות של ניהול מסד נתונים, שיפור ביצועים או נושאים דומים, ואף אינו מהווה מדריך מקיף ל- SQL Server. הוא משמש כמבוא בלבד. החומר המתואר בחצי הראשון של פרק זה יאפשר לך להבין את פעולת השרת SQL. המעבר ממחשוב בסביבת משתמש יחיד (single user) וקבוצות עבודה קטנות (workgroup) למערכת שרת-לקוח (client-server) אינו פשטני וגם אינו פשוט, אך אין לראות בכך קופסה שחורה. פרק זה מיועד להפיג את הפחד מהמעבר.

ש: אם רוב השאילתות שאני מבצע הן ישירות ואינן כוללות לוגיקה מסובכת, האם יש לי סיבה להשתמש בשגרות מאוחסנות?

ת: כן. למעשה, יש שני יתרונות עיקריים לשימוש בשגרות מאוחסנות במקום לקודד שאילתות SQL בקוד היישום שלך:

1. **ביצועים**. עבור מתכנתים רבים, יתרון זה מספיק כדי להשתמש בשגרות מאוחסנות! השיפור בביצועים שנובע מהשימוש בשאילתה המיושמת באמצעות שגרה מאוחסנת ולא בקוד היישום נובע מהעובדה שהשגרה המאוחסנת כבר עברה הידור (compiling) ותכנון על ידי SQL Server עוד לפני שבוצעה קריאה אליה. כאשר מועברת מחרוזת שאילתת SQL מהלקוח אל SQL Server, עליה לעבור תחילה תהליך פירוש (parsing) והידור וגם לקבל נתיב הפעלה, עוד לפני שניתן

להפעילה. פעולות אלו יוצרות תקורה רבה שמתבטאת בהאטת הפעולה בזמן אמת.

2. **יכולת ניהול**. יישום שאילתות כשגרות מאוחסנות גורם לכך שכל שאילתות היישומים יאוחסנו במקום מרכזי אחד, ולא בתפזורת של עשרות אלפי שורות בקוד היישום. כמו כן, שיטה זו מאפשרת למספר פרויקטים או יישומים לנצל קוד אחד כאשר יש מספר משתמשים במסד נתונים משותף. המשמעות היא פחות עבודה (קידוד/ניפוי שגיאות/בדיקה) ופחות שגיאות. שיטה זו גם מאפשרת לנצל את מנגנוני בקרת האבטחה המתקדמים של SQL Server. לבסוף, השימוש בשגרות מאוחסנות מאפשר למפתחי קוד היישום, אשר מתמחים בלוגיקה עסקית וזרימת יישום, להתמקד בתחום שלהם ולהשאיר את נושאי הגישה וביצוע השאילתות במסד הנתונים למומחי מסד הנתונים בצד השרת.

פרק 2

ADO.NET – Data Providers

מה בפרק זה?

- סקירת ADO.NET
- סקירת האובייקטים של .NET Data Provider.
- האובייקט Connection
- האובייקט Command
- האובייקט DataReader
- שימוש ברכיבים Connection ו-Command
- אובייקטים נוספים של ספק נתונים

לימוד ADO.NET בספר זה מבוסס על כך שיש לך ידיעה כלשהי בנושא וגם התנסות כלשהי בעבודה בטכנולוגיה זו. אם אין לך ניסיון בעבודה עם ADO.NET, תוכל להיעזר בספר "**Visual Basic.Net סדנת לימוד**", **הוצאת הוד-עמי**, פרקים 11 ו-12; ובספר "**ASP.NET סדנת לימוד**", **הוצאת הוד-עמי**, פרקים 8, 9 ו-10.

לעיתים נראה כי מיקרוסופט מפתחת מדי פעם שוב ושוב מודל נתונים חדש ושונה עבור מפתחי מסד נתונים. בפרק זה נתמקד בחדשות האחרונות (בעת כתיבת הספר) – ADO.NET. נפתח בהסבר של הסיבות שגרמו לדעתנו לפיתוח מודל חדש זה לגישה לנתונים, וגם נברר אם אכן היתה סיבה לפיתוחו. לאחר מכן נציג סקירה כללית של המודל ושל המבנה הפנימי שלו.

מטרתנו היא להכין את התשתית לעבודה עם ADO.NET באמצעות דיון בפעולות בסיסיות ומבט מעמיק על מספר אובייקטים בסיסיים של ספק הנתונים ADO.NET (ADO.NET Data Provider) – האובייקטים Connection, Command, Parameter ו-DataReader. בפרקים 3, 4 ו-5 נציג בחינה מקיפה של אובייקטים מתקדמים ומעניינים יותר של ADO.NET, אשר מבוססים כולם על DataSet – האובייקט המרכזי של ADO.NET.

סקירת ADO.NET

אם כבר עסקת בפיתוח יישומי מסד נתונים ב- Visual Basic, כבר התרגלת לכך שמיקרוסופט מוציאה מודל גישה חדש ומשופר לנתונים מדי כמה שנים. בנוסף ל-TLA (Three-Letter Acronym) חדש, קיים גם מודל אובייקטים או API חדש שיש לשלוט בו. בשנים האחרונות, מפתחים עבדו עם ODBC, DAO, RDO ו-ADO לפני שהתחילו לעבוד עם ADO.NET החדש. בכל פעם שיוצאת טכנולוגיה חדשה, עליך ללמוד את יעדיה ואת המבנה שלה, ואז לשאול את עצמך: האם יש טעם בכך שאני והצוות שלי נתחיל להשתמש בטכנולוגיה חדשה זו? ברוב המקרים, התשובה היא **כן**, אלא אם אתה עובד על פרויקט שהדרישות הנוכחיות והעתידיות שלו אינן מושפעות מתכונות ויכולות חדשות שקיימות בשוק. ברוב המקרים, אין זה המקרה, למרות שהזמינות של RDO (Remote Data Objects) אינה רלוונטית בפרויקטים שבוצעו באמצעות מנוע מסד הנתונים JET (MDB). שים לב ש-DAO עדיין מומלץ לגישה באמצעות MDB.

מוטיבציה ופילוסופיה

אם כן, הבה ונשאל את שאלת מיליון הדולר: מדוע אנו זקוקים למודל אובייקטים חדש עבור גישה לנתונים? התשובה הפשוטה היא: "ביקשתם – קיבלתם!". כלומר, ADO.NET משלב תכונות רבות אשר היו דרושות למפתחים מאז יציאת ADO לשוק לפני מספר שנים. עם השנים, נוספו ל-ADO מספר תכונות, כגון ערכות רשומות מנותקות (disconnected record sets) ותמיכה ב-XML. הבעיה היתה שאלו היו תוספות בלבד ולא הוטמעו בתכנון הראשוני, ולכן גם השימוש בהן היה מסורבל ולא ענה על כל הצרכים.

ל-ADO הקלאסי (מבוסס COM) יש אובייקט יחיד, Recordset, שמיועד לשימושים רבים. Recordset יתנהג בצורה שונה וישמש למטרות שונות בהתאם למאפייני הגדרה שונים, כגון סוג סמן, מיקום סמן וסוג נעילה.

ב-ADO.NET, הפונקציונליות והמטרות השונות מחולקות לאובייקטים נפרדים, אשר ניתן להשתמש בהם בנפרד או בשילוב. יכולת זו מאפשרת למפתחים להשתמש באובייקטים המיועדים במיוחד למטרות מסוימות, מבלי לשאת בעול של אובייקטים מיותרים. האובייקטים השונים מיועדים לעבודה הדדית זה עם זה כדי לספק פונקציונליות מתקדמת בצורה קלה יותר.

תמיכה ביישומים מבוזרים ובמודל תכנות מנותק

ADO.NET מספק תמיכה גמישה וטובה בפיתוח יישומים המבוזרים על פני מספר מחשבים (שרתי מסדי נתונים, שרתי יישומים ותחנות עבודה). תמיכה מיוחדת ניתנת ליישומים מנותקים (disconnected applications. כלומר, n/three-tier applications). שיטה זו מקטינה מאוד את העומס והנעילה של משאבים בשרת מסד הנתונים. התוצאה היא מדרוגיות (Scalability) טובה יותר – יכולת לתמוך במספר גדול יותר של משתמשים בו-זמנית על ידי הוספה הדרגתית של חומרה חדשה. יתרון זה קריטי במיוחד בפיתוח יישומי אינטרנט.

תמיכה מקיפה ב-XML

ADO קלאסי מסוגל לשמור ולקרוא נתונים בפורמט XML, אך הפורמט אינו טבעי למערכת וקשה לעבוד איתו. בנוסף לכך, תמיכת XML נוספה ל-ADO בשלב מאוחר יחסית לאחר פיתוחו ולכן רמת התמיכה מוגבלת ואינה גמישה. בניגוד לכך, תמיכת XML ב-ADO.NET הוטמעה במערכת מתחילתה. תפיסת ADO.NET אומרת כי "נתונים הם נתונים", ואין חשיבות למקור שלהם. המערכת צריכה לגשת ולעבד אותם כנתונים יחסיים או היררכיים על פי הצורך, כתלות בכלי הקיים או המבוקש.

בנוסף לכך, מבנה תעבורת הנתונים בין שכבות ו/או מחשבים משתמש ב-XML. עובדה זו אינה פותרת לגמרי את הבעייתיות שבצורך לאפשר קריאות COM דרך קירות אש (firewalls), אך היא מאפשרת שיתוף נוסף של נתונים עם יישומים הפועלים על פלטפורמות שאינן Windows (מכיון שכל מערכת מסוגלת לעבד XML מבוסס טקסט).

שילוב עם NET Framework.

ADO.NET אינה רק הגירסה הבאה של ADO. גירסה זו תוכננה ויושמה במיוחד כחלק מתפישת NET, או NET Framework.. כלומר, היא פועלת כקוד מנוהל (Managed Code) וכל האובייקטים שבה מתנהגים כפי שמצופה מאובייקט NET.. זהו חלק מחבילת NET Framework. הסטנדרטית, ולכן גם נפתרות בעיות השוני בין גרסאות בהן נתקלו מפתחי ADO בעבר.

"מראה ותחושה" של ADO

ADO.NET תוכננה ויושמה במיוחד כחלק של NET Framework., אך ברובה תיראה עדיין מוכרת למפתחים בעלי ניסיון ב-ADO הקלאסי. גם בתכונות חדשות או כאלו שמיושמות בצורה שונה ב-ADO.NET, מפתח ADO מנוסה יוכל לנצל את הידע שלו ב-ADO ולהבין במהירות כיצד לנצל את אובייקטי ADO.NET.

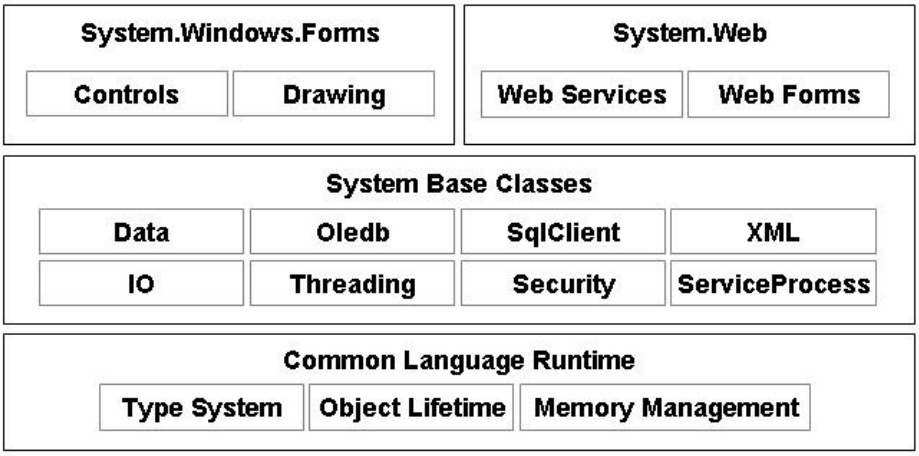
השוואת ADO.NET ל-ADO קלאסי (ADO 2.X)

כאשר אתה בודק את ההבדלים בין ADO.NET לבין ADO הקלאסי, קח בחשבון את הגורמים שלהלן:

- ADO הקלאסי מיועד לגישה מחוברת (connected access) ומוגבל למודל הנתונים הפיסי. בניגוד לכך, ADO.NET מיועד לגישה מנותקת (disconnected access) ומסוגל לעצב נתונים בצורה לוגית.
- ב-ADO.NET יש אבחנה ברורה בין מודל הגישה לנתונים מחוברים לבין מודל התכנות המנותק.
- ב-ADO.NET לא קיימים התכונות CursorType, CursorLocation או LockType. קיימים סמן צד לקוח, סמן סטטי ונעילה אופטימיסטית.
- במקום להשתמש באובייקט יחיד למטרות שונות, ADO.NET מחלק את תפקיד recordset שהיה ב-ADO הקלאסי לאובייקטים ייעודיים קטנים יותר, כגון DataReader, DataSet ו-DataTable.
- ADO.NET מאפשר טיפול בנתוני XML ולא להשתמש ב-XML כפורמט קלט/פלט בלבד.
- ADO.NET מציע סוגי DataSets חזקים ולא רק שדות מסוג Variant. תכונה זו מאפשרת גילוי שגיאות טובה יותר בשלב התכנון, וביצועים משופרים בהפעלה.

אובייקטי ADO.NET ב- NET Framework.

תרשים 2.1 מדגים כיצד מחלקות ADO.NET מתאימות ל- NET Framework. הכללית. בתחתית התרשים נמצא CLR (Common Language Runtime), אשר מהווה תשתית הפעלה לכל יישומי NET. (ללא תלות בשפה בה הם כתובים). CLR מספק מערכת סוגי נתונים (CTS – Common Type System), ניהול זיכרון וניהול משך חיי אובייקטים.



תרשים 2.1 מחלקות ADO.NET ב- NET Framework.

השכבה הלוגית הבאה, אשר בנויה על תמיכת CLR, היא ערכת מחלקות מערכת בסיסיות (System Base Classes). אלו הן מחלקות המספקות פונקציונליות עשירה שמנוצלת לשירות יישומי NET.. תרשים 2.1 מציג רק חלק מהמחלקות בספריית NET Framework.. למעשה, זהו ה- Windows API (Application Programming Interface) החדש. בעבר, כדי לגשת לפונקציונליות של מערכת ההפעלה Windows היה צורך לעבור דרך Windows API, אשר הורכב מסדרה גדולה של קריאות פונקציה שלא תוכננו בצורה עקבית. ב-NET., הגישה לפונקציונליות זו (וכן תכונות חדשות) היא על ידי שימוש בתכונות ושיטות של מחלקות בסיסיות של המערכת. גישה זו מותאמת יותר לסביבה מונחת אובייקטים, היא עקבית יותר ונוח לפתח באמצעותה יישומי Windows מכל סוג שהוא, וכמובן שניתן לפתח שירותי רשת (Web Services).

שכבה זו מכילה מספר מרחבי שמות (namespaces, המייצגים קבוצות של מחלקות והגדרות נוספות) הקשורים לגישה לנתונים: System.Data, System.Data.OleDb ו- System.Data.SqlClient. בפרק זה **ובפרקים 3, 4 ו-5** נבחן מקרוב מחלקות והגדרות רבות במרחבי שמות אלה.

ממשקי יישומים

הרמה העליונה היא המקום שבו הפיצול, או ההבדל, קיים בין סוגים שונים של יישומים אותם המפתח יכול ליצור. יש מחלקות ופקדים ליצירת יישומי Windows קלאסיים מבוססי טפסים (Windows Forms), מחלקות ופקדים נוספים ליצירת יישומי אינטרנט מבוססי דפדפן (Web Forms) ומחלקות ליצירת יישומים של Web Services. עם זאת, כל היישומים משתמשים במחלקות משותפות ללוגיקת היישום – **מחלקות מערכת בסיסיות** (system base classes).

כעת, כאשר ברור לך כיצד לשבץ את מחלקות ADO.NET במבנה הכולל של NET Framework., נבחן לעומק את האובייקטים העיקריים של ADO.NET.

סקירת אובייקטים של ספקי נתונים ב-NET.

למרות הדגש על מודל התכנות המנותק (disconnected model of programming), עדיין צריך להתחבר למסד הנתונים הפיסי כדי לאחזר, לעדכן, להוסיף ו/או למחוק ממנו נתונים. התוכנה אשר אחראית על הקשר עם מסד הנתונים הפיסי ב-ADO.NET נקראת NET Data Provider.. ספק נתונים (data provider) מורכב מקוד מנוהל של NET. המקביל לספק OLE DB או מנהל ODBC, וממספר אובייקטים אשר מבצעים את הפעולות הדרושות, כפי שמוגדר על ידי המחלקות והממשקים מהם הם נגזרים.

כרגע קיימים שלושה ספקי נתוני ADO.NET שונים, שכל אחד מוגדר במרחב שמות (namespace) נפרד. קידומות האובייקטים במרחבי שמות אלה הן OleDb, Sql ו-Odbc, בהתאמה. כאשר נתייחס לאובייקטים אלה במובן הגנרי, נשתמש בשם האובייקט ללא הקידומת.

SqlClient

ספק הנתונים SqlClient תוכנן לעבודה במיוחד עבור SQL Server 7.0 או גירסאות מתקדמות יותר. לספק נתונים זה יש ביצועים משופרים בהרבה, מכיון ש:

1. הוא יוצר קשר ישיר עם מסד הנתונים באמצעות פרוטוקול TDS (Tabular Data Stream) הטבעי שלו, ולא באמצעות OLEDB אשר ממפה את ממשק OLEDB לפרוטוקול TDS.
2. אין תקורה של שירותי הפעולה ההדדית של COM (COM Interop Services).
3. אין עומס או פונקציונליות מיותרים למרכיבים שאינם נתמכים על ידי SQL Server.

OLEDB

ספק הנתונים Oledb מנצל את OLEDB Provider (COM) הטבעי הקיים ואת שירותי הפעולה ההדדית NET COM. עבור גישה אל מסד הנתונים. השתמש בספק נתונים זה אם אינך ניגש למסד נתונים של SQL Server 7.0 ומעלה. ספק נתונים זה מאפשר לגשת לכל מסד נתונים שקיים בו ספק OLEDB. האובייקטים של ספק זה נמצאים במרחב השמות System.Data.OleDb.

ODBC

השתמש בספק הנתונים ODBC אם אתה ניגש למסדי נתונים שאין בהם NET Data Povider., או שאין להם ספק OLEDB (COM) משלהם. כמו כן, במסד נתונים קיים, ייתכן שספק ODBC עשוי לשפר את הביצועים בהשוואה לספק OLEDB, ולכן מומלץ לבדוק כדי להתאים את הספק הטוב ביותר ליישום. האובייקטים של ספק נתונים זה נמצאים במרחב השמות System.Data.Odbc.

הפיתוח של ספק נתונים עבור ODBC השתהה מעט מאחורי חלקים אחרים של NET Framework. או Visual Studio.NET. לכן, הוא לא נכלל במהדורת Visual Studio.NET הראשונה, אך ניתן להוריד אותו מאתר האינטרנט של מיקרוסופט. בדוק מדי פעם והורד ספקי נתונים נוספים של NET. אשר יהיו זמינים.

נכון לעתה, קיים באתר האינטרנט של מיקרוסופט ספק נתוני NET. Oracle. ספקי הנתונים ODBC ו-Oracle ניתנים להורדה.

בכל מקום בספר שמצוין System.Data.Odbc יש לשנות אותו ל- Microsoft.Data.Odbc. (מיקרוסופט שינתה את מרחב השמות).

ככל הניתן, במידה וקוד של פונקציה שונה – הוא קיים בקוד המצורף לספר כאשר הגירסאות הבאות נמצאות בתוך הערה.

אובייקטים עיקריים

כל ספק נתונים מורכב מארבעת האובייקטים העיקריים המתוארים בטבלה 2.1.

טבלה 2.1 אובייקטים של ספקי נתונים עיקריים

אובייקט	תיאור קצר
Connection	יוצר התחברות למקור נתונים מסוים.
Command	מפעיל פקודה במקור נתונים. חושף אוסף אובייקטי Parameter ושיטות להפעלת סוגים שונים של פקודות.
DataReader	קורא ומחזיר נתונים לקריאה בלבד (קדימה בלבד) של נתונים ממקור נתונים.
DataAdapter	מגשר בין DataSet לבין מקור נתונים לצורך אחזור ושמירת נתונים.

כל אובייקט נגזר ממחלקה בסיסית גנרית ומיישם ממשקים גנריים, אך מספק יישום מסוים משלו. לדוגמה, SqlDataAdapter, OleDbDataAdapter ו-OdbcDataAdapter נגזרים כולם מהמחלקה DbDataAdapter ומיישמים את אותם הממשקים. עם זאת, כל אחד מהם מיישם את הממשקים עבור מקור נתונים מסוים.

מרחב השמות System.Data.OleDb מכיל את האובייקטים האלה:

- OleDbConnection
- OleDbCommand
- OleDbDataReader
- OleDbDataAdapter

מרחב השמות System.Data.SqlClient מכיל את האובייקטים האלה:

- SqlConnection
- SqlCommand
- SqlDataReader
- SqlDataAdapter

מרחב השמות System.Data.Odbc מכיל את האובייקטים האלה:

- OdbcConnection
- OdbcCommand
- OdbcDataReader
- OdbcDataAdapter

על פי שיטה זו, לכל ספקי הנתונים הנוספים או לאלה שייווצרו בעתיד יהיו מרחבי שמות נפרדים וגם קידומות נפרדות וגם הם יישמו את האובייקטים הדרושים כראוי.

האובייקט Connection

האובייקט Connection של ADO.NET מזכיר מאוד את האובייקט Connection המוכר מ-ADO הקלאסי. השימוש באובייקט מובן מעצם שמו – יצירת חיבור למקור נתונים מסוים באמצעות חשבון משתמש וסיסמה מסוימים, כפי שמצוין במחרוזת ההתחברות. תוכל להתאים את ההתחברות על ידי ציון פרמטרים וערכים נוספים במחרוזת ההתחברות. לאחר יצירת ההתחברות, אובייקט Command (או DataAdapter) יוכל להשתמש בה כדי לבצע פעולות רצויות במקור הנתונים.

הערה

בשונה מהאובייקט Connection של ADO 2.X, האובייקט Connection של ADO.NET אינו כולל את השיטות Execute או OpenSchema. היכולת להפעיל פקודות SQL זמינה באובייקטים Command ו-DataAdapter בלבד. הפונקציונליות של השיטה OpenSchema זמינה אף היא באמצעות השיטה GetOleDbSchemaTable של האובייקט OleDbConnection.

האובייקטים הנגזרים OleDbConnection, SqlConnection ו-OdbcConnection מיישמים כולם את אותו הממשק, אך קיימים ביניהם הבדלים. לדוגמה, קיים שוני במבנה מחרוזת ההתחברות. מבנה OleDbConnection מתוכנן באופן שיתאים למבנה מחרוזת התחברות OLEDB סטנדרטית עם חריגות מזעריות. מבנה OdbcConnection בנוי באופן שיתאים למחרוזת התחברות ODBC סטנדרטית, אך הוא מכיל הבדלים אחדים. מבנה מחרוזת ההתחברות של SqlConnection שונה משני האחרים, מכיון שהוא מכיל פרמטרים רלוונטים לגירסה SQL Server 7.0 ומעלה בלבד.

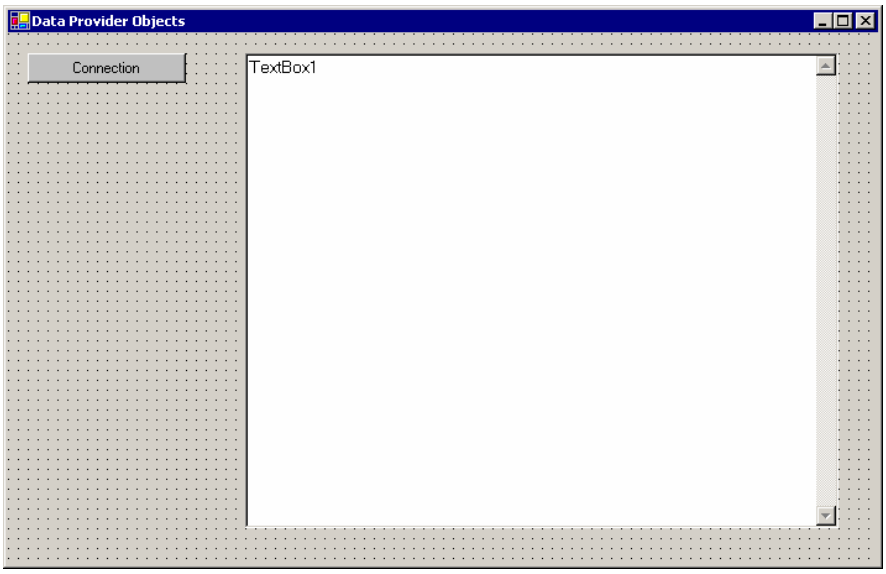
בנוסף לכך, באובייקטים מסוימים יתווספו תכונות. לדוגמה, OleDbConnection מכיל את התכונה Provider אשר מציינת את הספק OLEDB שתפעיל; OdbcConnection מכיל תכונה בשם Driver אשר מציינת את מנהל ODBC שתפעיל; SqlConnection אינו מכיל אף אחת מתכונות אלו מכיון שסוג מקור הנתונים נקבע מראש (SQL Server). עם זאת, SqlConnection מכיל את התכונות PacketSize ו-WorkStationID הקיימים רק ב- SQL Server ואינן נתמכות בשני סוגי ההתחברות האחרים.

אם כך, הבה ונתחיל לכתוב קוד! ננחה אותך להפעיל את כל אחד מהאובייקטים של ספקי הנתונים העיקריים. נעשה זאת בשלבים, ונתחיל בדוגמה הפשוטה הבאה, אשר נפתח במהלך ההסבר.

1. הפעל את **Visual Studio.NET**.
2. צור פרויקט **Visual Basic Windows Application** חדש.
3. קבע לפרויקט את השם **DataProviderObjects**.

4. ציין נתיב לשמירת קבצי הפרויקט.
5. הגדל את Form1.
6. בחלון Properties של Form1, קבע את ערך התכונה Text ל- **Data Provider Objects**.
7. בפינה השמאלית העליונה של הטופס, הוסף לחצן (Button) מהכרטיסיה Windows Forms של תיבת הכלים.
8. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**cmdConnection** ואת ערך התכונה Text ל-**Connection**.
9. מהכרטיסיה Windows Forms של תיבת הכלים, הוסף תיבת טקסט (TextBox) עבור Form1 והצב אותה בצד ימין של הטופס.
10. בחלון Properties, קבע את ערך התכונה Name של תיבת הטקסט ל-**txtResults**, את ערך התכונה Multiline ל-**True** ואת הערך ScrollBars ל-**Both**.
11. הגדל את תיבת הטקסט באופן שתכסה כ-80% משטח הטופס.

בסיום הפעולות, הטופס שיצרת אמור להיראות דומה לזה שמוצג בתרשים 2.2.



תרשים 2.2 הטופס Form1 של פרויקט הדוגמה DataProviderObjects.

עבור לתצוגת הקוד של הטופס והוסף את שורות הקוד הבאות בראש הקובץ. פעולה זו תגרום לייבוא מרחבי השמות שתשתמש בהם לפיתוח יישום הדוגמה בפרק זה:

```
Imports System.Data.SqlClient
Imports System.Data.OleDb
Imports System.Data.Odbc
```

שים לב למרחבי השמות של מחלקות ADO.NET הגנריות ולהגדרות ולמרחבי השמות הנפרדים של כל ספק נתונים.

ייתכן שעורך Visual Studio לא יזהה את מרחב השמות Microsoft.Data.Odbc, מכיון שמדובר בתוספת למוצר הבסיסי. במקרה זה, בצע את הפעולות שלהלן:

1. הורד את קובץ ההתקנה של ספק הנתונים ODBC מאתר האינטרנט של מיקרוסופט ופעל על פי ההנחיות המוצגות בו, כדי להתקין את הספק במחשב.
2. ב- Solution Explorer, לחץ לחיצה ימנית על הרכיב **References** של הפרויקט DataProviderObjects.
3. בחר **Add Reference** מהתפריט הנפתח.
4. בכרטיסיה NET. של תיבת הדו-שיח Add Reference, גלול את רשימת הרכיבים עד שתגיע ל- Microsoft.Data.Odbc.dll.
5. לחץ לחיצה כפולה על פריט הרשימה **Microsoft.Data.Odbc.dll** כדי להוסיף אותו לרשימה Selected Components שבתחתית תיבת הדו-שיח.
6. לחץ **OK** כדי לסגור את תיבת הדו-שיח.

אם מסיבה כלשהי, אחד ממרחבי השמות המיובאים אינו מזוהה, עליך להוסיף הפניה ל-System.Data.dll. בצע שלבים 2 עד 6 והחלף את **System.Data.dll** ב- **Microsoft.Data.Odbc** בשלב 4.

כעת הוסף את קוד 2.1 ל-btnConnection, כדי לפתוח התחברות למסד הנתונים pubs שב- SQL Server. קוד זה פותח התחברות ומציג את מצבה לפני ואחרי הניסיון לפתוח אותה.

קוד 2.1-VB קוד לפתיחת התחברות מסד נתונים והצגת מצב ההתחברות

```
Private Sub btnConnection_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnConnection.Click

    ' Create an instance of an Connection object
    Dim cnn As SqlConnection = New SqlConnection()

    ' Set the connection string
    cnn.ConnectionString = "server=localhost;uid=sa;database=pubs"
    txtResults.Clear()

    ' display connection state
```

```
    If (cnn.State = System.Data.ConnectionState.Open) Then
        txtResults.Text = txtResults.Text & "Connection is Open"
    Else
        txtResults.Text = txtResults.Text & "Connection is Closed"
    End If
    txtResults.Text = txtResults.Text & ControlChars.CrLf

    ' Open the Connection
    txtResults.Text = txtResults.Text & "Opening DB connection..." _
          & ControlChars.CrLf
    cnn.Open()

    ' display connection state
    If (cnn.State = System.Data.ConnectionState.Open) Then
        txtResults.Text = txtResults.Text & "Connection is Open"
    Else
        txtResults.Text = txtResults.Text & "Connection is Closed"
    End If
    txtResults.Text = txtResults.Text & ControlChars.CrLf

  End Sub
```

קוד #C-2.1 קוד לפתיחת התחברות מסד נתונים והצגת מצב ההתחברות

```
private void btnConnection_Click(object sender, System.EventArgs e)
     {
       // Create an instance of an Connection object
       SqlConnection cnn = new  SqlConnection();

       // Set the connection string
       cnn.ConnectionString = "server=localhost;uid=sa;database=pubs";

       txtResults.Clear();

       txtResults.Text = txtResults.Text +
          "Connection is "
          + cnn.State.ToString()
          +"\n\r";

       txtResults.Text = txtResults.Text + "\n\r\n\r";

       // Open the Connection
       txtResults.Text = txtResults.Text + "Opening DB connection..." +
```

```
        "\n\r\n\r";
    cnn.Open();

    // display connection state

    // the above lines can be replaced with the following
    txtResults.Text = txtResults.Text + "Connection is " +
    cnn.State.ToString();

    txtResults.Text = txtResults.Text;
}
```

טיפ

ב-VB.NET קיימת תכונה חדשה ושימושית להשגת ייצוג מחרוזת טקסט של ערך מנייה (enum) באופן אוטומטי, במקום כתיבת שגרה אשר מבצעת פקודת select-case על כל הערכים האפשריים של המנייה. כל סוגי המנייה שהם אובייקטים, יורשים את השיטה ToString אשר מחזירה את המחרוזת המתאימה לערך הנוכחי שלה.

בקוד 2.1 ניתן להחליף את ההצהרות If-Else אשר מציגות את מצב ההתחברות בשורת קוד אחת. כלומר, תוכל להחליף את השורות:

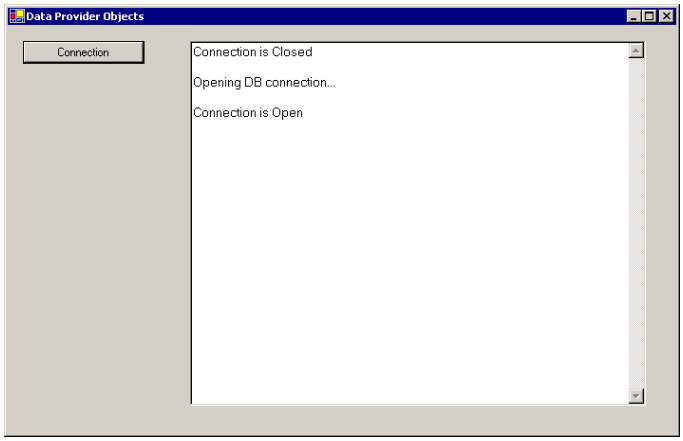
```
' display connection state
IF (cnn.State = System.Data.ConnectionState.Open) Then
    txtResults.Text = txtResults.Text & "Connection is Open"
ELSE
    txtResults.Text = txtResults.Text & "Connection is Closed"
End If
```

בשורת הקוד הזו:

```
VB    ' display connection state
      txtResutls.Text = txtResultsText & "Connection is" & cnn.State.ToString & _
        ControlChars.CrLf
C#    // display connection state
      txtResults.Text = txtResults.Text + "Connection is " +
        cnn.State.ToString() + "\n\r\n\r" ;
```

(הקוד המצורף לספר כבר מכיל שורות אלו. השורות הקודמות נמצאות בתוך הערה)

כאשר תפעיל את הפרויקט DataProviderObjects ותלחץ על Connection, תיבת הטקסט תציין כי ההתחברות סגורה, נפתחת (בשלבי פתיחה) ולאחר מכן פתוחה, כפי שמוצג להלן בתרשים 2.3.



תרשים 2.3 תוצאות לפני ואחרי פתיחת התחברות באמצעות שימוש בקוד 2.1.

הערה

כאשר כותבים קוד מסחרי, חייבים לקבוע את אסטרטגיית הטיפול בהודעות שגיאה ברוב השגרות והפעולות. בדרך כלל, אסטרטגיה זו תתבסס על מבנה הטיפול בשגיאות בלוק Try-Catch. לרוב, אנו לא כוללים קוד זה בדוגמאות המובאות, מכיון שאנו מתמקדים בתפיסות תכנות של מסד נתונים ולא בשיטות תכנות כלליות ב-VB.NET.

על Try-Catch תוכל לקרוא בספר "**Visual Basic.Net סדנת לימוד**", **הוצאת הוד-עמי**, בפרק 6, "תיקון שגיאות וטיפול בחריגים"; ובספר "**C# סדנת לימוד**", **הוצאת הוד-עמי**, בפרק 10, "Handling Exceptions (טיפול בחריגים)".

האובייקט Command

האובייקט Command של ADO.NET מוכר אף הוא למתכנתי ADO 2.X מנוסים. בדומה לאובייקט Connection של ADO.NET, האובייקט Command דומה לקודמו ב- ADO 2.X. אובייקט זה מאפשר להפעיל פקודות על מקור נתונים ולהשיג את התוצאות של הנתונים המוחזרים ו/או את הנתונים עצמם, במידה וקיימים כאלה.

כצפוי, לאובייקט זה יש את התכונות CommandText ו-CommandType אשר מגדירות את הטקסט והסוג בפועל של הפקודה, את התכונה Connection אשר מציינת באיזו התחברות יש להשתמש כדי להפעיל את הפקודה ואת התכונה CommandTimeout אשר מגדירה זמן המתנה לסיום הפקודה לפני יצירת שגיאה. האובייקט מכיל גם את התכונה Parameters, המכילה אוסף פרמטרים אשר מועברים אל הפקודה המבוצעת או ממנה. לבסוף, שלא כמו באובייקט Command של ADO קלאסי, התכונה Transaction מציינת את התנועה (transaction) שבה הפקודה מבוצעת.

לכל שלוש הגרסאות של האובייקט Command (OleDb, Sql Client ו-Odbc) יש את אותן תכונות ושיטות, בהבדל אחד. ב-SqlCommand, שהוא חלק מ- System.Data.SqlClient כלולה השיטה ExecuteXmlReader שאינה קיימת בשני האובייקטים האחרים. שיטה זו מנצלת את יכולתו של SQL Server להחזיר באופן אוטומטי נתונים במבנה XML (אם נוסף המשפט FOR XML לשאילתת SQL Select).

הערה

הבדל נוסף בין גרסאות האובייקט Command בספקי הנתונים השונים קשור בערכי התכונה CommandType. כל שלושת האובייקטים תומכים בסוג Text או StoredProcedure, אך האובייקט OledbCommand תומך גם בערך שלישי בשם TableDirect. שיטה זו טוענת בצורה יעילה תוכן של טבלה שלמה על ידי קביעת ערך CommandType ל-TableDirect ואת ערך CommandText לשם הטבלה.

כעת נמשיך בטופס שהכנת, כפי שמודגם בתרשים 2.1.

1. מהכרטיסיה Windows Forms של תיבת הכלים, הוסף לחצן מתחת ללחצן btnConnection.
2. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**btnCommand** ואת ערך התכונה Text ל-**Command**.
3. הוסף את הקוד הדרוש ללחצן btnCommand, כפי שמוצג בקוד 2.2.

קוד 2.2-VB קוד לפתיחת התחברות מסד נתונים והכנת אובייקט command

```
Private Sub btnCommand_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnCommand.Click

    ' Create an instance of an Connection object
    Dim cnn As SqlConnection = New SqlConnection( _
      "server=localhost;uid=sa;database=pubs")

    ' Create instances of Command and paramter objects
    Dim cmd As SqlCommand = New SqlCommand

    txtResults.Clear()

    ' Set command's connection and command text
    cmd.Connection = cnn
    cmd.CommandType = CommandType.Text
    cmd.CommandText = "Select au_lname,state from authors"
```

```
    ' Write out command string
    txtResults.Text = "Command String:" & ControlChars.CrLf
    txtResults.Text = txtResults.Text & ControlChars.Tab & _
            cmd.CommandText() & ControlChars.CrLf
End Sub
```

קוד #C-2.2 קוד לפתיחת התחברות מסד נתונים והכנת אובייקט command

```
private void btnCommand_Click(object sender, System.EventArgs e)
 {
        // Create an instance of an Connection object
        SqlConnection cnn = new SqlConnection(
                        "server=localhost;uid=sa;database=pubs");

        // Create instances of Command and paramter objects
        SqlCommand cmd = new SqlCommand();
        txtResults.Clear();

        // Set command's connection and command text
        cmd.Connection = cnn;
        cmd.CommandType = CommandType.Text;
        cmd.CommandText = "Select au_lname,state from authors";

        // Write out command string
        txtResults.Text = "Command String:" + "\n\r";
        txtResults.Text = txtResults.Text + "\t" + cmd.CommandText + "\n\r";
}
```

כאשר תפעיל את הפרויקט DataProviderObjects ותלחץ על Command, תיבת הטקסט תציג את הצהרת SQL שהקצית כ-CommandText באובייקט SqlCommand:

```
SELECT au_lname, state FROM authors
```

הערה

מחלקות רבות של .NET Framework, וגם מחלקות שנכתבו על ידי מפתחים אחרים, מכילות בנאים (constructors) רבים של אובייקטים. במילים אחרות, יש מספר דרכים ליצירת מופע (instance) חדש של המחלקה, כאשר כל בנאי מקבל ערכה שונה של ארגומנטים. תוכל להשתמש בגירסה המתאימה ביותר לשימוש או לצרכים הנוכחיים.

הבנאי שנמצא בשימוש בקוד 2.2 באובייקט SqlConnector שונה מזה שבשימוש בקוד 2.1. בקוד זה השתמשנו תחילה בבנאי ברירת המחדל, אשר אינו מקבל אף פעם ארגומנט. לאחר מכן הקצנו את מחרוזת ההתחברות לאובייקט SqlConnection על ידי הגדרת התכונה ConnectionString. להלן התוצאה:

```
' Create an instance of an Connection object
Dim cnn As SqlConnection = New SqlConnection()
' Set the connection string
cnn.ConnectionString = "server=localhost; uid=sa;database=pubs"
```

בקוד 2.2 השתמשנו בבנאי עבור האובייקט SqlConnection אשר מקבל מחרוזת התחברות כפרמטר. הדבר מאפשר ליצור את האובייקט ולהקצות לו מחרוזת התחברות בשורת קוד יחידה:

```
' Create an instance of an Connection object
Dim cnn As SqlConnection = New SqlConnection ( _
                                "server=localhost;uid=sa;database=pubs")
```

שימוש באובייקט Command עם פרמטרים ושגרות מאוחסנות

בעת ביצוע שאילתות או פקודות על מקור נתונים, עליך להעביר פעמים רבות ערכי פרמטרים. דבר זה נכון כמעט תמיד בעת ביצוע פעולה (פקודות Insert, Update או Delete) וקריאה לשגרה מאוחסנת (Stored Procedure). כדי לעמוד בדרישות אלו, האובייקט Command מכיל את התכונה Parameters, שהוא האובייקט ParameterCollection המכיל אוסף של אובייקטי Parameter. שוב, תכונה זו דומה מאוד ל- ADO 2.X.

אובייקט Parameter (וגם ParameterCollection) קשור בצורה הדוקה לספק הנתונים שלו, ולכן זהו אחד האובייקטים שיש ליישם כחלק מספק הנתונים של ADO.NET. יש הבדל משמעותי בין תכנות באמצעות SqlParameterCollection לבין תכנות OdbcParameterCollection ו-OledbParameterCollection.

האובייקטים OdbcParameterCollection ו-OledbParameterCollection מבוססים על פרמטרים של מיקום (positional parameters) ואילו SqlParameterCollection מבוסס על פרמטרים בעלי שמות (named parameters). הבדל זה משפיע על דרך ההגדרה של שאילתות ופרמטרים.

כעת נתחיל בביצוע שאילתת פרמטר פשוטה על הטבלה authors במסד הנתונים pubs. נניח שברצונך להחזיר את כל שמות המחברים ממדינה מסוימת. אם אתה משתמש בספק הנתונים Oledb או Odbc, השאילתה תיראה כך:

```
SELECT state, au_fname, au_lname FROM authors WHERE state = ?
```

שים לב ששומר המקום (ה-placeholder) של הפרמטר הוא סימן שאלה. שומרי מקום לפרמטרים אחרים יכולים להיות אף הם סימני שאלה. האבחנה בין פרמטרים נעשית על פי מיקומם. כלומר, הסדר בו הם מתווספים ל-ParameterCollection חייב להתאים בדיוק לסדר בו הם מופיעים בשאילתה או בשגרה המאוחסנת.

מצד שני, אם אתה משתמש בספק הנתונים SqlClient, השאילתה תיראה כך:

```
SELECT state, au_fname, au_lname FROM authors WHERE state = @MyParam
```

כאשר שומר המקום של הפרמטר הוא שם הפרמטר הספציפי. פרמטרים נוספים יצוינו על ידי שמותיהם הספציפיים. האבחנה בין פרמטרים נעשית לפי שמם, ולכן ניתן להוסיף אותם ל-ParameterCollection בכל סדר רצוי.

תוכל ליצור במפורש אובייקט Parameter באמצעות הבנאי Parameter (כלומר, New) או על ידי העברת הארגומנטים הדרושים לשיטה Add של האובייקט ParameterCollection – התכונה Parameters של האובייקט Command. זכור גם שלכל אחת מהשיטות האלו – הבנאי Parameter והשיטה Add – יש מספר גדול של אפשרויות.

הנה דרך אחת להוסיף פרמטר לפקודה על ידי יצירה מפורשת של אובייקט הפרמטר:

```
Dim myParameter As New OdbcParameter ("MyParam", OdbcType.Char, 2)
myParameter.Direction = ParameterDirection.Input
myParameter.Value = "CA"
cmd.Parameters.Add (myParameter)
```

והנה דרך להוסיף פרמטר לפקודה על ידי העברת הארגומנטים לשיטה Add:

```
cmd.Parameters.Add("@MyParam", OdbcType.Char, 2)
cmd.Parameters("@MyParam").Direction = ParameterDirection.Input
cmd.Parameters("@MyParam").Value = "CA"
```

השיטה השנייה קצרה יותר ומועדפת, בדרך כלל במקרים בהם יש סיבה לעשות שימוש חוזר באובייקט Parameter.

עליך לספק שם פרמטר ואת הסוג והאורך שלו (אם יש צורך בכך) לשיטה Add של Parameter. לאחר מכן תוכל לקבוע את הכיוון כ-Input, Output, InputOutput או ReturnValue. כיוון ברירת המחדל הוא Input. לבסוף, אם מסופק ערך עבור פרמטר זה, הקצה אותו לתכונה Value של האובייקט Parameter. תוכל להגדיר מספר תכונות נוספות, כולל Scale, Presicion ו-IsNullable.

אם היית משתמש בספק הנתונים SqlClient, הקוד בו היית משתמש היה כמעט זהה. ההבדלים היחידים הם שהקידומות Odbc היו מוחלפות בקידומות Sql וגם מניית הסוג (enum) היתה נקראת SqlDbType:

VB
```
Dim myParameter As New SqlParameter("MyParam", SqlDbType.Char, 2)
myParameter.Direction = ParameterDirection.Input
myParameter.Value = "CA"
cmd.Parameters.Add (myParameter)
```

C#
```
SqlParameter myParameter = new SqlParameter("@MyParam",SqlDbType.Char,2) ;
myParameter.Direction = ParameterDirection.Input ;
myParameter.Value = "CA" ;
cmd.Parameters.Add(myParameter) ;
```

או

```
VB  cmd.Parameters.Add("@MyParam", SqlDbType.Char, 2)
    cmd.Parameters("@MyParam").Direction = ParameterDirection.Input
    cmd.Parameters("@MyParam").Value = "CA"

C#  cmd.Parameters.Add(New SqlParameter("@MyParam", SqlDbType.Char, 2)) ;
    cmd.Parameters("@MyParam").Value = "CA" ;
```

טיפ

הדרך להעברה (מוצלחת) של ערך ריק (null) עבור פרמטר, היא באמצעות התכונה Value של האובייקט DBNull. שורת הקוד היא:

```
cmd.Parameters("@MyParam").Value = DBNull.Value
```

שנה את קוד הלחצן btnCommand, כפי שמוצג בקוד 2.3. כאשר תפעיל את התוכנית ותלחץ btnCommand, יוצגו טקסט השאילתה ושם הערך של הפרמטר.

קוד 2.3-VB קוד להכנה והצגת פקודות ואובייקטי פרמטרים

```
Private Sub btnCommand_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnCommand.Click

    ' Create an instance of an Connection object
    Dim cnn As SqlConnection = New SqlConnection( _
                                "server=localhost;uid=sa;database=pubs")

    ' Create instances of Command and paramter objects
    Dim cmd As SqlCommand = New SqlCommand
    Dim prm As SqlParameter = New SqlParameter

    txtResults.Clear()

    ' Open the Connection
    cnn.Open()

    ' Set command's connection and command text
    cmd.Connection = cnn
    cmd.CommandType = CommandType.Text
    cmd.CommandText = _
          "Select au_lname,state from authors where state = @MyParam"

    'Create parameter and set value
    cmd.Parameters.Add(New SqlParameter("@MyParam", SqlDbType.Char, 2))
```

```
    cmd.Parameters("@MyParam").Value = "CA"

    ' Write out command string
    txtResults.Text = "Command String:" & ControlChars.CrLf
    txtResults.Text = txtResults.Text & ControlChars.Tab & _
                      cmd.CommandText() & ControlChars.CrLf

    ' Write out command parameters and values
    txtResults.Text = txtResults.Text & "Command parameters:" & _
                      ControlChars.CrLf
    For Each prm In cmd.Parameters
       txtResults.Text = txtResults.Text & ControlChars.Tab & _
                         prm.ParameterName & "  =  " & prm.Value & _
                         ControlChars.CrLf
    Next

  End Sub
```

קוד #C-2.3 קוד להכנה והצגה של פקודות ואובייקטים של פרמטרים

```
private void btnCommand_Click(object sender, System.EventArgs e)
   {
      // Create an instance of an Connection object
      SqlConnection cnn = new SqlConnection(
                              "server=localhost;uid=sa;database=pubs");

      // Create instances of Command and paramter objects
      SqlCommand cmd = new SqlCommand();
      SqlParameter prm  = new SqlParameter();

      txtResults.Clear();

      // Open the Connection
      cnn.Open();

      // Set command's connection and command text
      cmd.Connection = cnn;
      cmd.CommandType = CommandType.Text;
      cmd.CommandText =
           "Select au_lname,state from authors where state = @MyParam";

      // Create parameter and set value
      cmd.Parameters.Add
           (new  SqlParameter("@MyParam", SqlDbType.Char, 2));
     cmd.Parameters["@MyParam"].Value = "CA";
```

```
    // Write out command string
    txtResults.Text = "Command String:" + "\n\r";
    txtResults.Text = txtResults.Text + "\t" + cmd.CommandText + "\n\r" ;

    // Write out command parameters and values
    txtResults.Text = txtResults.Text + "Command parameters:" + "\n\r" ;

   for (int j=0; j<cmd.Parameters.Count; j++)
   {
      prm = cmd.Parameters[j];
      txtResults.Text = txtResults.Text + "\t" +
                        prm.ParameterName + "  =  " + prm.Value + "\n\r";
   }
  }
```

באותה דרך ניתן לקרוא לשגרות מאוחסנות, פרט לכך ש-CommandType הוא CommandType.StoredProcedure ולא CommandType.Text. שם השגרה המאוחסנת מוקצה לתכונה CommandText. לכן, קריאה לשגרה המאוחסנת GetAuthorsFromState, אשר מצפה לפרמטר באורך של שני סימנים, תיראה כך:

VB
```
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = "GetAuthorsFromState"
cmd.Parameters.Add("@MyParam", SqlDbType.Char, 2)
cmd.Parameters("@MyParam").Direction = ParameterDirection.Input
cmd.Parameters("@MyParam").Value = "CA"
```

C#
```
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "GetAuthorsFromState";
cmd.Parameters.Add(new SqlParameter("@MyParam", SqlDbType.Char,2));
cmd.Parameters["@MyParam"].Direction = ParameterDirection.Input;
cmd.Parameters["@MyParam"].Value = "CA";
```

טיפ

כאשר אתה קורא לשגרה מאוחסנת באמצעות OdbcCommand, עליך לוודא שאתה משתמש ברצפי escape הסטנדרטיים של שגרה מאוחסנת מסוג ODBC, ולא לציין את שם השגרה ב-CommandText בלבד. סימני שאלה משמשים כשומרי מקום לפרמטרים ברצפי escape. המקביל של OdbcCommand בחלק הקוד הקודם הוא:

```
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = "{GetAuthorsFromState ?}"
cmd.Parameters.Add("@MyParam", OdbcType.Char, 2)
cmd.Parameters("@MyParam").Direction = ParameterDirection.Input
cmd.Parameters("@MyParam").Value = "CA"
```

אם השגרה המאוחסנת מחזירה גם ערך, ערך זה יצוין בשם השגרה הקודמת על ידי " ? =", כמו בדוגמה הבאה:

```
cmd.CommandText = "{? = GetAuthorsFromState ?}"
```

אם אתה צופה כי שגרה מאוחסנת (Stored Procedure) תחזיר ערך, עליך לציין את הכיוון כ-Output ולקרוא את התכונה Value של הפרמטר לאחר הפעלת השגרה המאוחסנת. לדוגמה, אנו מגדירים בנוסף ערך החזרה מהשגרה המאוחסנת. הסוג Int של SQL Server צוין, ולכן אין צורך לציין אורך עבור הפרמטר. האורך מוגדר מלכתחילה כארבעה בתים:

VB
```
cmd.Parameters.Add(New SqlParameter("result", SqlDbType.Int))
cmd.Parameters("result").Direction = ParameterDirection.ReturnValue
cmd.Parameters.Add(new SqlParameter("@MyParam", SqlDbType.Int))
cmd.Parameters("@MyParam").Direction = ParameterDirection.Output
' Call stored procedure here
MsgBox (cmd.Parameters("@MyParam").Value)
```

C#
```
cmd.Parameters.Add(New SqlParameter("result", SqlDbType.Int)) ;
cmd.Parameters("result").Direction = ParameterDirection.ReturnValue ;
cmd.Parameters.Add(new SqlParameter("@MyParam", SqlDbType.Int)) ;
cmd.Parameters("@MyParam").Direction = ParameterDirection.Output ;
' Call stored procedure here
MessageBox.Show(cmdParameters["@MyParam"].Value) ;
```

הערה

כאשר אתה מגדיר פרמטר אשר ישמש כערך החזרה (ReturnValue) של שגרה מאוחסנת, עליך להגדיר אותו כפרמטר הראשון אשר מתווסף לאוסף Parameters. הגדרה זו נחוצה עבור הפרמטרים Oledb ו-Odbc, מכיון שכפי שציינו קודם לכן, פרמטרים אלה מזוהים על פי מיקומם, ואילו ערך מוחזר צריך להיות במיקום הראשון. עם זאת, כאשר אתה עובד עם פרמטרים של SQL, תוכל למקם את פרמטר ערך ההחזרה בכל מקום רצוי, מכיון שפרמטרים אלה מזוהים באמצעות שמם.

מייד נציג דוגמאות קוד נוספות אשר מדגימות את השימוש בפרמטרים וניתן הסברים להפעלת פקודות אלו.

הפעלת הפקודות

עד עתה למדת כיצד להגדיר את התכונות והפרמטרים השונים של אובייקט מסוג Command, אך עדיין לא הפעלת אף אחת מהפקודות האלו. יש שלוש שיטות סטנדרטיות להפעלת הפקודות אשר מוגדרות על ידי האובייקט Command, ושיטה נוספת הזמינה באובייקט SqlCommand בלבד:

- ExecuteNonQuery – הפעלת פקודת SQL שאינה מחזירה רשומות.
- ExecuteScalar – הפעלת פקודת SQL והחזרת העמודה הראשונה בשורה הראשונה.
- ExecuteReader – הפעלת פקודת SQL והחזרת ערכת הרשומות המתקבלת באמצעות האובייקט DataReader.
- ExecuteXmlReader (SqlCommand בלבד) – הפעלת פקודת SQL והחזרת ערכת הרשומות המתקבלת כ-XML באמצעות האובייקט XmlReader.

כעת נתבונן בשלוש שיטות ההפעלה המשותפות הראשונות. הדיון בשיטה הרביעית לא יידון במסגרת ספר זה.

ExecuteNonQuery

השיטה ExecuteNonQuery היא אולי הדרך המתאימה ביותר להפעיל פקודות על מקור נתונים; היא מאפשרת להפעיל פקודות שאינן מחזירות ערכים כלשהם (resultset או scalar), פרט לערך המציין את ההצלחה או הכישלון של הפקודה. שיטה זו היא גם היעילה ביותר להפעלת פקודות על מקור נתונים. תוכל להפעיל הצהרת SQL או שגרה מאוחסנת שהיא:

1. פקודת Catalog או Data Definition Language (DDL), אשר יכולה ליצור או לשנות אובייקטי מסדי נתונים כגון טבלאות, תצוגות או שגרות מאוחסנות.

או

2. פקודת Update (Update, Insert או Delete), אשר משנה נתונים במסד הנתונים.

הערה

השיטה ExecuteNonQuery מחזירה ערך integer יחיד. משמעות ערך החזרה זה תלויה בסוג הפקודה המבוצעת.

אם אתה מפעיל פקודת Catalog או DDL לשינוי מבני מסד נתונים, ערך ההחזרה של השיטה יהיה 1-, כאשר הפעולה מסתיימת בהצלחה. אם אתה מעדכן רשומות באמצעות הצהרת Update, Insert או Delete, ערך ההחזרה יהיה מספר השורות המושפעות על ידי הפעולה. בכל המקרים האלה, ערך ההחזרה של השיטה יהיה 0 אם הפעולה נכשלה.

נמשיך עם הפרויקט DataProviderObjects. כעת תשתמש באובייקטים שבמרחב השמות בשם Oledb, ותעבוד עם מסד הנתונים pubs. מוטל עליך ליצור טבלה חדשה עבור מסד נתונים זה על ידי הפעלת הפקודה DDL הדרושה לשם כך. טבלה זו תמפה בין קודי המיקוד (zip) למדינות מתאימות. להלן הצהרת SQL ליצירת הטבלה הדרושה:

```
CREATE TABLE tblStateZipCodes (ZipCode char (5) NOT NULL,
        State char (2) NOT NULL)
```

כעת שנה את Form1 המקורי על ידי ביצוע הפעולות שלהלן:

1. פתח את Form1 ב- Visual Studio IDE.
2. בפינה השמאלית העליונה של הטופס הוסף לחצן מהכרטיסיה Windows Forms של תיבת הכלים.
3. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**btnNonQuery** וקבע את ערך התכונה Text ל-**ExecuteNonQuery**.

כעת הוסף את קוד 2.4 עבור האירוע Click של לחצן חדש זה.

קוד 2.4-VB קוד ליצירת טבלת מסד נתונים, באמצעות שימוש באובייקטים ממרחב השמות Oledb

```
Private Sub btnNonQuery_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnNonQuery.Click

    ' Create an instance of an Connection object
    Dim cnn As OleDbConnection = New OleDbConnection( _
            "provider=SQLOLEDB;server=localhost;uid=sa;database=pubs")
    Dim sql As String
    Dim result As Integer

    ' Create instance of Command object
    Dim cmd As OleDbCommand = New OleDbCommand

    ' Set command's connection and command text
    cmd.Connection = cnn
    cmd.CommandType = CommandType.Text

    ' Assign SQL statement to create a new table
    sql = "CREATE TABLE  tblStateZipCodes ( " & _
            "ZipCode char (5)  NOT NULL, " & _
            "State char (2)  NOT NULL )"
    cmd.CommandText = sql

    ' Open the Connection before calling ExecuteNonQuery()
```

```
    cnn.Open()

    ' We need to put the code inside a Try-Catch block
    ' since a failed command ALSO generates a run-time error
    Try
        result = cmd.ExecuteNonQuery()
    Catch ex As Exception
        ' Display error message
        MessageBox.Show(ex.Message)
    End Try

    ' Show results of command execution
    If result = -1 Then
        MessageBox.Show("Command completed successfully")
    Else
        MessageBox.Show("Command execution failed")
    End If
    cnn.Close()

End Sub
```

קוד #C-2.4 קוד ליצירת טבלת מסד נתונים, באמצעות שימוש באובייקטים ממרחב השמות Oledb

```
private void btnNonQuery_Click(object sender, System.EventArgs e)
{
  // Create an instance of an Connection object
  OleDbConnection cnn  = new OleDbConnection(
            "provider=SQLOLEDB;server=localhost;uid=sa;database=pubs");
  String sql = null;
  int result = 0;

  // Create instance of Command object
  OleDbCommand cmd  = new OleDbCommand();

  // Set command's connection and command text
  cmd.Connection = cnn;
  cmd.CommandType = CommandType.Text;

  // Assign SQL statement to create a new table
  sql = "CREATE TABLE  tblStateZipCodes ( " +
        "ZipCode char (5)  NOT NULL, " +
        "State char (2)  NOT NULL )";
```

```
cmd.CommandText = sql;

// Open the Connection before calling ExecuteNonQuery()
cnn.Open();

// We need to put the code inside a Try-Catch block
// since a failed command ALSO generates a run-time error
try
{
        result = cmd.ExecuteNonQuery();
}

catch (Exception ex)
{
        // Display error message
        MessageBox.Show(ex.Message);
}


// Show results of command execution
if (result == -1)
{
        MessageBox.Show("Command completed successfully");
}
else
{
        MessageBox.Show("Command execution failed");
}

cnn.Close();
}
```

כאשר תפעיל את הפרויקט DataProviderObjects ותלחץ על ExecuteNonQuery בפעם הראשונה, תופיע תיבת הודעה המציינת כי הפקודה הסתיימה בהצלחה. תוכל לוודא שהטבלה נוצרה בהצלחה על ידי בחינת רשימת הטבלאות במסד הנתונים Novelty, באמצעות Visual Studio Server Explorer או SQL Server Enterprise Manager (פרק 1).

אם תלחץ לאחר מכן על הלחצן ExecuteNonQuery פעם נוספת, יופיעו שתי תיבות הודעה. התיבה הראשונה מכילה את טקסט ההודעה משגיאת החריגה (exception) שנוצרה. תיבה זו מוצגת מתוך בלוק catch אשר מציין את הסיבה המדויקת לכישלון. במקרה זה, הפקודה נדחתה מכיון שטבלה בשם זה כבר קיימת במסד הנתונים. תיבת הודעה שנייה גם תוצג ותודיע כי הפעלת הפקודה נכשלה.

בדרך זו תוכל ליצור תצוגה, מבט (View) או שגרה מאוחסנת (Stored Procedure). ליצירת התצוגה EmployeeJobs_View אשר תחזיר ערכת תוצאות המכילה כותרות עבודות ושמות עובדים (ממוינות על פי תיאור העבודה), שנה את הצהרת SQL בקוד 2.3 באופן הבא:

```
VB  sql = "CREATE VIEW EmployeeJobs_View AS" & _
          "SELECT TOP 100 PERCENT jobs.job_desc," & _
          "employee.fname, employee.lname" & _
          "FROM jobs INNER JOIN" & _
          "employee ON jobs.job_id = employee.job_id" & _
          "ORDER BY jobs.job_desc"

C#  sql = "CREATE VIEW EmployeeJobs_view AS " +
          "SELECT TOP 100 PERCENT jobs.job_desc, " +
          "employee.fname, employee.lname " +
          "FROM jobs INNER JOIN " +
          "employee ON jobs.job_id = employee.job_id " +
          "ORDER BY jobs.job_desc " ;
```

כדי ליצור שגרה מאוחסנת אשר מקבלת פרמטר יחיד ומחזירה ערך כערך החזרה, שנה את הצהרת SQL כפי שמוצג בקוד 2.5.

הערה

כדי לכלול משפט ORDER BY בתוך שאילתה המגדירה **תצוגה** (View) לשם מיון נתונים, עליך לכלול משפט TOP בתוך הצהרת ה-SQL.

קוד 2.5-VB קוד המכיל הצהרת SQL ליצירת השגרה המאוחסנת AuthorsInState1

```
sql = "CREATE PROCEDURE AuthorsInState1 @State char(2)" & _
      " AS declare @result int" & _
      " select  @result = count (*) from authors " & _
      " where state = @State" & _
      " return (@result)"
```

קוד 2.5-C# קוד המכיל הצהרת SQL ליצירת השגרה המאוחסנת AuthorsInState1

```
sql = "CREATE PROCEDURE AuthorsInState1 @State char(2)" +
      " AS declare @result int" +
      " select  @result = count (*) from authors " +
      " where state = @State" +
      " return (@result)" ;
```

הערה

השיטה ExecuteNonQuery מחזירה ערך יחיד בלבד, אך אם תגדיר פרמטרי פלט (output) או פרמטר ערך החזרה עבור הפקודה, הם ימולאו בנתוני הפרמטר המתאימים. גישה זו יעילה יותר מהפעלת פקודה שמחזירה result set או ערך scalar.

כעת נפנה לסוג השני של פקודה שאינה שאילתה – nonquery. זו יכולה להיות פקודה לעדכון מסד נתונים (database update), כמו Update, Insert או Delete. פקודות אלו דורשות בדרך כלל פרמטרים, ובמיוחד במקרים שאתה משתמש בשגרות מאוחסנות לביצוע הפעולות האלו (ברוב המקרים תשתמש בשגרות כאלו לשיפור ביצועים).

כעת נמשיך עם Form1 של הפרויקט DataProviderObjects. נניח שהמו"ל אשר יצר את מסד הנתונים pubs החליט להעניק תמלוגים למחברים. הוספת לחצן פקודה ותיבת טקסט לטופס תאפשר להזין את הגידול בתמלוגים כפרמטר בפקודה Update. ניתן לעשות זאת בצורה הבאה:

1. הוסף לחצן מתחת ללחצן cmdExecuteNonQuery.
2. בחלון Properties קבע את ערך התכונה Name של הלחצן ל-**cmdUpdate** וקבע את ערך התכונה Text ל-**Update**.
3. הוסף תיבת טקסט מתחת ללחצן חדש זה מהכרטיסיה Windows Forms של תיבת הכלים.
4. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**txtParam1** ואת ערך התכונה Text ל-**0**. הגדרת ערך 0 עבור txtParam1 מבטיחה כי אם תפעיל את התוכנית ותשכח לקבוע ערך זה לפני לחיצה על Update, לא ייגרם נזק ולא תהיה שגיאת הפעלה (run-time error).
5. הוסף את הקוד עבור לחצן חדש זה, כפי שמוצג בקוד 2.6.

קוד 2.6-VB קוד לעדכון טבלת מסד נתונים, באמצעות שימוש בהצהרת SQL עם פרמטר

```
Private Sub btnUpdate_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnUpdate.Click

    Dim result As Integer

    ' Create an instance of an Connection object
    Dim cnn As SqlConnection = New SqlConnection( _
                "server=localhost;uid=sa;database=pubs")

    ' Create instance of Command object
    Dim cmd As SqlCommand = New SqlCommand()

    txtResults.Clear()
```

```
    ' Set command's connection and command text
    cmd.Connection = cnn
    cmd.CommandType = CommandType.Text
    cmd.CommandText = "UPDATE roysched SET royalty = royalty + @param1"

    ' Create parameter and set value
    cmd.Parameters.Add(New SqlParameter("@param1", SqlDbType.Int))
    cmd.Parameters("@param1").Direction = ParameterDirection.Input
    cmd.Parameters("@param1").Value = Val(txtParam1.Text)

    ' Open the Connection before calling ExecuteReader()
    cnn.Open()

    result = cmd.ExecuteNonQuery()
    MessageBox.Show(result & " records updated", "DataProviderObjects")

    cnn.Close()
End Sub
```

קוד #C-2.6 קוד לעדכון טבלת מסד נתונים, באמצעות שימוש בהצהרת SQL עם פרמטר

```
private void btnUpdate_Click(object sender, System.EventArgs e)
{
  int result;

  // Create an instance of an Connection object
  SqlConnection cnn = new SqlConnection(
            "server=localhost;uid=sa;database=pubs");

  // Create instance of Command object
  SqlCommand cmd = new SqlCommand();

  txtResults.Clear();

// Set command's connection and command text
cmd.Connection = cnn;
cmd.CommandType = CommandType.Text ;
cmd.CommandText = "UPDATE roysched SET royalty = royalty + @param1"

  // Create parameter and set value
  cmd.Parameters.Add(new SqlParameter("@param1", SqlDbType.Int));
  cmd.Parameters["@param1"].Direction = ParameterDirection.Input;
```

```
    cmd.Parameters["@param1"].Value = txtParam1.Text;

    // Open the Connection before calling ExecuteReader()
    cnn.Open();

    result = cmd.ExecuteNonQuery();
    MessageBox.Show(result + " records updated", "DataProviderObjects");

    cnn.Close();
  }
```

באפשרותך לעדכן את הטבלה roysched באמצעות הרצת הפרויקט, קביעת הערך המספרי בתיבת הטקסט ולחיצה על הלחצן Update. צריכה להופיע תיבת הודעה המציינת את מספר הרשומות שעודכנו. באפשרותך לבדוק את התוצאה באמצעות Server Enterprise Manager, והצגת הנתונים מהטבלה roysched לפני ואחרי ביצוע פעולת הלחיצה על הלחצן.

את אותה פעולת עדכון אפשר גם לבצע באמצעות שגרה מאוחסנת. היתרונות שלה הם שהביצועים יותר טובים וכל הקוד של השאילתות מרוכז במקום אחד. החסרונות שלה הם שיהיה עליך להחזיק איש DBA – או לפחות אחד שיודע כיצד כותבים שגרה מאוחסנת – כחבר בקבוצת פיתוח התוכנה שלך. בארגון גדול, יעבורו מספר ימים עד שאיש DBA יבצע שינויים בקוד של מספר שגרות מאוחסנות. אם באפשרותך לבצע את העבודה לבדך, תהליך זה יארך לך זמן קצר מאוד. הדרך להוספת שגרה כזו תוארה כבר בפרק 1. כחלופה לשיטה זו, ניתן להשתמש בפרויקט DataProviderObjects, כפי שעשינו קודם, כאשר עליך רק לשנות את הקוד הקיים, כדי שיתאים לצרכיך.

השגרה המאוחסנת עבור פעולת העדכון אמורה להיראות כך:

```
CREATE PROCEDURE UpdateRoyalties
@param1 int
AS
UPDATE roysched SET royalty  = royalty + @param1
```

בקוד 2.6 עליך לשנות את התכונות CommandType ו-CommandText של האובייקט Command כדי לבצע קריאה לשגרה המאוחסנת.

שתי השורות של הקוד הן כעת:

```
VB   cmd.commandType = CommandType.StoredProcedure
     cmd.CommandText = "UpdateRoyalties"

C#   cmd.commandType = CommandType.StoredProcedure ;
     cmd.CommandText = "UpdateRoyalties" ;
```

הרצת התוכנית המעודכנת צריכה לבצע את אותו דבר כמקודם. כרגע, העדכון התבצע באמצעות שגרה מאוחסנת, לעומת הגירסה הקודמת של קוד התוכנית שבה בוצע הדבר באמצעות השאילתה SQL.

ExecuteScalar

לעיתים ברצונך לבצע פעולה על בסיס הנתונים המחזירה ערך סקלרי (Scalar Value). כלומר, ערך מספרי יחיד. דוגמה טיפוסית לדבר כזה היא שאילתת סיכום המבצעת פונקציית סיכום פנימית כלשהי – כגון, SUM או COUNT. דוגמאות אחרות הן טבלת חיפוש המחזירה ערך יחיד או פקודה המחזירה ערך בוליאני (לדוגמה, ניקח את החיפוש של שם וסיסמה בטבלת משמשים כלשהי, כאשר התוצאה הנדרשת היא 1, במידה והמשתמש קיים). השיטה ExecuteScalar מבצעת את הפקודה ומחזירה את העמודה הראשונה של הרשומה הראשונה מקבוצת הרשומות המוחזרת. לשאר העמודות או הרשומות אין התייחסות.

נוסיף את השגרה המאוחסנת הבאה למסד הנתונים pubs:

```
CREATE PROCEDURE AuthorsInState2
@param1 char(2)
AS
SELECT count(*) FROM authors WHERE state = @param1
```

השגרה המאוחסנת AuthorsInState2 מקבלת פרמטר אחד המכיל קוד של מדינה באורך שני תווים ומחזירה מהטבלה authors את מספר המחברים באותה מדינה. שגרה זו זהה מבחינת הפונקציונליות שלה לשגרה AuthorsInState1, המופיעה בקוד 2.5, אשר מחזירה ערכת רשומות במקום החזרת התוצאה הדרושה.

הערה

יש מעט ירידה בביצועים אם אתה משתמש ב-ExecuteScalar במקום השימוש ב-ExecuteNonQuery והעברת הערך המספרי כפרמטר מוחזר. אם כן, לשם מה להשתמש בשיטה זו? התשובה לכך היא – שהדבר פשוט יותר ודורש פחות טרחה, חוסך ממך את הצורך להגדיר פרמטרים גם בהגדרת הפקודה וגם בקריאה אליה.

אתה קורא לשיגרה זו באמצעות ספק הנתונים של ODBC.

1. הוסף לחצן נוסף מתחת לתיבת הטקסט txtParam1.
2. בחלון Properties קבע את התכונה של הלחצן ל-**cmdScalar** ואת התכונה Text ל-**ExecuteScalar**.
3. הוסף את הקוד עבור לחצן חדש זה, כפי שמוצג בקוד 2.7.

קוד 2.7-VB קוד להחזרת ערך מספרי משגרה מאוחסנת, שמשתמש ב- Odbc Data Provider

```
Private Sub btnExecuteScalar_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExecuteScalar.Click

    Dim result As Integer

    ' Create an instance of an Connection object
    Dim cnn As OdbcConnection = New OdbcConnection( _
     "DRIVER={SQL Server};server=localhost;uid=sa;database=pubs")

    ' Create instance of Command object
    Dim cmd As OdbcCommand = New OdbcCommand()

    txtResults.Clear()

    ' Set command's connection and command text
    cmd.Connection = cnn
    cmd.CommandType = CommandType.StoredProcedure
    cmd.CommandText = "{call AuthorsInState2(?) }"

    ' Create parameter and set value
    cmd.Parameters.Add("@param1", OdbcType.Char, 2)
    cmd.Parameters("@param1").Value = txtParam1.Text

    ' Open the Connection before calling ExecuteReader()
    cnn.Open()

    result = cmd.ExecuteScalar()
    MessageBox.Show("Count is " & result, "DataProviderObjects")

    cnn.Close()
End Sub
```

קוד 2.7-C# קוד להחזרת ערך מספרי משגרה מאוחסנת, שמשתמש ב- Odbc Data Provider

```
private void btnExecuteScalar_Click(object sender, System.EventArgs e)
    {
     int result;

     OdbcConnection cnn  = new OdbcConnection(
                    "DRIVER={SQL Server};server=localhost;uid=sa;database=pubs");
```

```
    // Create instance of Command object
    OdbcCommand cmd  = new OdbcCommand();

    txtResults.Clear();

    // Set command's connection and command text
    cmd.Connection = cnn;
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.CommandText = "{call AuthorsInState2(?) }";

    // Create parameter and set value
    cmd.Parameters.Add("@param1", OdbcType.Char, 2);
    cmd.Parameters["@param1"].Value = txtParam1.Text;

    // Open the Connection before calling ExecuteReader()
    cnn.Open();

    String resultRes = cmd.ExecuteScalar().ToString();
    MessageBox.Show("Count is " + resultRes, "DataProviderObjects");

    cnn.Close();

}
```

הפעל את היישום והזן לתיבת הטקסט parameter קוד מדינה בן שני תווים. כאשר תלחץ על ExecuteScalar, תוצג תיבת הודעה המציינת את מניית המחברים באותה המדינה. תוכל לבדוק תוצאה זו על ידי הצגת נתונים מהטבלה authors באמצעות SQL Server Enterprise Manager.

הערה

נתוני ברירת המחדל של מסד הנתונים pubs אמורים להחזיר מנייה של 2 עבור המדינה UT (המדינה יוטה בארה"ב) ומנייה של 15 עבור המדינה CA (קליפורניה).

ExecuteReader

במובנים מסוימים, שמרנו את הדברים החשובים ביותר לסוף. בשיטה ExecuteReader תשתמש כדי לבצע פעולה אשר מחזירה קבוצת רשומות (שורות). ברוב יישומי מסדי הנתונים, זו תהיה ככל הנראה שיטת ההפעלה המועדפת לרוב המקרים. בדרך זו מופעלת פקודה אשר מחזירה תוצאה באמצעות האובייקט DataReader, אשר מוצגת כקבוצת רשומות נתונים. סריקת הרשומות (שורות) מתבצעת זו אחר זו, החל מהרשומה (שורה) הראשונה. בסעיף הבא יינתנו פרטים נוספים אודות DataReader וגם דוגמאות.

האובייקט DataReader

האובייקט DataReader מספק שטף נתונים ללא חוצץ (nonbuffered) לקריאה בלבד בכיוון ההתקדמות בלבד (forward only) של הרשומות (שורות), אשר נוצרות על ידי השיטה ExecuteReader של האובייקט Command. בעיקרון, האובייקט DataReader שווה ערך ל- read-only recordset ב- ADO 2.X, אשר מיועד לקריאה בכיוון קדימה בלבד. האובייקט אינו תומך בגלילה או בעדכון ומהווה את הדרך המהירה ביותר לגישה לנתונים ממקור נתונים. הנתונים אינם נשמרים במאגר או במטמון, ולכן שיטה זו טובה לאחזור כמויות גדולות של נתונים. קריאה לשיטה Read תקדם את DataReader לרשומה הבאה.

ניתן לגשת לשדות של כל שורה באמצעות סוג חזק של accessors (strongly typed accessors), בנוסף לאוסף Fields. הגישה לנתוני שדה באמצעות האוסף Fields – על ידי שם שדה ללא תלות בסוג שלו – נעשית באופן דומה לגישה לשדות בקבוצת רשומות של ADO 2.X, כגון:

```
VB   X = MyReader ("Myfield")
C#   X = MyReader ("Myfield") ;
```

הערה

ל-DataReader אין בנאי (constructor) מפורש ואינך יכול ליצור מופע (instance) חדש של האובייקט באמצעות ()New. עליך לקרוא לשיטה ExecuteReader של האובייקט Command כדי ליצור מופע חדש ממנו.

לחילופין, אם ידוע לך סוג הנתונים בכל שדה, תוכל לגשת לנתונים באמצעות שיטות מיוחדות לכל סוג. שיטות אלו מחזירות את העמודה המצוינת באמצעות אינדקס מבוסס אפס (zero-based index), כמו לדוגמה:

```
VB   X = MyReader.GetIn16(1)
C#   X = MyReader.GetIn16(1) ;
```

או

```
VB   Str = MyReader.GetString(2)
C#   Str = MyReader.GetString(2) ;
```

הגישה הראשונה אשר משתמשת בגישה פשוטה לפי שם שדה הינה קריאה יותר, קלה יותר לשימוש ותואמת לתוכניות ישנות. הגישה השנייה דורשת אומנם מאמץ רב יותר, אך היא מספקת ביצועים טובים יותר, מכיון שהיא מצמצמת את מספר המרות הסוגים הדרושות לביצועה (החיסרון שלה הוא בכך שסדר העמודות חייב להיות קבוע). במידה וביצעת שינוי כלשהו, בטבלאות ו/או בשאילתה, בדוק את כל המקומות בהם DataReader מקבל נתונים מהשאילתה ועדכן אותם כראוי. בכל מקרה, כדאי לעבוד בשיטה זו.

כעת בצע את התוספת האחרונה לטופס Form1 של הפרויקט DataProviderObjects:

1. הוסף לחצן מתחת ללחצן cmdExecuteScalar.
2. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**cmdExecuteReader** ואת ערך התכונה Text ל-**ExecuteReader**.
3. הוסף את הקוד עבור לחצן חדש זה, כפי שמוצג בקוד 2.8.

הערה

דוגמה זו מציגה כיצד לתכנת את DataReader וגם מציגה מספר תכונות נוספות. לדוגמה, ישנו ערך שלישי ברשימה (enum) בשם CommandType בתכונה CommandType של האובייקט Command. בנוסף ל-Text ו-StoredProcedure, קיים גם הערך TableDirect. אפשרות זו מציינת כי התכונה CommandText מתייחסת לשם טבלה בה כל העמודות מוחזרות על ידי הפקודה. **אפשרות זו נתמכת בספק הנתונים Oledb בלבד.**

בנוסף, ההתייחסות לתצוגות של בסיסי נתונים היא כאל טבלאות, ולכן אתה יכול לקבוע שם לתצוגה במקום לקבוע שם של טבלה, כאשר ערך התכונה CommandText הוא TableDirect.

קוד 2.8-VB קוד ליצירת DataReader ואחזור ערכי שדות, באמצעות שימוש בתצוגת מסד נתונים ובסוג הפקודה TableDirect

```
Private Sub btnExecuteReader_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExecuteReader.Click

    ' Create an instance of an Connection object
    Dim cnn As OleDbConnection = New OleDbConnection( _
        "provider=SQLOLEDB;server=localhost;uid=sa;database=pubs")

    ' Create instance of Command object
    Dim cmd As OleDbCommand = New OleDbCommand

    txtResults.Clear()

    ' Set command's connection and command text
    cmd.Connection = cnn
    cmd.CommandType = CommandType.TableDirect
    cmd.CommandText = "EmployeeJobs_View"

    ' Must open the Connection before calling ExecuteReader()
    cnn.Open()
```

```
    Dim reader As OleDbDataReader

    reader = cmd.ExecuteReader()

    While reader.Read()
        txtResults.Text = txtResults.Text & reader("fname") & _
                          ControlChars.Tab & reader("lname") & _
                          ControlChars.Tab & ControlChars.Tab & _
                          reader("job_desc") & ControlChars.CrLf
    End While
    reader.Close()
    cnn.Close()
End Sub
```

קוד #C-2.8 קוד ליצירת DataReader ואחזור ערכי שדות, באמצעות שימוש בתצוגת מסד נתונים ובסוג הפקודה TableDirect

```
private void btnExecuteReader_Click(object sender, System.EventArgs e)
{
    // Create an instance of an Connection object
    OleDbConnection cnn = new OleDbConnection(
            "provider=SQLOLEDB;server=localhost;uid=sa;database=pubs");

    // Create instance of Command object
    OleDbCommand cmd =  new OleDbCommand();

    txtResults.Clear();

    // Set command's connection and command text
    cmd.Connection = cnn;
    cmd.CommandType = CommandType.TableDirect;
    cmd.CommandText = "EmployeeJobs_View";

    // Must open the Connection before calling ExecuteReader()
    cnn.Open();

    OleDbDataReader reader = null;

    reader = cmd.ExecuteReader();
```

```
while(reader.Read())
{
        txtResults.Text = txtResults.Text + reader.GetString(1) +
                "\t" + reader.GetString(2) +
                "\t"  + "\t" +
                reader.GetString(0) +  "\r\n";
}
reader.Close();
cnn.Close();
}
```

הערה

זכור תמיד לבצע את הפקודה Read() לפני כל ניסיון לגשת לנתונים מ-DataReader. יש להציב את הפקודה DataReader של ADO.NET במפורש בשורה הראשונה על ידי קריאה לשיטה Read תחילה, ולא כמו ב-Recordset.

עליך לכתוב גם את הלולאה While באמצעות שימוש ברכיבי גישה לקבצים (field accessors) חזקים ויעילים יותר:

VB
```
While reader.Read()
    txtResults.Text = txtResults.Text & reader.GetString(1) & _
                      ControlChars.Tab & reader.GetString(2) & _
                      ControlChars.Tab & ControlChars.Tab & _
                      reader.GetString(0) & ControlChars.CrLf
End While
```

C#
```
while(reader.Read())
{
  txtResults.Text = txtResults.Text + reader.GetString(1) +
                    "\t" + reader.GetString(2) +
                    "\t"  + "\t" +
                    reader.GetString(0) +  "\r\n";
}
```

שינוי נוסף שתוכל לבצע הוא שילוב הצהרת DataReader והפעלת השיטה ExecuteReader בשורה אחת. תוכל להחליף את השורות האלו:

VB
```
Dim reader As OleDbDataReader
reader = cmd.ExecuteReader()
```

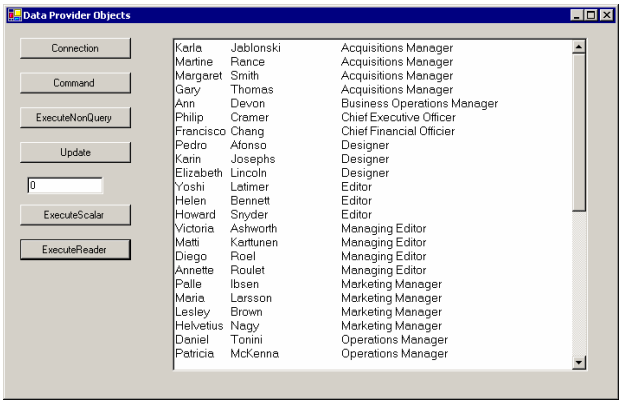
C#
```
OleDbDataReader reader = null;
reader = cmd.ExecuteReader();
```

בשורה אחת, כמו זו:

VB Dim reader As OleDbDataReader = cmd.ExecuteReader()

C# OleDbDataReader reader = cmd.ExecuteReader();

כאשר תפעיל את הפרויקט DataProviderObjects ותלחץ על ExecuteReader, תיבת הטקסט תציג את הנתונים מתוך EmployeeJobs_view, כפי שתוכל לראות בתרשים 2.4.



תרשים 2.4 תוצאות הפעלה מוצלחת של הפקודה ExecuteReader על פי קוד 2.8.

הערה

לאחר השימוש באובייקט DataReader, קרא תמיד לשיטה Close. אם קיימים פרמטרי פלט או ערכי החזרה המוגדרים עבור האובייקט Command שאתה משתמש בו, הם לא יהיו זמינים עד לסגירת DataReader. כמו כן, התחברות DataReader נשארת פתוחה עד לסגירתה. פתחת, סגור.

שימוש ברכיבים Connection ו-Command במהלך תכנון

בתיבת הכלים של Visual Studio, הכרטיסיה Data מכילה רכיבי גישה לנתונים. במהלך התכנון, רכיבים אלה מאפשרים לקבוע ערכי תכונות באמצעות חלון התכונות, במקום ביצוע כל הפעולות בזמן ההפעלה. רכיבים אלה מספקים גם כלים חזותיים להגדרת תכונות מורכבות יותר.

תוכל ליישם את קוד 2.8 באמצעות רכיבים מקבוצה זו, באופן הבא:

1. הוסף טופס בשם **Form2** לפרויקט DataProviderObjects.

2. קבע את ערך התכונה Text ל- **Connection and Command Components**, בחלון Properties של Form2.

3. הגדל את Form2.

4. הוסף תיבת טקסט בשם **Textbox1** לטופס Form2.

5. בחלון Properties, קבע את ערך התכונה Multiline של תיבת הטקסט ל-**True** ואת ערך התכונה ScrollBars ל-**Both**.

6. הגדל את Textbox1, כך שתכסה את רוב שטח הטופס Form2.

7. מהכרטיסיה Data של תיבת הכלים, גרור רכיב OledbConnection לשטח העיצוב של Form2. רכיב זה אינו נראה בזמן ההפעלה, אך הוא יופיע ב-tray של הרכיב מתחת לשטח העיצוב של הטופס.

8. בחלון Properties של הרכיב החדש OledbConnection1 מסוג OledbConnection, קבע את ערך התכונה ConnectionString בדרך זו:

```
provider=SQLOLEDB;server=localhost;uid=sa;database=pubs
```

9. מהכרטיסיה Data של תיבת הכלים, גרור רכיב OledbCommand לשטח העיצוב של Form2. רכיב זה אינו נראה בזמן ההפעלה, אך הוא גם יופיע ב-tray של הרכיב מאחורי שטח העיצוב של הטופס.

10. בחלון Properties של הרכיב החדש OledbCommand1 מסוג OledbCommand, קבע את ערך התכונה Connection כרכיב **SqlConnection1**, ואת ערך התכונה CommandText קבע כך:

```
SELECT * FROM EmployeeJobs_view
```

11. הוסף את קוד 2.9 עבור הרכיב Form2_Load המטפל באירועים (event handler).

טיפ

בפרק 4 נציג כלים אשר מאפשרים להכין את מחרוזת ההתחברות וטקסט הפקודה באופן גרפי, במקום הצורך להזין את מחרוזות הטקסט עצמן.

קוד 2.9-VB קוד ליצירת DataReader ואחזור ערכי שדות באמצעות הרכיבים SqlConnection ו-SqlCommand

```
Private Sub Form2_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    ' Must open the connection before calling ExecuteReader
    OleDbConnection1.Open()
    Dim reader = OleDbCommand1.ExecuteReader()
```

```
    TextBox1.Clear()
    While reader.Read()
        TextBox1.Text = TextBox1.Text & reader("fname") & _
                ControlChars.Tab & reader("lname") & _
                ControlChars.Tab & ControlChars.Tab & _
                reader("job_desc") & ControlChars.CrLf
    End While

    ' Deselect all lines in textbox
    TextBox1.SelectionLength = 0

    reader.Close()
    OleDbConnection1.Close()
End Sub
```

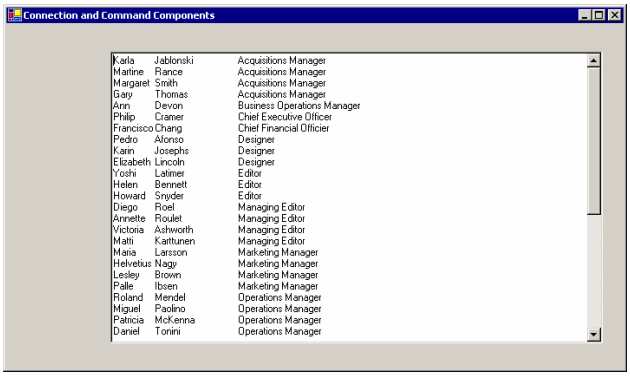
קוד #C-2.9 קוד ליצירת DataReader ואחזור ערכי שדות באמצעות הרכיבים SqlConnection ו-SqlCommand

```
private void Form2_Load(object sender, System.EventArgs e)
  {
    //Must open the connection before calling ExecuteReader
    oleDbConnection1.Open();
    System.Data.OleDb.OleDbDataReader reader = oleDbCommand1.ExecuteReader();
    textBox1.Clear();
    int counter = 0;

        while(reader.Read() && (++counter < 400))
        {
          textBox1.Text += reader["ShipAddress"];
          textBox1.Text += "\t" + reader["ShipCity"];
          textBox1.Text += "\t\t\t" + reader["ShipCountry"]
                + reader["CustomerID"] +"\n\r";
        }

        // Deselect all lines in textbox
        textBox1.SelectionLength = 0;
        reader.Close();

        oleDbConnection1.Close();
  }
```

12. לחץ לחיצה ימנית על הפרויקט DataProviderObjects ב- Solution Explorer, ומתפריט הקיצור בחר **Properties**.

13. בחר בפריט **General** שבתיקיה Common Properties, ולאחר מכן קבע את ערך התכונה Startup object ל-**Form2**.

כאשר תפעיל את הפרויקט DataProviderObjects, תיבת הטקסט ב-Form2 תציג את כל הנתונים המתקבלים מ-EmployeeJobs_view, כפי שמוצג בתרשים 2.5.



תרשים 2.5 תוצאות הצגת Form2, אשר מנצל את הרכיבים OledbConnection ו-OledbCommand.

אובייקטים נוספים של ספקי נתונים

כבר הצגנו מספר מחלקות ספקי נתונים, כגון האובייקטים Parameter ו-Parameters, בנוסף לארבעת האובייקטים העיקריים המופיעים בטבלה 2.1. בפרק 4 נחקור לעומק את DataAdapter, לאחר הצגת DataSet והאובייקטים הקשורים אליו בפרק 3.

לפני שנסיים פרק זה, נבחן את אובייקט ספק הנתונים האחרון – Transaction. השימוש בתנועות (transactions) מיועד להבטיח בקרת סיום של פעולות בעלות שלבים אחדים, ולאשר את כל המכלול או לבטל את כל שלבי הביצוע, בבחינת "הכל או לא כלום". כלומר, או שכל שלבי הפעולה יבוצעו בהצלחה, או שאף אחד מהם לא יבוצע. הדוגמה הקלאסית לתנועה היא העברת כסף בנקאית, המורכבת משני שלבים: הפחתת סכום מחשבון אחד והוספת סכום זה לחשבון אחר. לא נרצה לגרום למצב שבו רק הפעולה הראשונה בוצעה מבין השתיים, כמו במקרה שהחשבון האחר לא קיים או שהוא חסום!

ספקי נתונים של ADO.NET מיישמים את האובייקט Transaction, אשר מכיל שיטות עיקריות הדרושות לשימוש בעסקאות. השיטה Commit גורמת לאישור העסקה הנוכחית בשלמותה, והשיטה Rollback מבטלת את העסקה הנוכחית בשלמותה. התנועה מופעלת, והאובייקט Transaction נוצר על ידי קריאה לשיטה BeginTransaction באובייקט Connection הפתוח. את האובייקט Transaction נציג ב"תסריט עסקי 2.1" שלהלן.

תסריט עסקי 2.1: כתיבת שגרה לגניזת הזמנות ישנות בארכיון, לפי שנים

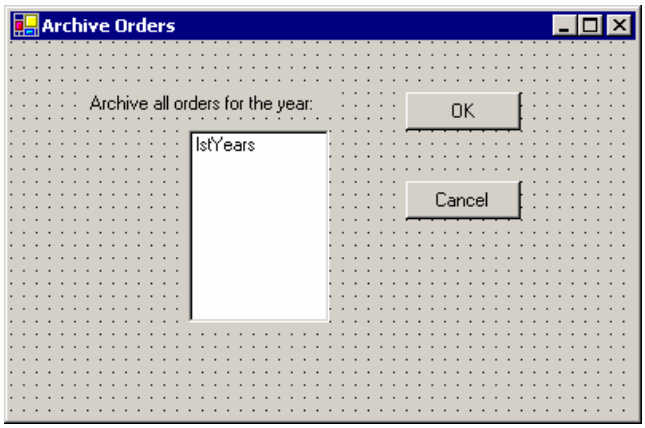
כאשר מערכת מסד נתונים נמצאת בשימוש במשך זמן רב (המונח "זמן רב" יחסי כמובן), ניתן ואף צריך לגנוז נתונים מסוימים בארכיון. גניזה כזו צריכה להתבצע בנוסף לפעולות הגיבוי המתוזמנות הרגילות שבכל מערכת מסחרית. הנתונים הגנוזים אינם דרושים בעבודה שוטפת (במצב מקוון), אך ייתכן שיהיה צורך לגשת אליהם מדי פעם. הסרת נתונים אלה מטבלאות מקוונות עיקריות עשויה לשפר את ביצועי הגישה אליהן, מכיון שהמערכת תצטרך לסרוק או לסנן מספר קטן יותר של רשומות. יש לזכור שטבלה גנוזה נשמרת לרוב במבנה זהה לטבלה המקוונת, ולכן ניתן לגשת אליה באותו האופן. בתסריט המוצג כאן, ננחה אותך בתהליך פיתוח טופס אשר מבצע גניזה פשוטה של נתונים מהטבלה tblOrder שבמסד הנתונים Novelty. הטופס יאפשר לבחור את השנה אשר ההזמנות שבוצעו בה ייגנזו. לאחר בחירת השנה הרצויה, תוכל לבצע את הפעולות שלהלן:

1. טבלה חדשה tblOrderXXXX תיווצר במסד הנתונים Novelty. התווים "XXXX" יוחלפו בשנת הרישום של ההזמנות בטבלת הגניזה.
2. כל הרשומות המתאימות יועתקו מהטבלה tblOrder לטבלה tblOrderXXXX.
3. כל הרשומות שהועתקו יימחקו מהטבלה tblOrder.

חשוב לזכור כי צריך לוודא שאם אחד מהשלבים האלה ייכשל, הפעולה כולה תבוטל. אינך רוצה ליצור טבלה חדשה אם לא ניתן להוסיף לה נתונים. אינך רוצה להוסיף רשומות לטבלת הגניזה אם לא ניתן למחוק אותן מהטבלה הראשית. בנוסף, אינך רוצה למחוק את הרשומות מהטבלה tblOrder אם לא תוכל להעתיק אותן לטבלת הגניזה. תוכל להשתמש באובייקט Transaction ולהחזיר את מסד הנתונים למצבו הקודם במקרה של כישלון בשלב כלשהו. צור טופס אשר יבצע זאת. בצורה זו תוכל לסקור ולתרגל את הנושאים שהצגנו בפרק זה.

1. הפעל את Visual Studio.NET.
2. צור פרויקט חדש מסוג Visual Basic Windows Application.
3. קבע לפרויקט את השם **BusinessCase4**.
4. ציין נתיב לשמירת קבצי הפרויקט.
5. הגדל את Form1.
6. בחלון Properties של Form1, קבע את ערך התכונה Name ל-**frmArchive** ואת ערך התכונה Text ל- **Archive Orders**.
7. הוסף תיבת רשימה **lstYears**, תווית **Label1**, לחצן **btnOK** ולחצן **btnCancel**.
8. קבע את ערך התכונה Text של התווית ל- **Archive all orders for the year**. קבע את ערך התכונה Text של הלחצנים ל-**OK** או **Cancel**, בהתאמה.

9. סדר את הפקדים כפי שמוצג בתרשים 2.6.



תרשים 2.6 סידור הפקדים ב-frmArchive.

בתחילת הקובץ הוסף את שורת הקוד הראשונה לייבוא מרחב השמות SqlClient:

```
Imports System.Data.SqlClient
```

בגוף הגדרת המחלקה עבור frmArchive, הוסף את קוד 2.10.

קוד 2.10-VB קוד לגניזת נתונים בטבלה חדשה

```
Private Sub frmArchive_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

    lstYears.Items.Add("1995")
    lstYears.Items.Add("1996")
    lstYears.Items.Add("1997")
    lstYears.Items.Add("1998")
    lstYears.Items.Add("1999")
    lstYears.Items.Add("2000")
    lstYears.Items.Add("2001")
    lstYears.Items.Add("2002")

    ' Set Default
    lstYears.SelectedIndex = 0
End Sub

Private Sub btnCancel_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnCancel.Click
    Me.Close()
End Sub
```

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnOK.Click

    Dim sql As String
    Dim result As Integer
    Dim records As Integer
    Dim SelectedYear As String

    ' Create an instance of an Connection and command objects
    Dim cnn As SqlConnection = New SqlConnection( _
        "server=localhost;uid=sa;database=novelty")
    Dim cmd As New SqlCommand()

    Dim trans As SqlTransaction

    ' First get year
    SelectedYear = lstYears.SelectedItem.ToString

    ' Put the code inside a Try-Catch block to trap failures
    Try

        ' Open the Connection and begin transaction
        cnn.Open()
        trans = cnn.BeginTransaction

        ' Enlist the command in this transaction
        cmd.Connection = cnn
        cmd.Transaction = trans

        ' SQL to insert appropriate records into archive table
        sql = "SELECT * INTO tblOrder" & SelectedYear & _
            " FROM tblOrder WHERE year (OrderDate) = " & SelectedYear

        ' This command is in the transaction
        cmd.CommandText = sql

        result = cmd.ExecuteNonQuery()

        ' Show results of inserting records into archive
        If result > 0 Then
            records = result
            MessageBox.Show(records & _
                " records inserted successfully into tblOrder" & _
                SelectedYear)
```

```
    Else
        MessageBox.Show( _
            "No records inserted into tblOrder" & _
            SelectedYear)
        ' Since no records, don't keep created table
        ' cancel / rollback transaction
        trans.Rollback()
    End If

    If records > 0 Then
        ' SQL to delete appropriate records from current table
        sql = "delete FROM tblOrder WHERE year(OrderDate) = " _
                & SelectedYear

        ' This command is also in the same transaction
        cmd.CommandText = sql

        result = cmd.ExecuteNonQuery()

        ' Show results of deleting records
        If result = records Then
            MessageBox.Show(records & " records deleted successfully")

            ' If we got to here, then everything succeeded
            trans.Commit()
        Else
            MessageBox.Show("No records deleted!")
        End If
    Else
        ' nothing to do
    End If


Catch ex As Exception
    ' If we got to here, then something failed and
    ' we cancel (rollback) the entire transaction
    Try
        ' Display error message
        MessageBox.Show(ex.Message & _
                ControlChars.CrLf & ControlChars.CrLf & _
                "Transaction Failed !")

        trans.Rollback()
```

```
        Catch ex2 As Exception
        End Try

    Finally
        cnn.Close()
    End Try
End Sub
```

קוד #C-2.10 קוד לגניזת נתונים בטבלה חדשה

```
public Form1()
{
  InitializeComponent();
  lstYears.Items.Add("1995");
  lstYears.Items.Add("1996");
  lstYears.Items.Add("1997");
  lstYears.Items.Add("1998");
  lstYears.Items.Add("1999");
  lstYears.Items.Add("2000");
  lstYears.Items.Add("2001");
  lstYears.Items.Add("2002");

  // Set Default
  lstYears.SelectedIndex = 0;
}

private void btnCancel_Click(object sender, System.EventArgs e)
{
  Close();
}

private void btnOK_Click(object sender, System.EventArgs e)
{
    String sql;
    int result;
    int records = 0;
    String SelectedYear;

    // Create an instance of an Connection and command objects
    SqlConnection cnn   = new  SqlConnection(
                              "server=KALTERM;uid=sa;database=novelty");
    SqlCommand cmd = new SqlCommand();

    SqlTransaction trans = null;
```

```
// First get year
SelectedYear = lstYears.SelectedItem.ToString();

// Put the code inside a Try-Cache block to trap failures
try
{
   // Open the Connection and begin transaction
   cnn.Open();
   trans = cnn.BeginTransaction();

   // Enlist the command in this transaction
   cmd.Connection = cnn;
   cmd.Transaction = trans;

   // SQL to insert appropriate records into archive table
   sql = "SELECT * INTO tblOrder" + SelectedYear +
         " FROM tblOrder WHERE year (OrderDate) = " + SelectedYear;

   // This command is in the transaction
   cmd.CommandText = sql;

   result = cmd.ExecuteNonQuery();

   // Show results of inserting records into archive
   if (result > 0)
   {
     records = result;
     MessageBox.Show(records +
                          " records inserted successfully into tblOrder" +
                          SelectedYear);
   }
   else
   {
      MessageBox.Show("No records inserted into tblOrder" +
                           SelectedYear);

      // Since no records, don't keep created
      // table cancel / rollback transaction
      trans.Rollback();
   }

   if (records > 0)
   {
      // SQL to delete appropriate records from current table
      sql = "delete FROM tblOrder WHERE year(OrderDate) = " +
            SelectedYear;
```

```
            // This command is also in the same transaction
            cmd.CommandText = sql;

            result = cmd.ExecuteNonQuery();

            // Show results of deleting records
            if (result == records)
            {
              MessageBox.Show(records + " records deleted successfully");

              // If we got to here, then everything succeeded
              trans.Commit();
            }
            else
            {
              MessageBox.Show("No records deleted!");
            }
          }
        }

        catch (Exception ex)
        {
          try
          {
            // Display error message
              MessageBox.Show(ex.Message + "\n" + "\n"
                                + "Transaction Failed !");
              trans.Rollback();
          }

          catch (Exception e1)
         {
         }

        }

        finally
        {
          cnn.Close();
        }
      }
    }
}
```

השגרה frmArchive_Load מאתחלת את lstYears ומכניסה את ערכי השנים לבחירה, וכן בוחרת כברירת מחדל בשנה הראשונה (המוקדמת ביותר). כמובן שתוכל לשפר שגרה זו באופן שתבצע שאילתה על tblOrder, כדי לאחזר את רשימת השנים שיש עבורן הזמנות בטבלה זו. לצורך ההדגמה, השיטה הפשוטה יותר תספיק.

מטפל האירועים Click (כלומר, event handler) של הלחצן Cancel סוגר את הטופס. במקרה זה, הפעולה גם תסיים את התוכנית. כל הפעולה מתרחשת במטפל האירועים click של btnOK. לאחר ההצהרה על משתנים, עליך להשיג מ-lstYears את השנה שנבחרה ולשמור אותה לשימוש בהמשך. כדי להבטיח שתוכל לבטל את העסקה במקרה של שגיאה, עליך לעטוף את כל הקוד הפעיל בבלוק Try-Catch-Finally.

התנועות מוגדרות ברמת ההתחברות, ולכן עליך לפתוח תחילה את ההתחברות ורק לאחר מכן ליצור את האובייקט Transaction על ידי קריאה ל-BeginTransaction בהתחברות הפתוחה. האובייקטים Connection ו-Transaction מוקצים לאובייקט Command אשר ישמש להפעלת פקודות מסד הנתונים.

שני השלבים הראשונים ביצירת טבלת הגניזה והעתקת הרשומות המצוינות אל הטבלה החדשה מבוצעים בהצהרת SQL אחת, באמצעות שימוש בהצהרה SELECT INTO. זוהי הצהרת SELECT רגילה, בתוספת המשפט INTO tablename. הטבלה הרשומה במשפט נוסף זה נוצרת באופן אוטומטי; הפקודה מייצרת exception אם הטבלה כבר קיימת. השנה שנבחרה מתווספת ל-tblOrder ליצירת שם טבלת הגניזה החדשה שתיווצר.

הערה

ההצהרה SELECT INTO אינה יוצרת את האינדקסים הקיימים בטבלה המקורית. כדי לשפר את ביצועי השאילתות המבוצעות על טבלה זו, סביר שתרצה ליצור אינדקסים בשדה אחד או יותר.

השיטה ExcecuteNonQuery נקראת כדי להפעיל את הפקודת SQL; היא מחזירה את מספר הרשומות שהושפעו מהפעולה. אם מספר זה גדול מאפס, תדע כי פעולת היצירה והמילוי של הטבלה החדשה הסתיימה בהצלחה וכעת ניתן להמשיך בביצוע השגרה. אם המספר הוא אפס, לא היה ניתן ליצור את הטבלה או שלא נמצאו רשומות להעתקה. בכל מקרה, התנועה תבוטל באופן שגם אם הטבלה תיווצר בהצלחה, מסד הנתונים לא יכיל טבלאות ריקות ומיותרות.

אם נוספה לפחות רשומה אחת לטבלת הגניזה, התהליך יימשך. השלב הבא הוא מחיקת הרשומות המתאימות מהטבלה tblOrder המקורית. תוכל לעשות זאת על ידי הפקודה DELETE, שבה השנה שנבחרה מתווספת למשפט WHERE. אם שיטה זו תבוצע בהצלחה – כלומר, מספר הרשומות שהושפעו באמצעותה יהיה שווה למספר הרשומות שהתווספו לטבלת הגניזה – הרי שהפעולה בוצעה בהצלחה והתנועה תתקיים. אם לא, משהו השתבש (כגון שינויים ברשומות רלוונטיות, הרשאות לא מתאימות או נפילת השרת), העסקה כולה תבוטל ומסד הנתונים יחזור למצבו הקודם. החזרת המצב לקדמותו מבטיחה כי אם מחיקת הרשומות המתאימות מהטבלה tblOrder נכשלה, טבלת הגניזה tblOrderXXXX תימחק.

עד לנקודה זו, השגרה טיפלה במצבים אשר מתרחשים כחלק מההפעלה הרציפה שלה. עם זאת, יש לטפל גם בחריגים (exceptions) אשר מתרחשים בזמן ההפעלה. לדוגמה, אם תנסה ליצור טבלה אשר קיימת כבר במסד הנתונים, ייווצר מצב חריג. חריגים כאלה נלכדים ומטופלים בבלוק Catch. הטקסט המתייחס למצב החריג מוצג, והתנועה כולה מבוטלת ומוחזרת למצב הקודם.

הערה

הבלוק Try-Catch השני דרוש כדי לטפל במצבים שלא ניתן ליצור בהם את טבלת הגניזה (לדוגמה, מכיון שהטבלה כבר קיימת). הסיבה היא, שלמרות שהתנועה החלה להתבצע, לא בוצעו פקודות שינוי נתונים, ולכן לא נכתב ליומן דבר שניתן להחזירו למצבו הקודם.

בבלוק Finally, נסגרת ההתחברות שנעשה בה שימוש. יש לבצע זאת בין אם התרחשה או לא התרחשה שגיאה.

בצע ניסויים בפרויקט זה על ידי יצירת ארכיונים לשנים שונות. תוכל לבדוק את הפעולות על ידי בחינת טבלת הגניזה החדשה וגם את הטבלה tblOrder המקורית (לפני ואחרי פעולת הגניזה). זכור שתוכל תמיד לאפס את תוכן הטבלה tblOrder על ידי הפעלת תסריטים ליצירה ו/או מילוי של הטבלה tblOrder.

סיכום

בפרק זה הצגנו את ADO.NET באופן כללי והתמקדנו באובייקטי ספק הנתונים NET.. ספקי נתונים (data provider) הם ממשקי ADO.NET למחסני נתונים פיסיים והם מספקים מודל תכנות במצב מחובר. סקרנו את התכונות והשיטות של האובייקטים Connection, Command, Parameter, DataReader ו-Transaction, הבאנו דוגמאות הכוללות את ספקי הנתונים הסטנדרטיים SqlClient, Oledb ו-Odbc. בפרק 3 נראה כיצד מודל התכנות המנותק, המבוסס על האובייקטים DataSet ו-DataAdapter, בנוי על האובייקטים והדוגמאות הללו.

שאלות ותשובות

ש: מתוך פרק זה אני מבין כי ADO.NET מיועד בעיקר לשימוש מנותק, ואין תמיכה בסמני שרת (server-side cursor) או בנעילה פסימיסטית (Pessimistic Locking). מה עלי לעשות אם היישום הקיים שלי משתמש באחת מהתכונות הללו, או אם מפרט הפרויקט החדש דורש סמני שרת או נעילה פסימיסטית? האם עלי להישאר עם Visual Basic 6.0?

ת: נתח תחילה בזהירות את היישום שלך וודא שאינך נעול בתפיסת חשיבה מיושנת. כלומר, שהסיבה לדרישה אינה נעוצה בעובדה שאתה רגיל להשתמש בנעילה פסימיסטית או בסמני שרת. אם, לאחר שבחנת את האפשרויות, אתה משוכנע כי תכונות אלו נחוצות לך, או שאתה זקוק למשהו נוסף ש-ADO.NET אינו תומך בו, אל תתייאש. תוכל להשתמש עדיין ב-VB.NET ליצירת יישומים אלה. המערכת NET Framework. מכילה תמיכה מקיפה בפעולה הדדית עם COM, אשר מאפשרת ליישומי NET. שלך לנצל אובייקטי COM, ולאובייקטי COM – לנצל קוד מנוהל (NET.). במילים אחרות, תוכל להמשיך להשתמש ב- ADO 2.X ואף בכל אובייקטי COM נוספים שלא קיימים עבורם עדיין תחליפים ב-NET.. המחיר שתשלם על כך יהיה כמובן בירידת ביצועים כאשר תעבור בין סביבת COM ל-NET., ולהיפך. האם מחיר הביצועים כבד מדי? בדומה לרוב השאלות הקשורות בביצועים, התשובה היא שעליך לבדוק בעצמך את היישום המסוים שלך.

ש: תכנות האובייקטים, השיטות והתכונות אשר מתואר בפרק זה אינו נראה שונה למדי מהתכנות ב- ADO 2.X. מדוע עלי לטרוח לעבור ל-ADO.NET?

ת: במובן מסוים, אתה צודק. ביצוע פעולות בסיסיות לשם התחברות למסד הנתונים אינן שונות במידה ניכרת מהשימוש ב- ADO 2.X. עם זאת, בנוסף לשיפורים השוליים שהזכרנו, עליך לקחת בחשבון מספר נקודות חשובות.

1. Visual Basic.NET ופלטפורמת NET. מהוות עולם חדש לגמרי, ו-ADO.NET מהווה את הדרך לגשת לנתונים בעולם זה.
2. תוכל להמשיך להשתמש ברכיבי COM, כגון ADO 2.X, במהלך פיתוח יישומי NET., אך המחיר יתבטא בתקורת ביצועים בעת גישה לרכיבי COM אלה ובצורך להתקין ולבצע רישום שלהם בצורה מתאימה.
3. אובייקטי ספקי הנתונים המתוארים בספר זה הם חלק אחד בלבד מהמכלול, אשר מתייחס לקריאה ולכתיבה הפיסיות ממסד הנתונים במצב מחובר. המצב המנותק של פעולה המבוססת על אובייקט DataSet הוא המקום שבו באים לידי ביטוי העוצמה האמיתית והיתרונות של ADO.NET. בפרק זה עסקנו תחילה באובייקטים הבסיסיים, מכיון שהם אבני הבניין של הנושאים הבאים.

פרק 3

ADO.NET – DataSet

מה בפרק זה?

- יישומים ורכיבים של DataSet
- מילוי נתונים ב-DataSet ושינוי שלהם
- שימוש ברכיב DataSet

האובייקט DataSet הוא האלמנט המרכזי והמהפכני ביותר בתפיסת ADO.NET בכל הקשור לגישה לנתונים. DataSet הוא מטמון השוכן בזיכרון של נתונים ממקור נתונים אחד או יותר. תוכל לראות באלמנט זה מסד נתונים מלא השוכן בזיכרון. המאפיין הבולט ביותר הוא בכך שהשימוש בו נעשה בצורה מנותקת לחלוטין. אובייקטי NET Data Provider. המנוהלים, אשר תוארו בפרק הקודם, מספקים פונקציונליות כאשר הם מחוברים פיסית למסד נתונים או למקור נתונים אחר. בניגוד לכך, DataSet והאובייקטים הקשורים אליו (כגון האובייקטים DataTable, DataRow, DataColumn ו-DataRelation) מספקים פונקציונליות עשירה כאשר הם מנותקים פיסית ממקור הנתונים עצמו.

מאפיין נוסף של DataSet הוא העובדה שלאחר שטוענים אותו בנתונים, הוא אינו "יודע" מהו מקורם. האלמנט יכיל נתונים ויאפשר לשנות אותם בצורה יחסית, תוך כדי התייחסות לאילוצים. לפיכך, DataSet והאובייקטים שלו הינם כלליים ואינם ייחודיים לספק נתונים מסוים. כלומר, יש אובייקט DataSet בלבד – ולא SqlDataSet, OledbDataSet וכדומה.

אם DataSet אינו יודע מהו מקור הנתונים שלו וכיצד הם נטענו אליו? יתרה מכך, כיצד השינויים שמבוצעים בו גורמים לעדכון גם במקור הנתונים? התשובות לשאלות אלו מגיעות לרוב מהאובייקט DataAdapter, אשר מהווה גשר בין DataSet לבין מקור הנתונים הפיסי. ל-DataAdapter נשלחות פקודות לקריאת הנתונים ממקור הנתונים וגם פקודות לעדכון, מחיקה והוספה של נתונים למקור זה. את DataAdapter נתאר באופן מפורט בפרק הבא.

יישומים ורכיבים של DataSet

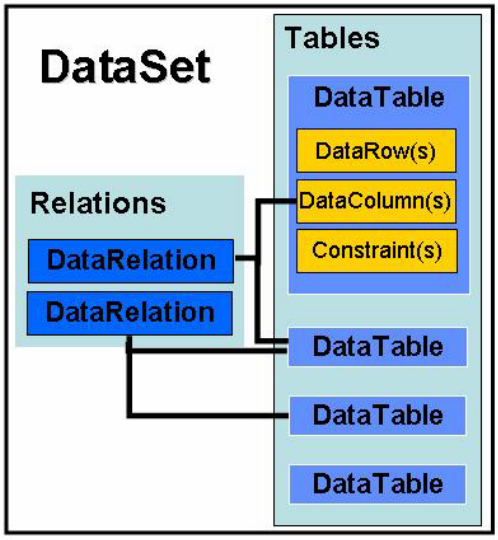
DataSet הוא אובייקט המפתח ב-ADO.NET. זוהי מכולה (container) אוניברסלית לנתונים, אשר אינה קשורה למקור הנתונים. DataSet והאובייקטים הקשורים אליו מציעים תצוגה יחסית של נתונים, למרות שהוא מסוגל לטעון או לשמור את הנתונים ו/או את נתוני הסכמה שלו כ-XML. DataSet מספק מודל נתונים השוכן במפורש בזיכרון אשר מנותק תמיד לגמרי ממקור הנתונים, וניתן להעביר אותו בקלות בין מרחבי זיכרון ומחשבים.

יישומים ושימושים נפוצים ב-DataSet:

- **נתוני יישום** – DataSet הוא אובייקט פשוט וגמיש לאחסון נתוני יישומים מקומיים. הגישה לנתונים נעשית בצורה קלה הדומה לגישה לנתונים במערך. עם זאת, DataSet מספק גם תכונות מתקדמות כגון מיון וסינון.
- **גישה מרוחקת לנתונים** – DataSet משתמש ב-XML באופן אוטומטי להעברת נתונים ממחשב אחד למחשב אחר. יכולת זו מקלה במידה ניכרת על הפיתוח של יישומי Web Services, SOAP או גישה מרוחקת ברמה נמוכה (low-level remoting).
- **הטמנת נתונים** – DataSet יכול להטמין נתונים במהלך ריצת יישומי ASP.NET או סוגים נוספים של יישומים מבוזרים, ומונע תנועה מיותרת ברשת.

- **שמירת נתונים** – DataSet מספק שיטות לשמירת נתונים ונתוני סכמה במבנה XML סטנדרטי.
- **התערבות משתמש** – DataSet תומך בצורה יעילה בהתערבות משתמש בממשקים גרפיים שונים על ידי שילוב יכולות המיון, הסינון והגלילה שלו; הוא גם יכול לכרוך תצוגות נתונים שונות בטפסי Windows וטפסי Web.

הנתונים ב-DataSet (אוסף טבלאות וקשרים) מאורגנים ב-DataTable (טבלאות) אחת או יותר. כל טבלה אינה יודעת מהו מקור הנתונים. כלומר, DataSet ו-DataTables (טבלה) מנותקים תמיד ממקור הנתונים. DataSet מכיל טבלאות נתונים שונות ומאפשר שינוי, העברה או קישור נתונים לפקדי ממשק משתמש. תרשים 3.1 מציג את DataSet ואת אובייקטי המשנה שלו.



תרשים 3.1 DataSet ואובייקטי משנה.

ניתן לתאר את אובייקטי המשנה האלה בצורה הבאה.

- **DataTable** – כל טבלה מכילה אוסף של אובייקטים מסוג DataRow (שורה), DataColumn (עמודה) ו-Constraint (אילוץ), יחד עם אוספים של אובייקטי DataRelation (קישורים) אל טבלאות אב ובן נוספים. תצוגת הנתונים של טבלה דומה לזו שקיימת בערכת רשומות של ADO 2.X.
- **DataColumn** – האובייקט DataColumn הוא היחידה הבסיסית המשמשת לאחזור או להגדרה של הסכמה עבור טבלה. אובייקט זה מכיל את המידע הייחודי עבור כל עמודה (שדה) בטבלה, כולל שם, סוג נתונים ותכונות נוספות כגון Unique, ReadOnly, AllowDBNull ו-AutoIncrement. באובייקט זה קיימת גם

התכונה Expression, אשר מאפשרת לחשב ערך שדה או ליצור שדה צבירה (aggregate column).

- **DataRow** – האובייקט DataRow מייצג רשומה (שורה) יחידה בטבלה והוא האובייקט שעליך להשתמש בו כדי להוסיף, לאחזר ו/או לשנות נתונים בטבלה. תוכל לנווט באוסף Rows של טבלה על פי רצף, או בגישה ישירה לרשומות רצויות.
- **DataRelation** – האובייקט DataRelation מגדיר את הקשר בין שתי טבלאות שכלולות באוסף DataSet. האובייקט מייצג קשר אב-בן (parent-child key) קלאסי או קישור של מפתח ראשי למפתח זר (primary key-foreign key) בין רשומות בשתי הטבלאות. תוכל לנווט בין יחסי הטבלאות באמצעות האוספים ChildRelations ו-ParentRelations (של אובייקטי DataRelation) השייכים לאובייקט DataTable.
- **Constraint** – האובייקט Constraint מגדיר כלל אשר תפקידו להבטיח את שלמות הנתונים בטבלה. אובייקט זה מכיל UniqueConstraint אשר מבטיח כי כל הערכים שבטבלה הם ייחודיים, ואת ForeignKeyConstraint אשר קובע מהי הפעולה שיש לבצע ברשומות שבטבלה המיוחסת. האילוץ המוגדר יכול להכיל DataColumn אחד או יותר. לכל טבלה יש את התכונה Constraints, שהיא אוסף אילוצים עבור טבלה זו.

מילוי נתונים ב-DataSet ושינוי שלהם

תוכל למלא נתונים בטבלה שב-DataSet, באחת משלוש הדרכים הבאות:

1. בתכנות, על ידי הגדרה ישירה של metadata והוספת נתונים.
2. ביצוע שאילתה במקור נתונים באמצעות DataAdapter (מתאם נתונים).
3. טעינת מסמך XML.

בפרק זה נציג את השיטה הראשונה, שבה הגדרת הסכמה וטעינת הנתונים מתבצעות בצורה ישירה. בפרק הבא נדון בשיטה השנייה אשר בה נעשה שימוש במתאם נתונים. בסעיף זה נתחיל בהסבר על מילוי DataSet בשיטות תכנות, ונסביר שיטות גישה בסיסיות ופונקציוליות בסיסית של DataSet ו-DataTable.

עם זאת, שים לב לכך שלאחר מילוי DataSet בנתונים, אין חשיבות לשיטה שבה משתמשים. כל הפעולות המבוצעות לאחר מכן, אשר קשורות לנתונים ולטיפול בהם, נעשות בדרך דומה.

הגדרת סכמות של DataTable

כעת נדריך אותך פעם נוספת ביצירת טופס פשוט אשר מדגים את התפיסה המוצגת. התחל בביצוע הפעולות שלהלן:

1. הפעל את Visual Studio.NET.
2. צור פרויקט **Visual Basic Windows Application** חדש.
3. קבע לפרויקט את השם **DataSetCode**.
4. ציין נתיב לשמירת קבצי הפרויקט.
5. הגדל את Form1.
6. בחלון Properties של Form1, קבע את ערך התכונה Name ל-**frmDataSets** ואת ערך התכונה Text ל-**DataSets**.
7. בפינה השמאלית העליונה של הטופס, הוסף לחצן בכרטיסיה Windows Forms שבתיבת הכלים.
8. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**btnCreateDS** ואת ערך התכונה Text ל- **Create DataSet**.
9. בכרטיסיה Windows Forms של תיבת הכלים, הוסף תיבת רשימה (listbox) ל-frmDataSets ומקם אותה בצד ימין של הטופס.
10. בחלון Properties, קבע את ערך התכונה Name של תיבת הרשימה ל-**lstOutput**.
11. הגדל את תיבת הרשימה באופן שתכסה כ-80% אחוז משטח הטופס.

להלן קטע הקוד הראשון שעליך להוסיף לקובץ:

```
Imports System
Imports System.Data
```

הערה

אם פתחת פרויקט #C בדרך זו, ההוספה כבר מתבצעת באופן אוטומטי בטופס הראשון שהמערכת מכניסה כברירת מחדל. אבל, בכל טופס אחר שתפתח, אינך מקבל זאת באופן אוטומטי, ועליך לבצע זאת בעצמך.

לאחר מכן הוסף את הקוד הבא בגוף הגדרת המחלקה של frmDataSets:

VB: Private dsEmployeeInfo As DataSet

C#: private System.Data.DataSet dsEmployeeInfo;

לחץ על הלחצן שיצרת ותקבל את פונקציית האירוע הבאה, אשר מטפלת בלחיצה על עכבר. הוסף את הקטע המסומן:

VB:

```
Private Sub btnCreateDS_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles brnCreateDS.Click
    CreateDataSet()
    AddData()
    DisplayDataSet()
End Sub
```

C#:

```
private void btnCreateDS_Click(object sender, System.EventArgs e)
  {
    CreateDataSet()
    AddData()
    DisplayDataSet()
  }
```

הערה: בשלב זה – התוכנית לא תעבור הידור.

השגרה btnCreateDS_Click (btnCreateDS_Click) תקרא ותפעיל שלוש שגרות, פעם אחת עבור כל אחד משלושת השלבים ביישום. המשתנה dsEmployeeInfo הוא DataSet שהגישה אליו מתבצעת באמצעות כל אחת מהשגרות הנקראות על ידי btnCreateDS_Click.

הערה

לרוב תשתמש ב-DataSet אשר יכיל את הטבלאות שתשתמש בהן, אך לחילופין תוכל להגדיר ולהשתמש באובייקטי DataTable (ובאובייקטי המשנה שלהם) בעצמם. תוכל לעשות זאת בשימושים פשוטים בטבלה. לדוגמה, כאשר עליך לתת למשתמש אפשרות לבחור צבע באמצעות ממשק WEB, תוכל להכניס את רשימת הצבעים אל הטבלה הרצויה ולהשתמש לאחר מכן בטבלה בתוך הפקד המתאים.

הדבר הראשון שעליך לעשות הוא להגדיר את הסכמה, או המבנה, של כל טבלה שברצונך להשתמש בה. פעולה זו כוללת הגדרה של אובייקט DataColumn והגדרת מאפייני כל שדה בטבלה. קוד 3.1 מציג את הקוד הדרוש עבור CreateDataSet.

קוד 3.1-VB קוד ליצירת הסכמות DataSet ו-DataTable

```
Private Sub CreateDataSet()
        'Create an "EmployeeInfo" DataSet
        dsEmployeeInfo = New DataSet()

        'Create an "Employees" Table
        Dim dtEmployees As DataTable = New DataTable("Employees")
        dtEmployees.CaseSensitive = False
```

```
    dtEmployees.Columns.Add("FirstName", Type.GetType("System.String"))
    dtEmployees.Columns.Add("LastName", Type.GetType("System.String"))
    dtEmployees.Columns.Add("DepartmentID", Type.GetType("System.Int32"))

    'Add Employees table to EmployeeInfo DataSet
    dsEmployeeInfo.Tables.Add(dtEmployees)

    ' Create an "Departments" Table
    ' We'll do this one with different function overrides
    ' This approach is more lengthy for standard columns,
    ' but allows setting other column properties (e.g. ReadOnly & Unique)
    ' before adding the DataColumn to the Columns collection
    Dim dtDepartments As DataTable
    dtDepartments = New DataTable()
    dtDepartments.TableName = "Departments"
    dtDepartments.MinimumCapacity = 5
    dtDepartments.CaseSensitive = False

    Dim NewColumn As New DataColumn()

    With NewColumn
        .ColumnName = "ID"
        .DataType = Type.GetType("System.Int32")
        .ReadOnly = True
        .Unique = True
        .AutoIncrement = True
    End With
    dtDepartments.Columns.Add(NewColumn)

    NewColumn = New DataColumn()
    With NewColumn
        .ColumnName = "DepartmentName"
        .DataType = Type.GetType("System.String")
        .Unique = True
        .AllowDBNull = False
    End With
    dtDepartments.Columns.Add(NewColumn)

    'Add Departments table to EmployeeInfo DataSet
    dsEmployeeInfo.Tables.Add(dtDepartments)
End Sub
```

```
private void CreateDataSet()
{
  //Create an "EmployeeInfo" DataSet
  dsEmployeeInfo = new DataSet();

  //Create an "Employees" Table
  DataTableDim dtEmployees = new DataTable("Employees");
  dtEmployees.CaseSensitive = false;

  dtEmployees.Columns.Add("FirstName", Type.GetType("System.String"));
  dtEmployees.Columns.Add("LastName", Type.GetType("System.String"));
  dtEmployees.Columns.Add("DepartmentID",
            Type.GetType("System.Int32"));

  // Add Employees table to EmployeeInfo DataSet
 dsEmployeeInfo.Tables.Add(dtEmployees);

  // Create an "Departments" Table
  // We'll do this one with different function overrides
  // This approach is more lengthy for standard columns,
  // but allows setting other column properties (e.g. ReadOnly &
            Unique)
  // before adding the DataColumn to the Columns collection
  DataTable dtDepartments;
  dtDepartments = new DataTable();
  dtDepartments.TableName = "Departments";
  dtDepartments.MinimumCapacity = 5;
  dtDepartments.CaseSensitive = false;

  DataColumn NewColumn = new DataColumn();

  NewColumn.ColumnName = "ID";
  NewColumn.DataType = Type.GetType("System.Int32");
  NewColumn.ReadOnly = true;
  NewColumn.Unique = true;
  NewColumn.AutoIncrement = true;

  dtDepartments.Columns.Add(NewColumn);

  NewColumn = new DataColumn();

  NewColumn.ColumnName = "DepartmentName";
```

```
        NewColumn.DataType = Type.GetType("System.String");
        NewColumn.Unique = true;
        NewColumn.AllowDBNull = false;

        dtDepartments.Columns.Add(NewColumn);

        //Add Departments table to EmployeeInfo DataSet
        dsEmployeeInfo.Tables.Add(dtDepartments);
    }
```

לאחר שנוצר מופע (instance) מ- DataSet dsEmployeeInfo, ניצור את הטבלה Employee באמצעות אחד הבנאים של DataTable והעברת שם הטבלה. לאחר מכן נקבע את ערך התכונה CaseSensitive של הטבלה. תכונה זאת קובעת אם פעולות המיון, החיפוש, הסינון ופעולות נוספות בטבלה מבוצעות עם רגישות לאותיות רישיות (case sensitive). כברירת מחדל, ערך זה נקבע כערך התכונה CaseSensitive של אובייקט DataSet המכיל אותו, או כ-False אם האובייקט DataTable נוצר בנפרד מ-DataSet.

הערה

התכונה CaseSensitive חלה על נתונים באובייקט DataTable בלבד. תכונה זו אינה משפיעה על כללי התלות ברישיות אשר חלים על אובייקטי DataTable עצמם. אובייקט DataSet עשוי לכלול טבלה אחת או יותר (או יחסים) עם שם דומה, אך ברישיות אותיות שונה, כגון mytable ו-Mytable. במקרה כזה, ההפניות לטבלאות חייבות להתאים במדויק ('חיפוש תלוי רישיות'). עם זאת, אם קיימת רק טבלה אחת בשם זה, כל התייחסות לשם תתבצע בהצלחה ('חיפוש שאינו תלוי רישיות').

לאחר מכן נוסיף שלוש הגדרות של שדות לאובייקט DataTable בשם dtEmployees, באמצעות הפעלת השיטה Add של Columns עם שם השדה וסוג הנתונים. שים לב כי סוג הנתונים שצוין הוא סוג נתוני NET. ולא סוג מסד נתונים. אם לא נציין סוג נתונים, סוג השדה ייקבע כברירת מחדל כמחרוזת (string). לאחר מכן, הטבלה Employees תתווסף ל-DataSet בשם dsEmployeeInfo.

נחזור על התהליך בטבלה Departments, והפעם נשתמש בבנאי ובפונקציות אחרות, אשר יבצעו את אותה הפעולה בדרך שונה. תוכל לבחור גישה על פי העדפותיך, לפי נהלי החברה או לפי דרישות של המשימה הספציפית.

ערך התכונה MinimumCapacity של אובייקט DataTable בשם dtDepartments נקבע לערך 5, אשר מציין כי מופע הטבלה יתחיל ביצירה פנימית של 5 רשומות. הגדרת תכונה זו עם ערך השונה מברירת המחדל (25) מאפשרת להשפיע על הקצאת משאבים נחוצים, ועשויה לשפר את הביצועים במצבים קריטיים. מבחינת המשתמש, רשומות אלו לא יתקיימו בפועל עד להוספת רשומות לטבלה.

בעמודות אלו נקבעים גם מאפייני עמודות שונים לפני שנוסיף אותם לסכמה המוגדרת באמצעות האוסף Columns. אתה כבר אמור להכיר תכונות כגון ReadOnly, Unique, AllowDBNull ו-AutoIncrement, מכיון שכבר צברת ניסיון ביצירת יישומי מסד נתונים.

- ReadOnly – ערך True מציין כי לא ניתן לשנות את ערך השדה.
- Unique – ערך True מציין כי כל אחד מהערכים בשדה זה בכל רשומות הטבלה חייב להיות ייחודי. ניתן ליישם מאפיין זה על ידי יצירת UniqueConstraint אוטומטי עבור השדה. נדון בשיטה זו בסעיף "אילוצי טבלאות" בהמשך פרק זה.
- AllowDBNull – ערך True מציין כי ערכים ריקים (null) מותרים בשדה זה.
- AutoIncrement – ערך True מציין כי ערך השדה גדל בכל פעם שמתווספת רשומה (שורה) חדשה לטבלה. תוכל לציין ערך התחלתי וקפיצה באמצעות התכונות AutoIncrementSeed ו-AutoIncrementStep, בהתאמה.

הערה

טבלה תקבל רשומה עם ערך המוקצה עבור שדה, כאשר AutoIncrement נקבע ל-True, וישתמש בערך AutoIncrement רק אם ערך השדה שונה מערך ברירת המחדל.

בין התכונות הרגילות האחרות של DataColumn נמצא את MaxLength (לשדות מסוג String), DefaultValue ו-Table, אשר מחזיר התייחסות לטבלה שבה נמצא השדה.

ניתן להגדיר שדה אשר יכיל ביטוי המשמש לחישוב ערכו, ליצור שדה מצטבר או כדי לסנן רשומות. ביטוי זה יכול להיות מורכב מערכי השדות הנלקחים מהרשומה הנוכחית או מרשומות אחרות, מקבועים, אופרטורים, תווי הכללה (wildcards), פונקציות מצטברות (aggregate functions) או פונקציות ביטוי אחרות. למידע נוסף ולדוגמאות של ביטויי שדות (עמודות), פנה לעזרה עבור התכונה Expression של DataColumn.

הוספת נתונים ל-DataTable

לאחר הגדרת DataTable והסכמה הדרושה, תוכל להוסיף רשומות (שורות) נתונים. קוד 3.2 שעליך להוסיף ל-frmDataSets, מדגים כיצד להוסיף לטבלה רשומות נתונים על ידי תכנות. השגרה AddData מוסיפה ארבע רשומות נתונים לטבלאות Departments ושלוש רשומות נתונים לטבלה Employee, בתהליך של שלושה שלבים.

1. צור מופע חדש של DataRow עבור הטבלה הרצויה, על ידי קריאה לשיטה NewRow המתאימה לטבלה.
2. הקצה ערכים לשדות ברשומה זו.
3. הוסף את הרשומה (שורה) לאוסף Rows של הטבלה על ידי העברת האובייקט DataRow לשיטה Add של התכונה Rows השייכת לטבלה.

קוד 3.2-VB קוד להוספת נתונים לטבלה על ידי תכנות

```
Private Sub AddData()
    Dim dtDepartments As DataTable = dsEmployeeInfo.Tables("Departments")
    Dim dtEmployees As DataTable = dsEmployeeInfo.Tables("Employees")

    'Add 4 records to Departments table
    Dim rowDept As DataRow

    rowDept = dtDepartments.NewRow
    rowDept("ID") = 11
    rowDept("DepartmentName") = "Administration"
    dtDepartments.Rows.Add(rowDept)

    rowDept = dtDepartments.NewRow
    rowDept("ID") = 22
    rowDept("DepartmentName") = "Engineering"
    dtDepartments.Rows.Add(rowDept)

    rowDept = dtDepartments.NewRow
    rowDept("ID") = 33
    rowDept("DepartmentName") = "Sales"
    dtDepartments.Rows.Add(rowDept)

    rowDept = dtDepartments.NewRow
    rowDept("ID") = 44
    rowDept("DepartmentName") = "Marketing"
    dtDepartments.Rows.Add(rowDept)

    'Add 3 records to the Employees table
```

```
    Dim rowEmployee As DataRow
    rowEmployee = dtEmployees.NewRow
    rowEmployee("FirstName") = "Jackie"
    rowEmployee("LastName") = "Goldstein"
    rowEmployee("DepartmentID") = 44
    dtEmployees.Rows.Add(rowEmployee)

    rowEmployee = dtEmployees.NewRow
    rowEmployee("FirstName") = "Jeffrey"
    rowEmployee("LastName") = "McManus"
    rowEmployee("DepartmentID") = 33
    dtEmployees.Rows.Add(rowEmployee)

    rowEmployee = dtEmployees.NewRow
    rowEmployee("FirstName") = "Meir"
    rowEmployee("LastName") = "Johnson"
    rowEmployee("DepartmentID") = 33
    dtEmployees.Rows.Add(rowEmployee)

    rowEmployee = dtEmployees.NewRow
    rowEmployee("FirstName") = "Sam"
    rowEmployee("LastName") = "Johnson"
    rowEmployee("DepartmentID") = 22
    dtEmployees.Rows.Add(rowEmployee)

End Sub
```

קוד #C-3.2 קוד להוספת נתונים לטבלה על ידי תכנות

```
private void AddData()
{
  DataTable dtDepartments = dsEmployeeInfo.Tables["Departments"];
  DataTable dtEmployees = dsEmployeeInfo.Tables["Employees"];

  //Add 4 records to Departments table
  DataRow rowDept;
  rowDept = dtDepartments.NewRow();
  rowDept["ID"] = 11;
  rowDept["DepartmentName"] = "Administration";
  dtDepartments.Rows.Add(rowDept);

  rowDept = dtDepartments.NewRow();
  rowDept["ID"] = 22;
```

```
    rowDept["DepartmentName"] = "Engineering";
    dtDepartments.Rows.Add(rowDept);

    rowDept = dtDepartments.NewRow();
    rowDept["ID"] = 33;
    rowDept["DepartmentName"] = "Sales";
    dtDepartments.Rows.Add(rowDept);

    rowDept = dtDepartments.NewRow();
    rowDept["ID"] = 44;
    rowDept["DepartmentName"] = "Marketing";
    dtDepartments.Rows.Add(rowDept);

    //Add 3 records to the Employees table
    DataRow rowEmployee;
    rowEmployee = dtEmployees.NewRow();
    rowEmployee["FirstName"] = "Jackie";
    rowEmployee["LastName"] = "Goldstein";
    rowEmployee["DepartmentID"] = 44;
    dtEmployees.Rows.Add(rowEmployee);

    rowEmployee = dtEmployees.NewRow();
    rowEmployee["FirstName"] = "Jeffrey";
    rowEmployee["LastName"] = "McManus";
    rowEmployee["DepartmentID"] = 33;
    dtEmployees.Rows.Add(rowEmployee);

    rowEmployee = dtEmployees.NewRow();
    rowEmployee["FirstName"] = "Meir";
    rowEmployee["LastName"] = "Johnson";
    rowEmployee["DepartmentID"] = 33;
    dtEmployees.Rows.Add(rowEmployee);

    rowEmployee = dtEmployees.NewRow();
    rowEmployee["FirstName"] = "Sam";
    rowEmployee["LastName"] = "Johnson";
    rowEmployee["DepartmentID"] = 22;
    dtEmployees.Rows.Add(rowEmployee);

  }
```

הערה

תוכל להוסיף לטבלה רשומת נתונים חדשה, על ידי העברת מערך (אובייקטים) המכיל את הנתונים בסדר השדות שבהגדרת הטבלה אל השיטה Add. גישה זו להוספת עובד נוסף בקוד 3.2 תיראה כך:

VB

```
Dim empData(2) As Object
empData(0) = "Sam"
empData(1) = "Johnson"
empData(2) = 3
dtEmployees.Rows.Add(empData)
```

C#

```
object[] empData = new object[3];
empData[0] = "Sam";
empData[1] = "Johnson";
empData[2] = 3;
dtEmployees.Rows.Add(empData);
```

גישה לנתונים מתוך האובייקט DataTable

מכיון שה-DataSet והאובייקטים מסוג DataTable הכלולים בו תמיד מלאים ואינם מחוברים למקור נתונים, הדרך לגשת לרשומות נתונים שונה במידה ניכרת מהשיטה שהשתמשת בה ב-ADO ובמודלים קודמים לגישה לנתונים (כגון: ODBC, DAO ו-RDO). **כל** הנתונים זמינים בו-זמנית, ולכן תפיסת הרשומה הנוכחית אינה קיימת. עובדה זו מרמזת על כך שאין תכונות או שיטות למעבר מרשומה אחת לרשומה אחרת. כל טבלת נתונים, DataTable, מכילה את התכונה Rows שהיא למעשה אוסף אובייקטי DataRow. הגישה לאובייקטים אלה מתבצעת באמצעות אינדקס או באמצעות הפקודה For Each (ב-#C רושמים מחובר – foreach). לכן, אובייקטי ADO.NET מציעים גישת מערך פשוטה, קלה ויעילה יותר לניווט, וגישה אל רשומות נתונים.

קוד 3.3 מציג את השגרה DisplayDataSet, אשר מציגה את תוכן הטבלאות שהגדרת קודם וגם טענת בנתונים. שגרה זו משתמשת בגישה המועדפת של ביצוע לולאה על כל רכיביו של אוסף מסוים – במקרה זה האוספים Rows ו-Columns – להצגת הטבלה Employees. להצגת טבלאות Department, שגרה זו משתמשת באינדקס נומרי שהיא הגישה החילופית להצגת נתוני הרשומות והשדות.

קוד 3.3-VB הצגת נתונים מטבלאות

```
Private Sub DisplayDataSet()
        Dim dr As DataRow
        Dim dc As DataColumn

        Me.lstOutput.Items.Add("DISPLAY DATASET")
        Me.lstOutput.Items.Add("===============")

        ' Display data in Employees table
        For Each dr In dsEmployeeInfo.Tables("Employees").Rows
            For Each dc In dsEmployeeInfo.Tables("Employees").Columns
                Me.lstOutput.Items.Add(dc.ColumnName & " : " & dr(dc))
            Next
            Me.lstOutput.Items.Add("-----")
        Next
        Me.lstOutput.Items.Add("")

        ' Display data in Departments table
        ' Show how to use index, instead of For Each
        Dim row As Integer
        Dim col As Integer

        For row = 0 To dsEmployeeInfo.Tables("Departments").Rows.Count - 1
            For col = 0 To dsEmployeeInfo.Tables("Departments").Columns.Count - 1
                Me.lstOutput.Items.Add( _
                 dsEmployeeInfo.Tables("Departments").Columns(col).ColumnName _
                 & " : " & dsEmployeeInfo.Tables("Departments").Rows(row)(col))
            Next col
            Me.lstOutput.Items.Add("-----")
        Next row
    End Sub
```

קוד 3.3-C# הצגת נתונים מטבלאות

```
private void DisplayDataSet()
{
    lstOutput.Items.Add("DISPLAY DATASET");
    lstOutput.Items.Add("===============");

    // Display data in Employees table
    foreach (DataRow dr in dsEmployeeInfo.Tables["Employees"].Rows)
    {
```

```
        foreach (DataColumn dc in dsEmployeeInfo.Tables["Employees"].Columns)
        {
          lstOutput.Items.Add(dc.ColumnName + " : " + dr[dc]);
        }
        lstOutput.Items.Add("-----");
        lstOutput.Items.Add("");
    }

    int row,col;

    for (row=0; row<dsEmployeeInfo.Tables["Departments"].Rows.Count-1; ++row)
    {
        for(col=0; col<dsEmployeeInfo.Tables["Departments"].Columns.Count - 1;++col)
        {
    lstOutput.Items.Add(dsEmployeeInfo.Tables["Departments"].Columns[col].Colu
mnName + " : "  + dsEmployeeInfo.Tables["Departments"].Rows[row][col]);
        }
        lstOutput.Items.Add("-----");
    }
}
```

ניתן לכתוב את השגרה כולה בצורה כללית יותר, על ידי שימוש בלולאות גנריות לא רק לרשומות ושדות, אלא גם לטבלאות שב-DataSet. קוד 3.4 מדגים גישה זו.

קוד 3.4-VB יישום גנרי של DisplayDataSet

```
Private Sub DisplayDataSet(ByVal ds As DataSet)
        ' Generic routine to display the contents of a DataSet
        ' DataSet to be displayed is passed as parameter
        Dim dt As DataTable
        Dim dr As DataRow
        Dim dc As DataColumn

        Me.lstOutput.Items.Add("DISPLAY DATASET")
        Me.lstOutput.Items.Add("===============")

        For Each dt In ds.Tables
            Me.lstOutput.Items.Add("")
            Me.lstOutput.Items.Add("TABLE: " & dt.TableName)
            Me.lstOutput.Items.Add("---------------------------")
            For Each dr In dt.Rows
                For Each dc In dt.Columns
                    Me.lstOutput.Items.Add(dc.ColumnName & " : " & dr(dc))
                Next
```

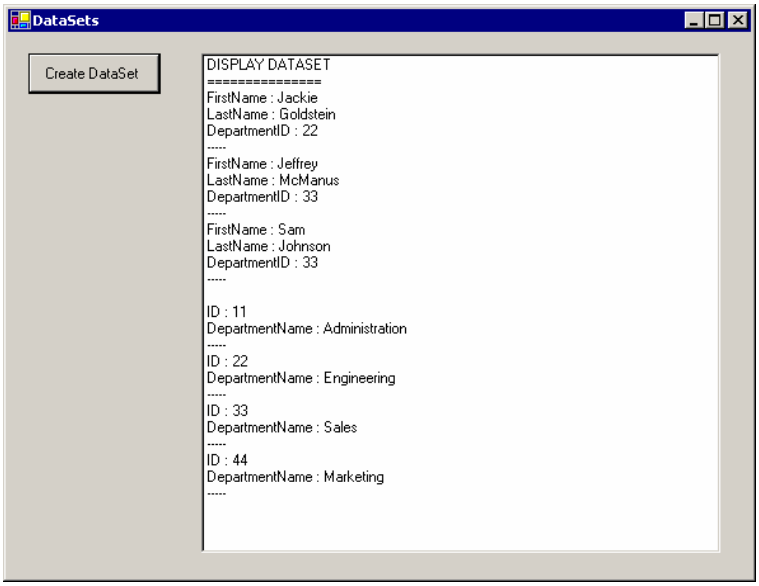
```
            Me.lstOutput.Items.Add("-----")
        Next
    Next dt
End Sub
```

קוד #C-3.4 יישום גנרי של DisplayDataSet

```
private void DisplayDataSet(DataSet ds)
{
    // Generic routine to display the contents of a DataSet
    // DataSet to be displayed is passed as parameter
    lstOutput.Items.Add("DISPLAY DATASET");
    lstOutput.Items.Add("===============");

    // Display data in Employees table
    foreach (DataRow dr in ds.Tables["Employees"].Rows)
    {
        foreach (DataColumn dc in ds.Tables["Employees"].Columns)
        {
          lstOutput.Items.Add(dc.ColumnName + " : " + dr[dc]);
        }
        lstOutput.Items.Add("-----");
        lstOutput.Items.Add("");
    }
}
```

שים לב להעמסת יתר (overloading) של השגרה DisplayDataSet במצב בו יש לנו שגרה עם שם זהה וחתימה נפרדת. הגירסה הגנרית שלנו מקבלת להצגה את ה-DataSet כפרמטר.

כעת, לאחר שבידך כל המרכיבים הדרושים, תוכל להפעיל את הפרויקט DataSetCode. כאשר תלחץ על Create DataSet, ה-DataSet והטבלאות ייווצרו, ימולאו בנתונים ויוצגו. פלט התוצאה מוצג בתרשים 3.2.



תרשים 3.2 תוצאות יצירה, מילוי והצגת הטבלאות Employees ו-Departments.

הערה

כדי לבדוק את הגירסה הגנרית של DisplayDataSet מ-btnCreateDS_Click, הוסף את הפרמטר ds לקריאה ל-DisplayDataSet שב-btnCreateDS_Click.

```
DisplayDataSet(dsEmployeeInfo)
```

חיפוש, סינון ומיון רשומות

לעיתים, לא תרצה לפעול על כל הרשומות הזמינות בטבלה. ייתכן שתרצה לפעול על רשומה מסוימת אחת או על חלק מהרשומות הזמינות. שני מנגנונים מאפשרים לעשות זאת: השיטה Find והשיטה Select.

השיטה Find שייכת לתכונה Rows של האובייקט DataTable. כלומר, של האובייקט DataRowCollection. שיטה זו מאפשרת למצוא ולהחזיר רשומה יחידה על פי ערך המפתח העיקרי של הטבלה.

לפני שתוכל להשתמש ב-Find לאיתור רשומה מסוימת בטבלה Departments אשר מוגדרת בקוד 3.1, עליך להגדיר מפתח ראשי עבור הטבלה, או לוודא שכבר הוגדר מפתח כזה. ניתן לעשות זאת על ידי הקצאת עמודת טבלה אחת או יותר לתכונה PrimaryKey (התכונה PrimaryKey תכיל מערך אובייקטים מסוג DataColumn גם אם המפתח הראשי הוא שדה יחיד בלבד).

הוסף קוד לטופס frmDataSets אשר יציג את הרשומה שאותרה ב-DataSet.

1. מהכרטיסיה Windows Forms של תיבת הכלים, הוסף לחצן מתחת ללחצן btnChildRows.
2. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**btnFind1** ואת ערך התכונה Text ל-**Find1**.
3. הוסף את קוד 3.5.

קוד 3.5-VB קוד לאיתור רשומה, כאשר המפתח הראשי מכיל שדה אחד

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles Button1.Click
      'Create reference to the table Departments
      Dim dtDepartments As DataTable
      dtDepartments = dsEmployeeInfo.Tables("Departments")

      ' Set Primary Key
      Dim pk(0) As DataColumn
      pk(0) = dtDepartments.Columns("DepartmentName")
      dtDepartments.PrimaryKey = pk

      ' Find the row
      Dim desiredRow As DataRow
      desiredRow = dtDepartments.Rows.Find("sales")
   End Sub
```

קוד 3.5-C# קוד לאיתור רשומה, כאשר המפתח הראשי מכיל שדה אחד

```
private void btnFind1_Click(object sender, System.EventArgs e)
       {
         // Create reference to the table Departments
         DataTable dtDepartments = dsEmployeeInfo.Tables["Departments"];

         // Set Primary Key
         DataColumn[] pk = new DataColumn[1];
         pk[0] = dtDepartments.Columns["DepartmentName"];
         dtDepartments.PrimaryKey = pk;

         // Find the row
         DataRow desiredRow;
         desiredRow = dtDepartments.Rows.Find("sales");
         HelperDisplayDataRow(desiredRow);

       }
```

הקוד מייצר מופע של הטבלה dtDepartments הקשורה לטבלה Department. לאחר מכן, מתבצעת הקצאת השדה DepartmentName לתכונה PrimaryKey (שים לב שניתן להוסיף קטע קוד זה גם בסוף קוד 3.1), ואז מופעלת השיטה Find. ערך המשתנה desiredRow ייקבע ל-DataRow עם ערך המפתח הראשי המתאים, או שהוא ייקבע ב-VB ל-Nothing וב-#C ל-null, אם לא נמצאה הרשומה הרצויה. את הרשומה שאוחזרה מציגים באמצעות הפונקציה HelperDisplayDataRow, שנמצאת בקוד המקור.

הערה

כאשר שדה יחיד מגדיר את המפתח הראשי (PrimaryKey) עבור טבלה, ערך התכונה AllowDBNull של השדה נקבע באופן אוטומטי ב-VB לערך False וב-#C ל-false, ואילו ערך התכונה Unique של השדה נקבע באופן אוטומטי ל-True. אם המפתח הראשי מורכב ממספר שדות, ערך התכונה AllowDBNull של השדות ייקבע לערך False בלבד.

אם מפתח ראשי של טבלה מורכב מיותר מאשר שדה אחד, הערכים הרצויים של כל אחד משדות המפתח הראשי יועברו כאלמנטים של מערך מסוג Object אל השיטה Find (ב-#C המערך הוא מסוג object):

VB

```
' Set Primary Key
Dim pk(1) As DataColumn
pk(0) = dtEmployees.Columns("FirstName")
pk(1) = dtEmployees.Columns("LastName")
dtEmployees.PrimaryKey = pk

' Find the row
Dim desiredRow As DataRow
Dim desiredValues(1) As Object
desiredValues(0) = "Sam"
desiredValues(1) = "Johnson"

desiredRow = dtEmployees.Rows.Find(desiredValues)
```

C#

```
// Set Primary Key
DataColumn[] pk = new DataColumn[2];
pk[0] = dtEmployees.Columns["FirstName"];
pk[1] = dtEmployees.Columns["LastName"];
dtEmployees.PrimaryKey = pk;

DataRow desiredRow;
// Find the row
object[] desiredValues = new object[2];
```

```
desiredValues[0] = "Sam";
desiredValues[1] = "Johnson";

desiredRow = dtEmployees.Rows.Find(desiredValues);
```

השיטה Select של האובייקט DataTable מחזירה מערך אובייקטים מסוג DataRow. הרשומות המוחזרות עשויות להתאים לקריטריון סינון, סדר מיון ו/או ציון מצב (DataViewRowState).

שורות הקוד הבאות מחזירות ומציגות את השמות הפרטיים של כל העובדים אשר שם המשפחה שלהם הוא "Johnson":

VB

```
Dim selectedRows() As DataRow
selectedRows = dtEmployees.Select("LastName = 'Johnson' ")
Dim i As integer
For i = 0 To selectedRows.GetUpperBound(0)
    MessageBox.Show(selectedRows(i) ("FirstName"))
Next
```

C#

```
//Create reference to the table Departments
DataTable dtEmployees;
dtEmployees = dsEmployeeInfo.Tables["Employees"];

DataRow[] selectedRows;

// Do the selection of the data.
selectedRows = dtEmployees.Select("LastName = 'Johnson'");

for (int i = 0; i <= selectedRows.GetUpperBound(0); ++i)
   MessageBox.Show(selectedRows[i]["FirstName"].ToString());
```

הקוד המלא נמצא בתוך האירוע btnFind3_Click על הפקד Find3.

אם תרצה גם למיין את הרשומות לפי שם פרטי בסדר יורד, תוכל לשנות את הקריאה לשיטה Select:

```
selectedRows = dtEmployees.Select("LastName = 'Johnson' ", "FirstName DESC")
```

לבסוף, ציון מצב בארגומנט של השיטה Select יאפשר לאחזר גרסאות רצויות של הרשומות מהטבלה, כאשר אתה נמצא במצב עריכה. לדוגמה, אחזור כל הערכים המקוריים של הרשומות, למרות העובדה כי בוצעו בהן שינויים רבים (אך לא בוצעה עדיין הקריאה ל-AcceptChanges), יבוצע על ידי ציון מצב הרשומה OriginalRows בשיטה שנבחרה:

```
selectedRows = dtEmployees.Select(Nothing, Nothing, _
    DataViewRowState.OriginalRows)
```

השתמש בשורה שלהלן כדי לבחור ברשומות החדשות שהתווספו, אשר מכילות את שם המשפחה "Johnson":

```
selectedRows = dtEmployees.Select("LastName = 'Johnson'", Nothing, _
    DataViewRowState.Added)
```

אם ברצונך גם למיין אותן לפי שם פרטי, כפי שבוצע קודם לכן, השתמש בשורת קוד זו:

```
selectedRows = dtEmployees.Select("LastName = 'Johnson'", "FirstName DESC", _
    DataViewRowState.Added)
```

האפשרויות שתוכל לציין עבור מצב הרשומה מוצגות בטבלה 3.1, אשר מתארת את הערכים של התכונה DataViewRowState. ההפניות מתיחסות לשינויים שבוצעו מאז הטעינה האחרונה של הטבלה, או מאז הקריאה האחרונה ל-AcceptChanges.

טבלה 3.1 המנייה DataViewRowState

ערך	תיאור
Added	רשומות חדשות שהתווספו.
CurrentRows	כל הרשומות הנוכחיות (כולל רשומות חדשות, רשומות שהשתנו ורשומות שלא השתנו).
Deleted	כל הרשומות שסומנו למחיקה.
ModifiedCurrent	הגרסאות הנוכחיות של רשומות שהשתנו.
ModifiedOriginal	הגרסאות המקוריות של רשומות שהשתנו.
None	אף לא אחת.
OriginalRows	כל הרשומות המקוריות (כולל רשומות שלא השתנו ורשומות שנמחקו, אך לא רשומות חדשות).
Unchanged	כל הרשומות שלא השתנו.

עדכון DataSet

כדי לעדכן רשומה מסוימת בטבלה, עליך לגשת לרשומה זו ולהקצות ערך חדש לאחת מעמודותיה. אם ברצונך לשנות את שם המחלקה sales בטבלה Departments ל- sales and money, עליך למצוא את הרשומה בטבלה (כפי שראית בסעיף הקודם), ולאחר מכן תוכל עדכן אותה. לאחר קטע הקוד בו מצאת את הרשומה מהסעיף הקודם, עליך רק להוסיף את השורה הבאה:

VB

```
desiredRow ("DepartmentName") = "sales and money"
```

C#

```
desiredRow ["DepartmentName"] = "sales and money";
```

תוכל לערוך שינויים כרצונך, אך הם יתבצעו בפועל רק לאחר קריאה לשיטה AcceptChanges שעושה זאת. תוכל לקרוא גם לשיטה RejectChanges אשר תבטל את הפעולה ותחזיר את המצב הקודם, תבטל שינויים שהתבצעו מאז טעינת הנתונים או מאז הקריאה האחרונה ל-AcceptChanges.

הערה

השיטה AcceptChanges (וגם RejectChanges) זמינה ברמות שונות. המחלקות DataTable, DataSet ו-DataRow תומכות כולן בשיטה זו. קריאה ל-AcceptChanges של DataSet תגרום לקריאה ל-AcceptChanges בכל טבלה ב-DataSet. קריאה ל-AcceptChanges של DataTable תגרום לקריאה ל-AcceptChanges בכל רשומה בטבלה זו. לכן, תוכל לבצע שינויים רשומה אחר רשומה על ידי קריאה נפרדת ל-AccpetChanges עבור כל רשומה, או בפעולה אחת על ידי קריאה אחת ל-AcceptChanges ב-DataSet המכיל את הנתונים. דבר זה נכון גם עבור השיטה RejectChanges.

מחיקת רשומה

ניתן להוסיף או למחוק רשומות שלמות. כבר הדגמנו כיצד להוסיף רשומות וכעת נעסוק במחיקת רשומה. דרך אחת למחיקת רשומה היא לקרוא לשיטה Remove של האובייקט DataRowCollection (כלומר, התכונה Rows של האובייקט DataTable). שיטה זו מסירה את הרשומה מהאוסף. דרך נוספת לעשות זאת היא לקרוא לשיטה Delete של אובייקט DataRow ספציפי, אשר מסמנת את הרשומה למחיקה. זוהי פעולה אשר מתקיימת בפועל לאחר קריאה ל-AcceptChanges.

לאחר הקריאה והפעלת השיטה Remove, יאבדו כל נתוני הרשומה שמתייחסים אליה. לא ניתן יהיה להחזיר את הרשומה שנמחקה גם על ידי קריאה ל-RejectChanges.

RowStates ו-Versions

לכל אובייקט DataRow יש תכונה בשם RowState אשר מציינת את המצב (הסטטוס) הנוכחי של הרשומה. בנוסף, כל רשומה מכילה עד ארבע גרסאות של הערכים הקיימים עבורה. כאשר מבוצעות פעולות עריכה שונות על הרשומה, מצבה ו/או גרסאות הערכים ישתנו. הערכים של התכונה RowState בטבלה 3.2 והערכים של התכונה DataRowVersion בטבלה 3.3.

טבלה 3.2 התכונה RowState

ערך	תיאור
Unchanged	לא בוצעו שינויים מאז הקריאה האחרונה ל-AcceptChanges, או מאז שהרשומה מולאה במקור על ידי DataAdapter.
Added	הרשומה התווספה ל-DataRowCollection (כלומר, התכונה Rows של DataTable), אך לא בוצעה קריאה ל-AcceptChanges.
Deleted	השיטה Delete נקראה למחיקת הרשומה, אך לא בוצעה עדיין קריאה ל-AcceptChanges.
Modified	הרשומה שונתה, אך לא בוצעה קריאה ל-AcceptChanges.
Detached	הרשומה נוצרה אך לא התווספה ל-DataRowCollection, או שנקראה השיטה Remove אשר מסירה את הרשומה מ-DataRowCollection, או שנקראה השיטה Delete להסרת הרשומה ולאחר מכן בוצעה קריאה ל-AcceptChanges.

טבלה 3.3 התכונה DataRowVersion

ערך	תיאור
Original	הערכים המקוריים של הרשומה. גירסה זו אינה קיימת ברשומה שבה ערך RowState הוא Added.
Current	הערכים הנוכחיים (ייתכן שהשתנו) של הרשומה. גירסה זו אינה קיימת ברשומה שבה ערך RowState הוא Deleted.
Default	גרסת ברירת המחדל של הרשומה, אשר תלויה ב-Rowstate הנוכחי. אם הערך RowState הוא Deleted, גרסת ברירת המחדל של הרשומה היא Original. אם הערך RowState הוא Detached, גרסת ברירת המחדל של הרשומה היא Proposed. אחרת, גרסת ברירת המחדל של הרשומה היא Current.
Proposed	הערכים המוצעים עבור הרשומה. גירסה זו קיימת כאשר מתבצעת פעולת עריכה בלבד (התחלה על ידי BeginEdit וסיום על ידי EndEdit או CancelEdit), או ברשומה שלא התווספה ל-DataRowCollection.

אם ערך RowState של רשומה הוא Deleted, כאשר מתבצעת קריאה לשיטה AcceptChanges, הרשומה תוסר מ-DataRowCollection. אחרת, גרסת הרשומה Original תעודכן בגרסת הרשומה Current, וערך RowState יהפוך ל-Unchanged.

בהתאם לכך, אם ערך RowState של רשומה הוא Added, כאשר בוצעה קריאה ל-RejectChanges, הרשומה תוסר מ-DataRowCollection. אחרת, גרסת הרשומה Current תעודכן בגרסת הרשומה Original, וערך RowState יהפוך ל-Unchanged.

הערה

ארבע גרסאות הרשומה אינן זמינות בכל המצבים. באפשרותך לקרוא לשיטה HasVersion של DataRow כדי לבדוק אם גירסה מסוימת זמינה במצב הנוכחי. שיטה זו מעבירה את אחד מאיברי מניית DataRowVersion ומחזירה ערך בוליאני המציין אם הגירסה שצוינה זמינה כעת.

גרסת Proposed של רשומה

עליך לדעת מספר נקודות על הגירסה Proposed של DataRow. כאשר מבוצעת קריאה לשיטה BeginEdit של DataRow, הפעולות והאירועים הרגילים מושהים והדבר מאפשר למשתמש לבצע שינויים שונים ברשומה מבלי לגרום להפעלת כללי בדיקה. במצב זה, השינויים המבוצעים אינם נראים בגירסה Current של הרשומה, אלא בגירסה Proposed. לאחר שבוצעה קריאה לשיטה EndEdit, ערכי Proposed מועברים אל ערכי Current. ניתן לבטל את השינויים על ידי קריאה ל-CancelEdit לפני הקריאה ל-EndEdit. שים לב כי אף אחד מהשינויים האלה לא יהיה קבוע עד לקריאה ל-AcceptChanges.

הערה

תוכל לגשת תמיד לגירסה מסוימת של עמודת DataRow (בהנחה שגירסה זו קיימת), על ידי ציון הגירסה הרצויה כפרמטר שני בשיטה Item של DataRow. דבר זה נכון בכל פעם שתקרא לשיטה Item בצורה מפורשת (explicity) או משתמעת (implicity):

```
VB:  dtEmployees.Rows(2).Item ("lastname", DataRowVersion.Proposed)
C#:  dtEmployees.Rows[2].Item ("lastname", DataRowVersion.Proposed);
```

או כך:

```
VB:  dtEmployees.Rows(2) ("lastname", DataRowVersion.Original)
C#:  dtEmployees.Rows[2].Item ("lastname", DataRowVersion.Original);
```

שגיאות בעמודות ורשומות

ב-ADO.NET כלול מנגנון גמיש להגדרה וטיפול בשגיאות משתמש ברשומות ובעמודות של הטבלה. מנגנון זה מאפשר בדיקת נתונים המוגדרת על ידי היישום. ניתן לסמן שגיאות כאשר הן מתגלות, אך לדחות את ההתראה עליהן עד לשלב מאוחר יותר במהלך פעולת היישום (אין לבלבל שגיאות אלו עם שגיאות המערכת הרגילות אשר מוגדרות על ידי מערכת החריגים – exceptions, ומטופלות על ידי מנגנון הטיפול הסטנדרטי – Try-Catch-Finally של NET Frameworks.).

כאשר יישום מגלה שגיאת בדיקה, הוא מסמן אותה על ידי קביעת תיאור שגיאה לרשומה או לשדה מסוים. הגדרת ערך התכונה RowError של DataRow מציינת שהתגלתה שגיאה כלשהי ברשומה המסוימת, כגון:

VB: myDataRow.RowError = "Something wrong here"
C#: myDataRow.RowError = "Something wrong here";

קריאה לשיטה SetColumnError של DataRow מציינת שגיאה בשדה מסוים, כגון:

VB: myDataRow.SetColumnError (2, "Bad data in this column")
C#: myDataRow.SetColumnError (2, "Bad data in this column");

תוכל לאחזר את מחרוזות השגיאה עבור רשומה או שדה על ידי גישה לתכונה RowError או על ידי קריאה ל-GetColumnError, בהתאמה. תוכל לאפס שגיאות אלו על ידי קביעת מחרוזות השגיאה המתאימות כמחרוזת ריקה ("")), או על ידי קריאה לשיטה ClearErrors של DataRow, אשר מאפסת את התכונה RowError ואת כל השגיאות שנקבעו על ידי קריאה ל-SetColumnError.

ל-DataRow קיימת תכונה בשם HasErrors אשר ערכה True, אם קיימות שגיאות ברשומה (ברמת הרשומה או ברמת השדה). ערך תכונה זו מוצג ברמות הטבלה וב-DataSet – אם ערך HasErrors הוא True עבור רשומה כלשהי בטבלה, התכונה HasErrors של הטבלה תהיה גם היא True. בדומה לכך, אם ערך התכונה HasError בטבלה כלשהי ב-DataSet הוא True, ערך התכונה HasErrors של ה-DataSet יהיה אף הוא True. השיטה GetErrors של האובייקט DataTable מחזירה מערך אובייקטים מסוג DataRow אשר מכיל שגיאות. מערך זה מהווה מנגנון לגילוי מהיר אם והיכן קיימות שגיאות בדיקה, כפי שמוצג בקוד 3.6.

קוד 3.6-VB איתור שגיאות בכל טבלאות DataSet

```
Private Sub ResolveErrors(ByVal myDataSet As DataSet)
    Dim rowsWithErrors() As DataRow
    Dim myTable As DataTable
    Dim myCol As DataColumn
    Dim currRow As Integer
```

```
    For Each myTable In myDataSet.Tables
        If myTable.HasErrors Then
            ' Get All rows that have errors
            rowsWithErrors = myTable.GetErrors()

        End If
        For currRow = 0 To rowsWithErrors.GetUpperBound(0)
            For Each myCol In myTable.Columns
                ' Find columns with errors and decide
                ' how to deat with it.
                ' A column's error is retrieved with:
                ' rowsWithErrors(currRow).GetColumnError(myCol)
            Next

            ' Clear the Row's error
            rowsWithErrors(currRow).ClearErrors()
        Next
    Next

  End Sub
```

קוד #3.6-C איתור שגיאות בכל טבלאות DataSet

```
private void  ResolveErrors(DataSet myDataSet)
{
   DataRow[]  rowsWithErrors;
   DataColumn[] colArr;

  foreach(DataTable myTable in myDataSet.Tables)
  {
     if (myTable.HasErrors)
     {
        rowsWithErrors = myTable.GetErrors();
        for (int currRow = 0; currRow <rowsWithErrors.Length; ++currRow)
        {
         colArr = rowsWithErrors[currRow].GetColumnsInError();
         for(int i = 0; i < colArr.Length; i++)
         {
           // A column's error is retrieved with:
           rowsWithErrors[currRow].GetColumnError(colArr[i]);
         }
```

```
        rowsWithErrors[currRow].ClearErrors();
      }
    }
  }
}
```

קשרים בין טבלאות

DataSet יכול להכיל מספר טבלאות, ולכן הציפייה הטבעית (אם עבדת בעבר עם מסדי נתונים טבלאיים) היא כי ניתן יהיה ליצור קשרים (relations) בין הטבלאות. ב-ADO.NET, האובייקט DataRelation מספק תכונה זו.

קישור DataRelation מתייחס לעמודות בשתי טבלאות אשר יש ביניהן קשרי אב-בן (Parent-Child), או קשרי מפתח ראשי-מפתח זר (primary key-foreign key). הדוגמה הקלאסית לקשר זה היא טיפול בלקוחות ובהזמנות, כאשר רשומת לקוח קשורה לרשומת הזמנות אחת או יותר. בדוגמה זו רשומת הלקוח היא האב, ואילו ההזמנה או ההזמנות הן הבנים. נמחיש נושא זה באמצעות דוגמה שהצגנו קודם לכן – הטבלאות Department (אב) ו-Employees (בן) המוגדרות ב-DataSet שלנו.

האובייקט DataRelation תומך בשתי פעולות נפרדות:

- הוא מאפשר ניווט בין הטבלאות הקשורות, בכך שהוא הופך את הרשומות הקשורות לרשומה שאתה פועל עליה לזמינות. אם אתה פועל על רשומת אב, האובייקט DataRelation יספק את רשומות הבן. אם אתה פועל על רשומת בן, האובייקט יספק את רשומת האב שלה.
- הוא מאפשר אכיפת כללים אחידים בקשרים, כגון שינויים מדורגים לטבלאות אליהן קיימים הקשרים, כאשר מבצעים פעולות על רשומות באחת מהטבלאות האלו.

המשך עם הטופס frmDataSets שהכנת קודם.

1. מהכרטיסיה Windows Forms של תיבת הכלים, הוסף לחצן תחת הלחצן btnCreateDS.
2. בחלון Properties קבע את ערך התכונה Name של הלחצן ל-**btnCreateRelations** ואת ערך התכונה Text ל- **Create Relations**.
3. הוסף את קוד 3.7.

קוד 3.7-VB קוד ליצירה והצגת קשרים בין טבלאות

```
Private Sub btnCreateRelations_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles btnCreateRelations.Click
      Dim rel As DataRelation

      CreateDataSet()

      ' Create the relation between the Departments and Employees tables
      rel = dsEmployeeInfo.Relations.Add("relDepartmentEmployees", _
        dsEmployeeInfo.Tables("Departments").Columns("ID"), _
        dsEmployeeInfo.Tables("Employees").Columns("DepartmentID"))

      DisplayRelations(dsEmployeeInfo)
    End Sub

    Private Sub DisplayRelations(ByVal ds As DataSet)
      Dim rel As DataRelation

      ' Print the names of each column in each table through the Relations.
      Me.lstOutput.Items.Add("")
      Me.lstOutput.Items.Add("DISPLAY RELATIONS")
      For Each rel In ds.Relations
         ' Display Relation Name
         Me.lstOutput.Items.Add("NAME: " & rel.RelationName)
         ' Show Parent table & field
         Me.lstOutput.Items.Add("PARENT: " & rel.ParentTable.ToString & " -- " & _
           rel.ParentColumns(0).ColumnName)
         ' Show Child table & field
         Me.lstOutput.Items.Add("CHILD: " & rel.ChildTable.ToString & " -- " & _
           rel.ChildColumns(0).ColumnName)
      Next
      Me.lstOutput.Items.Add("")
    End Sub
```

קוד #C-3.7 קוד ליצירה והצגת קשרים בין טבלאות

```
private void btnCreateRelations_Click(object sender, System.EventArgs e)
{
    DataRelation rel;
    CreateDataSet();

    // Create the relation between the Departments and Employees tables
    rel = dsEmployeeInfo.Relations.Add("relDepartmentEmployees",
    dsEmployeeInfo.Tables["Departments"].Columns["ID"],
    dsEmployeeInfo.Tables["Employees"].Columns["DepartmentID"]);
    DisplayRelation(dsEmployeeInfo);
}

private void DisplayRelation(DataSet ds)
{
    // Print the names of each column in each table through the Relations.
    lstOutput.Items.Add("");
    lstOutput.Items.Add("DISPLAY RELATIONS");
    foreach (DataRelation rel in ds.Relations)
    {
        // Display Relation Name
        lstOutput.Items.Add("NAME: " + rel.RelationName);
        // Show Parent table & field
        lstOutput.Items.Add("PARENT: " + rel.ParentTable.TableName.ToString() + " -- "
          + rel.ParentColumns[0].ColumnName);
        // Show Child table & field
        lstOutput.Items.Add("CHILD: " + rel.ChildTable.ToString() + " -- "
          + rel.ChildColumns[0].ColumnName);
    }
    lstOutput.Items.Add("");
}
```

הדבר הראשון שעליך לעשות הוא ליצור את אובייקט DataRelation המתאים. לכל DataSet יש אוסף קשרים (relations) שהוא נחשף אליהם כתכונת Relations שלו. תכונה זו היא מסוג DataRelationCollection ותומכת במספר העמסות יתר (overloads) של השיטה Add. הצורה בה קוד 3.7 משתמש מקבלת שלושה ארגומנטים: שם הקשר, הפניה ל-DataColumn בטבלת האב והפניה ל-DataColumn בטבלת הבן. אם הקשר בין הטבלאות כולל יותר משדה אחד, ניתן לקרוא לגירסה אחרת של השיטה Add באמצעות שימוש בארגומנטים שהיו מערכים של אובייקטים מסוג DataColumn.

הפרוצדורה DisplayRelations מבצעת לולאה על הקשרים בתכונה Relations של DataSet שהיא מקבלת כארגומנט. עבור כל קשר קיים מוצגים שם הקשר, שם טבלת האב ועמודת הקישור, וגם שם טבלת הבן ועמודת הקישור.

הערה

כדי להפוך את DisplayRelations לכללי יותר, תוכל להוסיף קוד אשר יבצע לולאה על כל השדות במערך התכונות ParentColumns ו-ChildColumns, במקום להציג את האלמנט הראשון בלבד, כפי שעשית כעת.

כאשר תפעיל את הפרויקט DataSetCode ותלחץ על הלחצן Create Relations, תיבת הרשימה תציג את פרטי הקשרים שנוצרו בין הטבלאות Employees ו-Departments.

בנוסף לאוסף Relations של ה-DataSet, אשר מכיל את כל הקשרים המוגדרים בין טבלאות באותו ה-DataSet, לכל טבלה יש שני אוספי קשרים (תכונות): ParentRelations ו-ChildRelations, אשר מכילים את הקשרים בין הטבלה לבין טבלאות האב והבן הקשורות, בהתאמה.

כעת, לאחר שאתה יכול לגשת להגדרות הקשרים שבין הטבלאות, תוכל גם לנווט בין הטבלאות ולאחזר את הנתונים הקשורים. התחל בהוספת לחצן וקוד לטופס frmDataSets שהכנת קודם.

1. מהכרטיסיה Windows Forms שבתיבת הכלים, הוסף לחצן מתחת ללחצן btnCreateRelations.
2. בחלון Properties קבע את ערך התכונה Name של הלחצן ל-**btnChildRows** ואת ערך התכונה Text ל- **Child Rows**.
3. הוסף את קוד 3.8.

קוד 3.8-VB קוד להצגת נתוני אב ובן מטבלאות שיש אליהן קישור

```
Private Sub btnChildRows_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles btnChildRows.Click
      Dim rel As DataRelation
      CreateDataSet()
      AddData()

      ' Create the relation between the Departments and Employees tables
      rel = dsEmployeeInfo.Relations.Add("relDepartmentEmployees", _
        dsEmployeeInfo.Tables("Departments").Columns("ID"), _
        dsEmployeeInfo.Tables("Employees").Columns("DepartmentID"))

      DisplayChildRows(dsEmployeeInfo.Tables("Departments"))
  End Sub
```

```
Private Sub DisplayChildRows(ByVal dt As DataTable)
    Dim rel As DataRelation
    Dim relatedRows() As DataRow
    Dim row As DataRow
    Dim col As DataColumn
    Dim i As Integer
    Dim rowData As String

    Me.lstOutput.Items.Add("")
    Me.lstOutput.Items.Add("CHILD ROWS FOR TABLE:  " _
        & dt.TableName)
    For Each row In dt.Rows
        For Each rel In dt.ChildRelations
            Me.lstOutput.Items.Add( _
              dt.TableName & ": " & _
              rel.ParentColumns(0).ColumnName & _
              "= " & row(rel.ParentColumns(0).ToString))

            relatedRows = row.GetChildRows(rel)
            ' Print values of rows.
            For i = 0 To relatedRows.GetUpperBound(0)
                rowData = "**** " & _
                          rel.ChildTable.TableName & ":"
                For Each col In rel.ChildTable.Columns
                    rowData = rowData & " " & _
                          relatedRows(i)(col.ToString)
                Next col
                Me.lstOutput.Items.Add(rowData)
            Next i
        Next rel
    Next row
End Sub
```

קוד #C-3.8 קוד להצגת נתוני אב ובן מטבלאות שיש אליהן קישור

```
private void btnChildRows_Click(object sender, System.EventArgs e)
{
    DataRelation rel;
    CreateDataSet();
    AddData();

    // Create the relation between the Departments and Employees tables
    rel = dsEmployeeInfo.Relations.Add("relDepartmentEmployees",
    dsEmployeeInfo.Tables["Departments"].Columns["ID"],
    dsEmployeeInfo.Tables["Employees"].Columns["DepartmentID"]);

    DisplayChildRows(dsEmployeeInfo.Tables["Departments"]);
}

private void DisplayChildRows(DataTable dt)
{
    DataRow[] relatedRows;
    string rowData;
    lstOutput.Items.Add("");
    lstOutput.Items.Add("CHILD ROWS FOR TABLE:  " + dt.TableName);

    foreach(DataRow row in dt.Rows)
    {
        foreach(DataRelation rel in dt.ChildRelations)
        {
        lstOutput.Items.Add(
          dt.TableName + ": "
          + rel.ParentColumns[0].ColumnName
          + "= " +  row[rel.ParentColumns[0].ToString()]);
        relatedRows = row.GetChildRows(rel);
          // Print values of rows.
          for(int i=0; i<= relatedRows.GetUpperBound(0); ++i)
          {
                    rowData = "**** "
                    + rel.ChildTable.TableName + ":";
                    foreach(DataColumn col in rel.ChildTable.Columns)
                              rowData = rowData + " "
                                        + relatedRows[i][col.ToString()];

                              lstOutput.Items.Add(rowData);
          }
        }
    }
}
```

מטפל האירועים (ה-handler) btnChildRows_Click של הלחצן יוצר תחילה את ה-DataSet ו-DataTables על ידי קריאה ל-CreateDataSet (שהוצגה קודם לכן בקוד 3.1), ולאחר מכן קריאה ל-AddData (שהוצגה קודם לכן בקוד 3.2), כדי למלא את הטבלה בנתונים. בשלב הבא, מטפל האירועים יוצר את הקשר בין הטבלאות Employees ו-Departments על ידי שורת קוד מהשגרה btnCreateRelations_Click, אשר הוצגה כבר בקוד 3.7. לבסוף מתבצעת קריאה ל-DisplayChildRows, כאשר הטבלה Departments מועברת כטבלת ההורה.

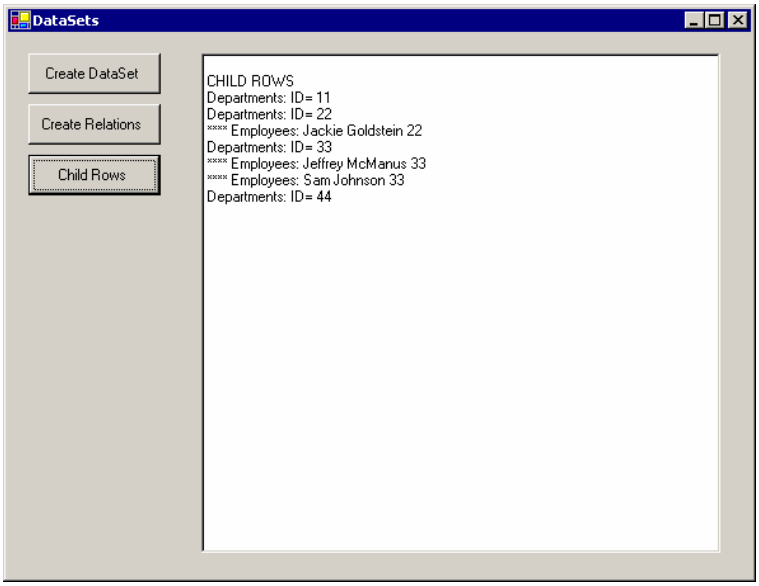
האובייקט DisplayChildRows מיישם לולאה עם קינון משולש, אשר מציגה את כל שדות הנתונים בכל טבלה שקיים אליה קישור (במקרה זה, טבלה אחת בלבד) עבור כל רשומה בטבלת האב. עבור כל רשומה בטבלת האב אשר מועברת כארגומנט, האובייקט עובר על כל הקשרים המוגדרים בתכונה ChildRelations של הטבלה, מציג את שם הטבלה, את שם השדה בטבלת האב ואת ערך השדה הזה ברשומה הנוכחית. השיטה GetChildRows של הרשומה נקראת, והקישור הנוכחי מועבר לה כארגומנט; היא מחזירה מערך DataRows המכיל את רשומות הבן המתאימות. כל ערכי השדות מוצגים עבור כל אחת מרשומות אלו, ולפני כל רשומה מוצגים כוכביות ושם טבלת הבן.

הערה

גרסאות מסוימות של GetChildRows מקבלות ארגומנט נוסף אשר מגדיר איזו גירסה של הרשומות יש להחזיר (כפי שמוגדר במנייה DataRowVersion המוצגת בטבלה 3.3). קיימות שיטות דומות להשגת רשומה או רשומת אב של רשומת בן נתון.

אתה עשוי לתהות מדוע ההצהרה הקודמת מתייחסת לרשומות אב, ברבים. כיצד יכול בן לקבל יותר מאב אחד? התשובה היא, שלמרות שהקשר מגדיר לרוב אב יחיד עבור כל רשומת בן (כלומר, ערכי עמודת אב ייחודיים), היא מאפשרת גם הגדרה של עמודות אב לא ייחודיות והגדרת ערכת שיטות לאחזור מספר רשומות אב (GetParentRows), ולא רק רשומת אב יחידה (GetParentRow).

כאשר תפעיל את הפרויקט DataSetCode ותלחץ על הלחצן Child Rows, רשומות הבן שבטבלה Employees, אשר מתייחסות לכל רשומת אב בטבלה Department, יוצגו בתיבת הרשומה, כפי שמוצג בתרשים 3.3.



תרשים 3.3 תוצאות הצגת רשומות אב ורשומות בן מתאימות בטבלאות Departments ו-Employees.

אילוצי טבלאות

אילוצים (Constraints) הם כללים לאכיפת הגבלות מסוימות על שדה אחד או יותר בטבלה, ומטרתם היא להבטיח את שלמות הנתונים. ב-ADO.NET כלולה תמיכה בשני סוגי אילוצים: UniqueConstraint ו-ForeignKeyConstraint. האילוץ מסוג UniqueConstraint מבטיח כי כל הערכים בשדה או בשדות שפורטו הינם ייחודיים בטבלה. אילוץ מסוג ForeignKeyConstraint מגדיר התייחסות של מפתח ראשי אל מפתח זר (primary key-foreign key relationship), אשר קיימת בין עמודות בשתי טבלאות. הדבר מאפשר ציון פעולות אשר יבוצעו כאשר רשומות בן (מפתח עיקרי) מתווספות, נמחקות או משתנות. ניסיון להפר את האילוץ יגרום לשגיאת הפעלה (run-time error).

שים לב לכך שניתן לאכוף אילוצים רק כאשר ערך התכונה EnforceConstraints של ה-DataSet, אשר מכיל את הטבלה, הוא True. ערך ברירת המחדל של תכונה זו הוא True.

ניתן ליצור אילוצים בצורה ישירה, אך לרוב הם נוצרים בצורה עקיפה. למעשה, כבר יצרת מספר אילוצים בדוגמאות הקוד הקודמות. אובייקט UniqueConstraint נוצר ומתווסף באופן אוטומטי לאוסף Constraints של DataTable בכל פעם שאתה קובע את ערך התכונה Unique של DataColumn ל-True, ובכל פעם שאתה יוצר מפתח ראשי עבור טבלה. בנוסף, UniqueConstraint ו-ForeignKeyConstraint נוצרים באופן אוטומטי בכל פעם שאתה יוצר DataRelation בין שתי טבלאות. UniqueConstraint נוצר בשדה או

בשדות הקשורים בטבלת האב, ואילו ForeignKeyConstraint נוצר בשדה או בשדות הקשורים בטבלת הבן.

הערה

תוכל ליצור קישור נתונים, DataRelation, אשר מתייחס לשתי טבלאות מבלי ליצור בפועל את שני האילוצים שהוזכרו. יעילות גישה זו מוטלת בספק, אך ניתן להשתמש בה.

כעת נוסיף קוד לטופס frmDataSets, אשר יציג את אילוצי הטבלאות ב-DataSet:

1. מהכרטיסיה Windows Forms של תיבת הכלים, הוסף לחצן מתחת ללחצן btnChildRows.
2. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**btnConstraints** ואת ערך התכונה Text ל-**Constraints**.
3. הוסף את קוד 3.9.

קוד 3.9-VB קוד להצגת אילוצים ייחודיים ואילוצי מפתח זר

```
Private Sub btnConstraints_Click(ByVal sender As _
            System.Object, ByVal e As System.EventArgs) _
            Handles btnConstraints.Click
    Dim dt As DataTable
    Dim rel As DataRelation

    CreateDataSet()

    ' Create the relation between the Departments and Employees tables
    rel = dsEmployeeInfo.Relations.Add( _
      "relDepartmentEmployees", _
      dsEmployeeInfo.Tables("Departments").Columns("ID"), _
      dsEmployeeInfo.Tables("Employees").Columns("DepartmentID"))

    For Each dt In dsEmployeeInfo.Tables
      DisplayConstraints(dt)
    Next dt
  End Sub

  Private Sub DisplayConstraints(ByVal dt As DataTable)
    Dim i As Integer
    Dim cs As Constraint
    Dim uCS As UniqueConstraint
    Dim fkCS As ForeignKeyConstraint
```

```
Dim columns() As DataColumn

Me.lstOutput.Items.Add("")
Me.lstOutput.Items.Add( _
        "CONSTRAINTS FOR TABLE:  " & dt.TableName)
Me.lstOutput.Items.Add( _
        "====================================")

For Each cs In dt.Constraints
   Me.lstOutput.Items.Add( _
        "Constraint Name: " & cs.ConstraintName)
   Me.lstOutput.Items.Add( _
        "Type: " & cs.GetType().ToString())

   If TypeOf cs Is UniqueConstraint Then
      uCS = CType(cs, UniqueConstraint)
      ' Get the Columns as an array.
      columns = uCS.Columns

      ' Print each column's name.
      For i = 0 To columns.Length - 1
         Me.lstOutput.Items.Add( _
               "Column Name: " & _
               columns(i).ColumnName)
      Next i
   ElseIf TypeOf cs Is ForeignKeyConstraint Then
      fkCS = CType(cs, ForeignKeyConstraint)

      ' Get the child Columns and display them
      columns = fkCS.Columns
      For i = 0 To columns.Length - 1
         Me.lstOutput.Items.Add( _
            "Column Name: " & _
            columns(i).ColumnName)
      Next i

      ' Display the related (parent) table name
      Me.lstOutput.Items.Add( _
            "Related Table Name: " & _
            fkCS.RelatedTable.TableName)

      ' Get the related (parent) columns and display them.
      columns = fkCS.RelatedColumns
      For i = 0 To columns.Length - 1
```

```
            Me.lstOutput.Items.Add( _
                "Related Column Name: " & _
                columns(i).ColumnName)
          Next i
        End If
        Me.lstOutput.Items.Add("-----")
      Next cs
    End Sub
```

קוד #C-3.9 קוד להצגת אילוצים ייחודיים ואילוצי מפתח זר

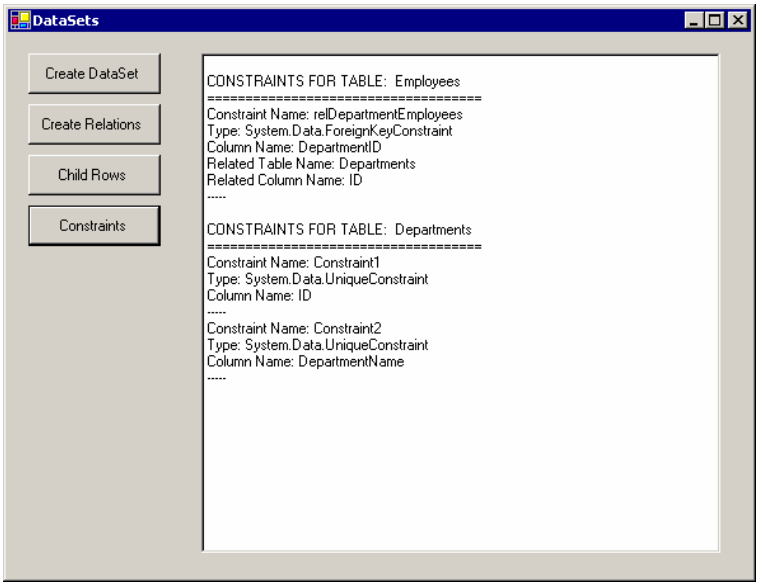
```
private void btnConstraints_Click(object sender, System.EventArgs e)
      {

        DataRelation rel;

        CreateDataSet();

        // Create the relation between the Departments and Employees tables
        rel = dsEmployeeInfo.Relations.Add(
          "relDepartmentEmployees",
          dsEmployeeInfo.Tables["Departments"].Columns["ID"],
          dsEmployeeInfo.Tables["Employees"].Columns["DepartmentID"]);

        foreach(DataTable dt in dsEmployeeInfo.Tables)
          DisplayConstraints(dt);

      }

      private void DisplayConstraints(DataTable dt)
      {
        UniqueConstraint uCS;
        ForeignKeyConstraint  fkCS;
        DataColumn[] columns;
        lstOutput.Items.Add("");
        lstOutput.Items.Add("CONSTRAINTS FOR TABLE:  " + dt.TableName);
        lstOutput.Items.Add(
              "==================================");

        foreach(Constraint cs in dt.Constraints)
        {
```

```
lstOutput.Items.Add("Constraint Name: " + cs.ConstraintName);
lstOutput.Items.Add("Type: " + cs.GetType().ToString());

        if (cs is UniqueConstraint)
        {

                uCS = (UniqueConstraint)cs;
                columns = uCS.Columns;
                // Print each column's name.
                for (int i=0; i< columns.Length; ++i)
                {
                        lstOutput.Items.Add(
                                "Column Name: "
                                +columns[i].ColumnName);
                }

        }
        else if (cs is ForeignKeyConstraint)
        {
                fkCS = (ForeignKeyConstraint)cs;
                columns = fkCS.Columns;

                for (int i=0; i< columns.Length; ++i)
       lstOutput.Items.Add(
          "Column Name: "
          +columns[i].ColumnName);
      // Display the related (parent) table name
      lstOutput.Items.Add(
         "Related Table Name: "
         +fkCS.RelatedTable.TableName);

                // Get the related (parent) columns and display them.
      columns = fkCS.RelatedColumns;
      for (int i=0; i< columns.Length; ++i)
        lstOutput.Items.Add(
         "Related Column Name: "
         +columns[i].ColumnName);

        }
        lstOutput.Items.Add("-----");
  }

}
```

מטרת השיטה btnConstraints_Click היא להגיב ללחיצת הלחצן; הגדרת ה-DataSet, ה-DataTables וה-DataRelation (באמצעות שימוש בקוד שנכתב ברשימות קודמות); ולאחר מכן קריאה ל-DisplayConstraints, אשר מבצעת את הפעולות העיקריות.

DisplayConstraints היא שגרה גנרית אשר מקבלת טבלה כפרמטר ומציגה נתונים אודות האילוצים המוגדרים עבור טבלה זו. השגרה מבצעת לולאה על כל האילוצים בתכונה Constraints של הטבלה המעובדת. עבור כל אילוץ, יש לבדוק אם הוא UniqueConstraint או ForeignKeyConstraint. שתי המחלקות האלו נגזרות מהמחלקה המופשטת Constraint, כאשר הן יכולות להתקיים באוסף מאותו הסוג. עם זאת, לכל אחת מהן יש קבוצת תכונות שונות, ולכן עליך לזהות את האילוץ ולאחר מכן להמיר את האובייקט לסוג המסוים המתאים. עבור UniqueConstraint, הצג את שם או שמות כל העמודות המוגדרות באילוץ. עבור ForeignKeyConstraint, הצג גם את שם טבלת האב הקשורה, יחד עם שמות העמודות הקשורות בטבלה זו.

תוצאות ההפעלה של הפרויקט DataSetCode ולחיצה על הלחצן Constraints מוצגות בתרשים 3.4. זכור כי למרות שמוצגים שלושה אילוצים (אחד בטבלה Employees ושניים בטבלה Departments), אף אחד מהם לא נוצר בצורה מפורשת. אילוצים אלה נוצרו באופן אוטומטי כאשר קבעת את ערך התכונה הייחודית DataColumn ל-True, ויצרת את הקישורים בין הטבלאות.



תרשים 3.4 תוצאות הצגת האילוצים בטבלאות Departments ו-Employees.

לאובייקט ForeignKeyConstraint יש שלוש תכונות של חוקים, אשר שולטות בפעולות המבוצעות במהלך עריכת הטבלאות הקשורות. התכונות UpdateRule ו-DeleteRule

מגדירות את הפעולות שיש לבצע, כאשר רשומה בטבלת האב מעודכנת או נמחקת. האפשרויות עבור תכונות אלו מוגדרות ב-enum בשם Rule, ומוצגות בטבלה 3.4.

טבלה 3.4 enum בשם Rule

ערך	תיאור
Cascade	מחיקה או עדכון המבוצעים ברשומת האב מבוצעים גם ברשומה או ברשומות הבן. זהו ערך ברירת המחדל.
None	מחיקה או עדכון המבוצעים ברשומת האב לא יבוצעו ברשומה או ברשומות הבן הקשורות. תנאי זה עלול ליצור רשומות בן שקיימים בהן קשרים אל רשומות אב לא תקפות.
SetDefault	מחיקה או עדכון המבוצעים ברשומת האב לא יבוצעו ברשומה או ברשומות הבן הקשורות. במקום זאת, השדה הקשור (מפתח זר) ברשומות הבן הקשורות מקבל את ערך ברירת המחדל המוגדר עבורו.
SetNull	מחיקה או עדכון המבוצעים ברשומת האב לא יבוצעו ברשומה או ברשומות הבן. לחילופין, ערך השדה הקשור נקבע ל-DBNull. תנאי זה עלול ליצור רשומות בן "יתומות", אשר אין להן קשר לרשומות אב.

התכונה השלישית היא AcceptRejectRule; ערכה יכול להיות Cascade או None, והיא קובעת אם הפעלת השיטה AcceptChanges או RejectChanges על רשומת אב גורמת להפעלה אוטומטית של AcceptChanges או RejectChanges ברשומות הבן הקשורות אליה (אל רשומת האב). ערך ברירת המחדל הוא Cascade, ומשמעותו – אם מבוצעת קריאה ל-AcceptChanges או RejectChanges ברשומת אב, השיטה המתאימה תיקרא באופן אוטומטי עבור רשומות הבן הקשורות. אם הערך הוא None, קריאה לאחת מהשיטות האלו ברשומת האב לא תשפיע על עריכת רשומות הבן הקשורות.

שימוש ברכיב DataSet

הכרטיסיה Data של תיבת הכלים Visual Studio מכילה רכיב DataSet המאפשר לקבוע ערכי תכונות באמצעות החלון Properties, במקום לבצע זאת באמצעות קוד. ניתן להשוות שיטה זו לשיטה המתוארת בפרק 2 באובייקטים Connection ו-Command. כעת נעבור על השלבים להגדרת DataSet והאובייקטים המשויכים, באמצעות ההגדרות שבדוגמאות הקוד בסעיפים הקודמים.

תחילה נגדיר את סכמת הטבלה וה-DataSet.

1. הוסף טופס **frmDataSetComponent** אל הפרויקט DataSetCode.
2. בחלון Properties של frmDataSetComponent, קבע את ערך התכונה Text ל- **DataSet Component**.

3. הגדל את frmDataSetComponent.

4. מהכרטיסיה Windows Forms של תיבת הכלים, הוסף תיבת רשימה (listbox) ל-frmDataSetComponent והצב אותה בצד הימני של הטופס.

5. בחלון Properties, קבע את ערך השם של תיבת הרשימה ל-lstOutput.

6. הגדל את תיבת הרשימה באופן שתכסה כ-80% משטח הטופס.

7. מהכרטיסיה Data של תיבת הכלים, גרור רכיב DataSet אל שטח העיצוב של frmDataSetComponent. בתיבת הדו-שיח שתופיע, בחר בלחצן האפשרויות **Untyped dataset** ולחץ **OK**. רכיב זה אינו גלוי בזמן ההפעלה, ולכן הוא יופיע ב-tray הרכיבים שתחת שטח העיצוב של הטופס.

8. בחלון Properties של רכיב DataSet החדש בשם DataSet1, קבע את ערך התכונה Name ל-**dsEmployeeInfo**.

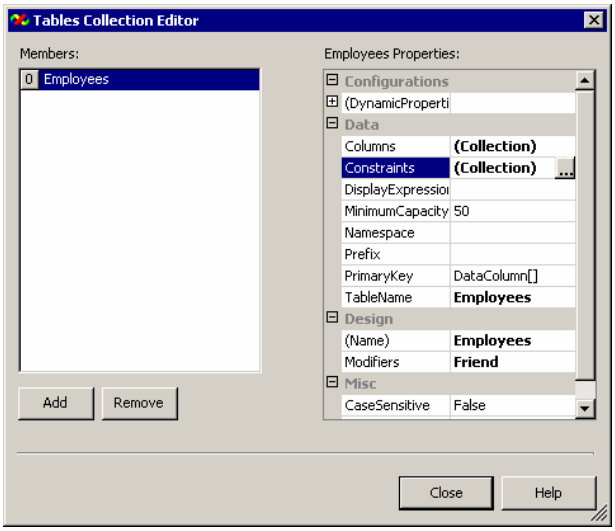
9. בחר בתכונה **Tables** שבחלון Properties, ולאחר מכן לחץ על הלחצן ("...") כדי להציג את Tables Collection Editor.

10. לחץ **Add** כדי להציג את תכונות הטבלה הראשונה שב-DataSet.

11. בלוח Table1 Properties, קבע את ערך התכונה TableName ל-**Employees**.

בלוח Employees (שים לב, שם הלוח **משתנה** לאחר ביצוע הפעולה), קבע את ערך התכונה (Name) ל-**Employees**.

התוצאות עד עתה מוצגות בתרשים 3.5 (בנקודה זו חלק מהשדות שלא שינית בעצמך **אינם זהים** ב-VB וב-C#).



תרשים 3.5 Tables Collection Editor לאחר הוספת הטבלה Employees.

12. בחר בתכונה **Columns** שבלוח Employees Properties, ולאחר מכן לחץ על הלחצן ("...") כדי להציג את Columns Collection Editor.

13. לחץ **Add** כדי להציג את תכונות השדה הראשון בטבלה Employees.

14. בלוח Column1 Properties, קבע את ערך התכונה ColumnName ל-**FirstName**.

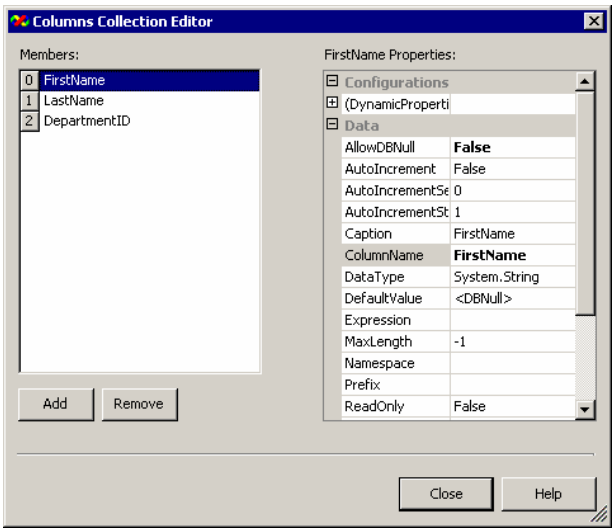
15. לחץ **Add** כדי להציג את תכונות השדה השני בטבלה Employees.

16. בלוח Column1 Properties, קבע את ערך התכונה ColumnName ל-**LastName** (שים לב ששם הלוח **משתנה** לאחר ביצוע הפעולה).

17. לחץ **Add** כדי להציג את תכונות השדה השלישי בטבלה Employees.

18. בלוח LastName Properties, קבע את ערך התכונה ColumnName ל-**DepartmentID** ואת ערך התכונה DataType ל-**System.Int32**.

תרשים 3.6 מציג את Columns Collection Editor של הטבלה Employees.



תרשים 3.6 Columns Collection Editor של הטבלה Employees.

19. ב- Columns Collection Editor לחץ **Close**, כדי לחזור ל- Tables Collection Editor של ה-DataSet בשם dsEmployeesInfo ולהוסיף את הטבלה Departments.

20. לחץ **Add** כדי להציג את תכונות הטבלה השנייה שב-DataSet.

21. בלוח Table1 Properties, קבע את ערך התכונה TableName ל-**Departments**.

22. בחר בתכונה **MinimumCapacity** שבלוח Departments Properties וקבע את ערכה ל-**5**.

23. בחר בתכונה **Columns** בלוח Departments Properties ולאחר מכן לחץ על הלחצן (...) כדי להציג את Columns Collection Editor.

24. לחץ **Add** כדי להציג את תכונות השדה הראשון בטבלה Departments.

25. בלוח Column1 Properties, קבע את ערך התכונה ColumnName ל-**ID** ואת ערך התכונה DataType ל-**System.Int32**.

26. בלוח ID Properties, קבע ל-True את ערכי התכונות **ReadOnly**, **Unique** ו-**AutoIncrement**.

27. לחץ **Add** כדי להציג את תכונות השדה השני בטבלה Departments.

28. בלוח Column1 Properties, קבע את ערך התכונה ColumnName ל-**DepartmentName**.

29. בלוח DepartmentName Properties, קבע את ערך התכונה Unique ל-**True** ואת ערך התכונה AllowDBNull ל-**False**.

30. לחץ **Close** ב- Columns Collection Editor כדי לחזור ל- Tables Collection Editor. לאחר מכן לחץ **Close** כדי לסגור את Tables Collection Editor.

כעת יישמת את ה-DataSet בשם dsEmployeesInfo ואת הטבלאות Employees ו-Department על ידי הגדרת תכונות של רכיבי עיצוב בערכים זהים, כפי שעשית בקוד (זמן הפעלה) המוצג בקוד 3.1.

כעת, המשך להגדיר את רכיבי העיצוב ואת הקשרים בין הטבלאות שב-DataSet.

1. בחלון Properties של רכיב ה-DataSet בשם dsEmployeesInfo, בחר בתכונה **relations**. לאחר מכן לחץ על הלחצן (...) כדי להציג את Relations Collection Editor.

2. לחץ **Add** כדי להציג את תכונות הקשר הראשון ב-DataSet.

3. קבע את ערך התכונה Name ל-**relDepartmentEmployees**.

4. קבע את ערך התכונה ParentTable ל-**Departments**, על ידי בחירת **Departments** מתיבת הרשימה הנפתחת השמאלית.

5. קבע את ערך התכונה ChildTable ל-**Employees**, על ידי בחירת **Employees** מתיבת הרשימה הנפתחת הימנית.

6. באזור Columns, קבע את ערך הרשומה הראשונה בשדה השמאלי Key Columns ל-**ID**, על ידי בחירת ID מתיבת הרשימה הנפתחת. פעולה זו תקבע את ערך התכונה ParentColumns של DataRelation.

7. באזור Columns, קבע את ערך הרשומה הראשונה בשדה Foreign Key Columns ל- Department ID, על ידי בחירת האפשרות **DepartmentID** מתיבת הרשימה הנפתחת. פעולה זו תקבע את ערך התכונה ChildColumns של DataRelation.

8. קבל את ערכי ברירת המחדל של תכונות הכללים Update, Delete ו-AcceptReject, ואל תשנה אותם.

9. לחץ **OK** כדי לסגור את Relations Collection Editor.

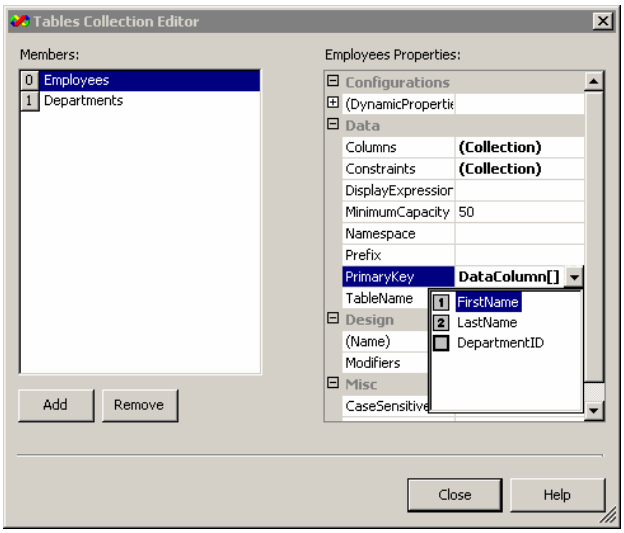
ההגדרה החסרה היחידה כעת היא התכונה PrimaryKey של כל טבלה. לקביעת ערך התכונה, בצע את הפעולות שלהלן:

1. בחר בתכונה **Tables** בחלון Properties של הרכיב dsEmployeeInfo, ולחץ על הלחצן (...) כדי להציג את Tables Collection Editor.

2. בחלונית Members, בחר בטבלה **Employees**.

3. בחלונית Employees Properties, בחר בתכונה **PrimaryKey** ולאחר מכן לחץ על הלחצן בצורת החץ כדי לפתוח את תיבת הרשימה הנפתחת.

4. בחר בשדה או בשדות המרכיבים את המפתח הראשי מרשימת השדות הזמינים אשר מוצגת. אם המפתח הראשי מורכב ממספר שדות, ודא שבחרת אותם בסדר הרצוי. במקרה זה, בחר בשדה FirstName ולאחר מכן בשדה LastName, כפי שמוצג בתרשים 3.7.



תרשים 3.7 בחירת מספר עמודות להגדרת ערך התכונה PrimaryKey.

5. הקש **Enter** כדי לקבל את ההגדרות שביצעת בתכונה PrimaryKey.

6. בחלונית Members, בחר בטבלה **Departments**.

7. בחלונית Departments Properties, בחר בתכונה **PrimaryKey** ולאחר מכן לחץ על לחצן החץ כדי לפתוח את תיבת הרשימה הנפתחת.

8. בחר בשדה **DepartmentName** מרשימת השדות הזמינים אשר מוצגים, ולאחר מכן הקש **Enter** כדי לקבל את ההגדרות עבור התכונה PrimaryKey.

9. לחץ **Close** כדי לסגור את Tables Collection Editor.

כדי להראות שאותן תוצאות תתקבלנה גם על ידי שימוש ברכיבי עיצוב, כפי שעשית קודם בעת השימוש בקוד בלבד, העתק והדבק מספר שגרות מ-frmDataSets אל frmDataSetComponents ולאחר מכן הפעל את frmDataSetComponent.

1. בחר והעתק את השגרה **AddData** (וכל הקוד שכלול בה) מ-frmDataSet, והדבק אותה אל frmDataSetComponent.

2. הוסף את שורת הקוד הבאה בתחילת המחלקה. שורת קוד זו **אינה** דרושה עבור **VB**.

C#

```
using System.Data;
```

3. חזור על שלב 1 עם השגרות DisplayDataSet ו-DisplayChildRows.

4. הוסף את הקוד שלהלן למטפל באירועים (event handler) בשם frmDataSetComponent_Load, אשר נמצא ב-DataSetComponent:

```
Private Sub frmDataSetComponent_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load
    AddData()
    DisplayDataSet()
    DisplayChildRows(dsEmployeeInfo.Tables("Departments"))
End Sub
```

5. אם אתה משתמש ב-VB – דלג לסעיף 7. בעת שימוש ב-C# – המשך לסעיף 6.

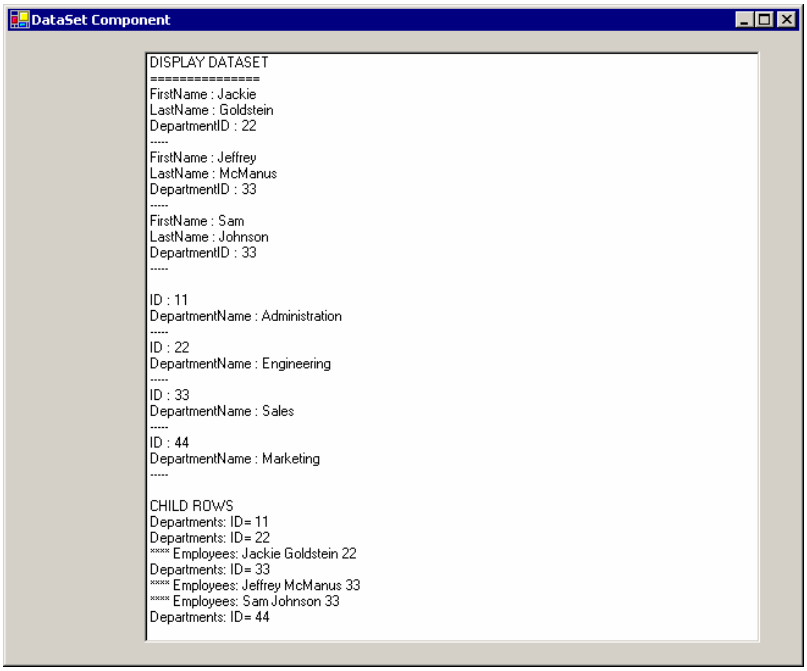
6. הוספת Main למחלקה של C# מתבצעת באמצעות הוספת קטע הקוד הבא (שים לב ששם המחלקה המודגש בקוד **זהה** לשם המחלקה שהשתמשת בו):

```
static void Main()
{
  Application.Run(new frmDataSetComponent());
 }
```

7. לחץ לחיצה ימנית על הפרויקט **DataSetCode** שב- Solution Explorer, ומתפריט הקיצור בחר **Properties**.

8. בחר בפריט **General** שבתיקיה Common Properties, ולאחר מכן קבע את ערך התכונה Startup object ל-frmDataSetComponent (אם אתה משתמש ב-C#, בחירה זו אפשרית רק עבור מחלקות שהפונקציה Main כלולה בהן, כמו בסעיף 6 לעיל).

אם תפעיל כעת את הפרויקט DataSetCode, תיבת הרשימה ב-frmDataSetComponent תציג את כל נתוני ה-DataSet ואת רשומות הבן הקשורות בטבלה Departments, כפי שמוצג בתרשים 3.8.



תרשים 3.8 הצגת נתוני DataSet ורשומות בן קשורות באמצעות רכיב העיצוב DataSet.

הערה

מומלץ לעיין בקוד שנוצר על ידי רכיבי העיצוב. קוד זה דומה במידה רבה לקוד שכתבת באופן ידני בסעיפים הקודמים בפרק זה. תוכל לצפות בקוד שנוצר על ידי פתיחת תצוגת הקוד של Form1 בעורך הקודים של Visual Studio (Visual Studio code editor), והרחבת האזור שנקרא Windows Form Designer generated code.

סיכום

בפרק זה עסקנו בנושאים רבים, מכיון שהתפיסות והאובייקטים המתוארים כאן (וכן אלה שבפרק 2) מהווים את הבסיס לכל יישומי מסד הנתונים שנפתח באמצעות ADO.NET ו- Visual Basic.NET. הראינו כיצד ה-DataSet והאובייקטים הקשורים אליו, כגון: DataTable, DataRelation, DataRow ו-DataColumn, מספקים גמישות רבה ומודל תכנות עשיר לטיפול בנתונים, כאשר המערכת מנותקת ממקור הנתונים הפיסי. בפרק 4 נדגים כיצד ניתן להשתמש ב-DataAdapter למילוי ה-DataSet בנתונים, ולעדכן באופן אוטומטי את מקור הנתונים בשינויים שבוצעו בו.

שאלות ותשובות

ש: נראה לי שאני עדיין יכול לגשת לנתונים אשר נמצאים במקור הנתונים שלי בצורה ישירה (באמצעות פקודות data) או בצורה עקיפה (באמצעות disconnected). האם קיימת המלצה כללית לשימוש בגישה כלשהי בתרחיש שונה?

ת: לשימוש באמצעות DataSets יש מספר יתרונות על פני גישה ישירה למסד נתונים. גישה זו מציעה דרך פשוטה ואחידה להעברת נתונים בין שכבות שונות ומיקומים שונים ביישום מסד נתונים מבוזר ובין יישומים שונים, הודות לתמיכת XML הקיים. גישה זו מספקת מנגנון להטמנת נתונים ומאפשרת לבצע מיון, סינון וחיפוש בנתונים אלה מבלי לגשת למקור הנתונים בכל פעולה. לבסוף, גישה זו מאפשרת להשיג מספר טבלאות, גם ממקורות נתונים שונים, ולטפל בהן בנפרד או על פי ההתייחסות ביניהן.

לגישה ישירה למקור הנתונים באמצעות האובייקט Command יש גם יתרונות. ניתן לבצע פעולות מסוימות, כגון פעולות לשינוי מבנה מסד נתונים, בגישה ישירה בלבד. גם בפקודות SQL סטנדרטיות או בקריאות לשגרות מאוחסנות, הפקודות הישירות מספקות שליטה חזקה יותר על התזמון או שיטת ההפעלה שלהן, דבר שעשוי לשפר את הביצועים או המדרוגיות. לבסוף, ניתן לצמצם את תקורת הזיכרון הדרוש עבור ה-DataSet, במיוחד אם לא קיים צורך יישומי להטמין את הנתונים, כמו במקרים של יצירת דף Web או בעת מילוי תיבת רשימה.

אם כך, מתי כדאי להשתמש בגישה ישירה למסד נתונים במקום ב-DataSets? ברור שאם אתה מבצע פעולה שניתן לבצעה על ידי אובייקט Command בלבד, עליך להשתמש בשיטה זו. מצב זה כולל קריאה לשגרות מאוחסנות המבצעות פעולות ומחזירות ערך החזרה ו/או ערכי פרמטרים, כגון מבנה מסד נתונים או פעולות DDL. כמו כן, עליך להימנע מהשימוש ב-DataSets כאשר:

1. הנתונים הם לקריאה בלבד.
2. אם אתה משתמש בו לזמן קצר בלבד והטעינה והשארת ה-DataSet בזיכרון אינן משתלמות.

או

3. אם הנתונים מיועדים לשרת ואין צורך להעבירם לשכבה אחרת או אל מחשב אחר. ברוב המקרים, מומלץ להשתמש ולנצל את הגמישות המוצעת על ידי אובייקט DataSet של ADO.NET.

פרק 4

ADO.NET – DataAdapter

מה בפרק זה?

- מילוי DataSet ממקור נתונים
- עדכון מקור הנתונים

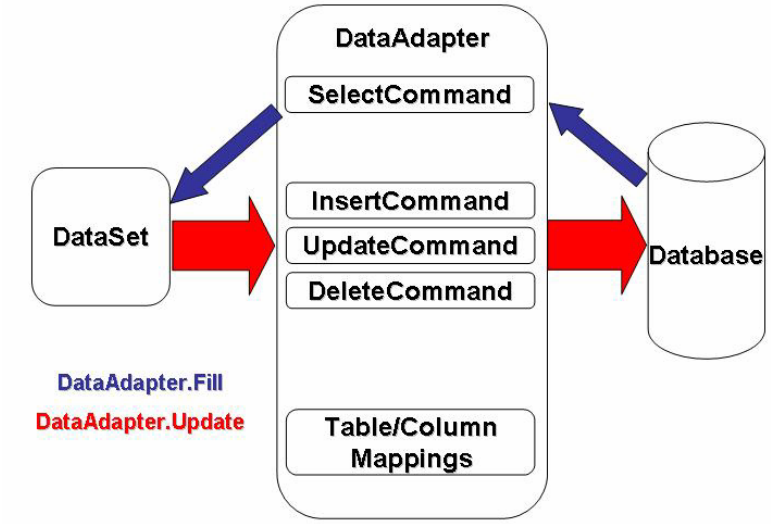
כל מה שהצגנו בפרק 3 – ובכלל זה האובייקטים DataSet, DataTable, DataRow, DataColumn, DataRelation ו-Constraint – אינם מספיקים עדיין. כעת תרצה בוודאי לעסוק במסד הנתונים עצמו. עדיין לא הדגמנו כיצד לטעון את ה-DataSet בנתונים ממסד נתונים או ממקור נתונים אחר, ולעדכן לאחר מכן את מקור הנתונים בשינויים שביצעת בנתונים שב-DataSet. האובייקט DataAdapter, או מתאם נתונים, הוא החוליה החסרה ברצף פעולות זה.

DataAdapter הוא המתאם בין ה-DataSet לבין מקור הנתונים עצמו. כלומר, זהו גם המתאם בין שני עולמות ה-ADO.NET – העולם המחובר של האובייקטים של ספקי הנתונים ב-NET.,, אשר תוארו בפרק 2 (Connection, Command ו-DataReader); לבין העולם המנותק של ה-DataSet והאובייקטים שלו, אשר תוארו בפרק 3.

מתאם הנתונים, DataAdapter, הוא האובייקט היסודי האחרון אשר על ספק נתוני NET. לתת. כל ספק נתונים במסגרת NET. חייב לספק יישום משלו עבור מתאם נתונים זה (SqlDataAdapter, OledbDataAdapter, OdbcDataAdapter וכן הלאה). הסיבה שבגללה השהינו את הצגתו עד עתה היא שללא הבנת ה-DataSet והאובייקטים שלו, לא תוכל לעשות שימוש מעשי באובייקט DataAdapter. כעת נדגים כיצד מתאם הנתונים מצרף ומשפר את כל הדברים שהצגנו עד עתה.

תרשים 4.1 מדגים את השימוש במתאם הנתונים. המשימה הבסיסית של אובייקט זה היא לנהל את תעבורת הנתונים בין מקור הנתונים לבין ה-DataSet. משימה זו מבוצעת באמצעות ערכה קטנה של תכונות ושיטות.

ב-DataAdapter כלולות שתי שיטות עיקריות. השיטה Fill ממלאת טבלה בנתונים ממקור נתונים, והשיטה Update מעדכנת מקור נתונים בשינויים שבוצעו בטבלאות DataSet.



תרשים 4.1 DataAdapter הוא החלק המתאם בין מקור הנתונים לטבלאות DataSet.

ה-DataAdapter מכיל ערכה בת ארבע תכונות (SelectCommand, InsertCommand, UpdateCommand ו-DeleteCommand), אשר מייצגות אובייקטי Command של ADO.NET המוגדרים עבור כל אחת מהפעולות המתאימות. האובייקט SelectCommand מופעל כאשר מתבצעת קריאה לשיטה Fill של ה-DataAdapter. כאשר מתבצעת קריאה לשיטה Update של המתאם, מופעלת אחת משלוש הפקודות האחרות בהתאמה עבור כל שורה, DataRow, אשר השתנתה. למפתח מסד הנתונים יש שליטה מלאה על פקודות אלו, אשר מאפשרות התאמה של הפקודות המשמשות לפעולות Insert, Update ו-Delete.

לבסוף, אוסף של מיפויי טבלאות ועמודות, DataTableMappings, מאפשר להגדיר מיפויים בין מקור נתונים לבין טבלה ושמות עמודות.

מילוי DataSet ממקור נתונים

הדבר המינימלי אשר דרוש כדי להשתמש ב-DataAdapter הוא התחברות ופקודת SELECT. ניתן ליצור, להגדיר ולהקצות אובייקטי Connection ו-Command נפרדים לתכונות Connection ו-SelectCommand של המתאם, אך נוח יותר להשתמש בבנאי של המתאם, אשר מקבל שני פרמטרי מחרוזת – אחד עבור הפקודה SELECT והשני עבור מחרוזת ההתחברות. להלן הקוד עבור משימה זו.

VB

```
Dim da As SqlDataAdapter = New SqlDataAdapter("SELECT * FROM _
tblDepartment", "server=localhost;uid=sa; database=novelty")
```

C#

```
SqlDataAdapter da = new SqlDataAdapter( "SELECT * FROM tblDepartment",
"server=localhost;uid=sa; database=novelty");
```

הערה

זכור כי פקודת SQL שאתה מציין עבור SelectCommand עשויה להכיל פרמטרים. במידת הצורך, חזור לפרק 2 כדי לרענן את זיכרונך בנושא הגדרת הפרמטרים של ספקי נתונים שונים ב-NET..

כעת נקרא לשיטה Fill, אשר תאחזר נתונים ממסד הנתונים novelty ותטען אותם אל DataSet. תחילה נוסיף קוד לטופס frmDataSets בפרויקט DataSetCode שהתחלנו בפרק 3. לשם כך, בצע את הפעולות שלהלן.

1. לחץ לחיצה ימנית על הפרויקט **DataSetCode** ב- Solution Explorer, ובתפריט הקיצור בחר **Properties**.
2. בחר בפריט **General** מהתיקיה Common Properties ולאחר מכן קבע את ערך התכונה Startup object ל-**frmDataSets**.

3. ב- Form Designer הצג את **frmDataSets**.

4. בכרטיסיה Windows Forms של תיבת הכלים, הוסף לחצן תחת הלחצן btnConstraints.

5. בחלון Properties, קבע את ערך התכונה Name של הלחצן ל-**btnDataAdapterFill** ואת ערך התכונה Text ל- **DataAdapter Fill**.

6. להלן נשתמש בספק הנתונים SqlClient, ולכן נוסיף קוד לאחר פקודות הייבוא (import) הקיימות בראש הצהרת imports עבור namespace זה (מוצג בהדגשה):

```
Imports System
Imports System.Data
```

VB

```
Imports System.Data.SqlClient
```

C#

```
Using System.Data.SqlClient;
```

7. הוסף את קוד 4.1 ל-frmDataSets (לאחר שתלחץ על הלחצן שהוספת עכשיו).

קוד 4.1-VB מילוי DataSet בשם dsEmployeeInfo באמצעות SqlDataAdapter

```
Private Sub btnDataAdapterFill_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDataAdapterFill.Click
        ReadData()
        'ReadDataMapped()
    End Sub

    Private Sub ReadData()
        Dim daDepartments As SqlDataAdapter = New SqlDataAdapter( _
                "SELECT * FROM tblDepartment", _
                "server=localhost;uid=sa;database=novelty")

        dsEmployeeInfo = New DataSet()
        daDepartments.Fill(dsEmployeeInfo, "Departments")

        DisplayDataSet(dsEmployeeInfo)
    End Sub
```

קוד #C-4.1 מילוי DataSet בשם dsEmployeeInfo באמצעות SqlDataAdapter

```
private void btnDataAdapterFill_Click(object sender, System.EventArgs e)
        {
          ReadData();
        }
        private void ReadData()
        {
         SqlDataAdapter daDepartments = new SqlDataAdapter(
            "SELECT * FROM tblDepartment",
            "server=localhost;uid=sa;database=novelty");

          dsEmployeeInfo = new DataSet();

           daDepartments.Fill(dsEmployeeInfo, "Departments");

            DisplayDataSet(dsEmployeeInfo);
        }
```

לאחר שתיצור את מתאם הנתונים daDepartments באמצעות הפקודה SELECT, ואת מחרוזת ההתחברות, תוכל לקרוא לשיטה Fill כדי למלא את הטבלה Departments שב-DataSet בשם dsEmployeeInfo. השיטה Fill מחזירה גם את מספר השורות שהתווספו (או רוענו) ל-DataSet. השלבים הבאים מבוצעים בצורה עקיפה על ידי ה-DataAdapter כדי להפעיל את השיטה Fill:

1. פתיחת ההתחברות של SelectCommand, אם אינה פתוחה.
2. הפעלת הפקודה הרשומה בתכונה CommandText של SelectCommand (ושימוש בפרמטרים, אם קיימים).
3. יצירת DataReader אשר מחזיר את שמות וסוגי העמודות המשמשים ליצירת טבלה חדשה ב-DataSet שרשום, אם אינה קיימת.
4. אחזור הנתונים ומילוי הטבלה באמצעות שימוש ב-DataReader.
5. סגירת ה-DataReader.
6. סגירת ההתחברות, אם נפתחה על ידי ה-DataAdapter. אם ההתחברות היתה פתוחה עוד לפני כן, היא תישאר פתוחה.

הערה

כאשר אתה מפעיל פקודה אחת על מקור נתונים, יהיה פשוט ויעיל יותר לאפשר ל-DataAdapter ליצור ולנהל את האובייקטים Command ו-Connection באופן פנימי, על ידי כך שתספק את מחרוזת ההתחברות ומשפט ה-SELECT בזמן יצירת

ה-DataAdapter. עם זאת, אם בכוונתך להפעיל מספר פקודות על אותו מסד נתונים, יהיה יעיל יותר ליצור ולפתוח אובייקט Connection ולאחר מכן להעביר אותו אל המתאם. פעולה זו תשמור על ההתחברות פתוחה במקום לפתוח ולסגור אותה פעמים רבות, דבר שיביא לשיפור ניכר בביצועים. אם כך, הקוד המתאים יהיה:

```
Private Sub ReadData()
    Dim rows as Integer
    Dim daDepartments As New SqlDataAdapter()
    Dim conn As New SqlConnection( _
            "server=localhost;uid=sa;database=novelty")
    Dim cmdSelect As New SqlCommand( _
        "SELECT * FROM tblDepartment")

    dsEmployeeInfo = New DataSet()

    cmdSelect.Connection = conn
    daDepartments.SelectCommand = cmdSelect
    ' Open the connection before starting operations
    conn.Open()

    rows = daDepartments.Fill(dsEmployeeInfo, "Departments")
    ' Do other database operations here
    ' ...
    DisplayDataSet(dsEmployeeInfo)

    ' close the connection
    conn.Close()
End Sub
```

כדי להפעיל את הקוד, עליך להחליף בתוך btnDataAdapterFill_Click את הקריאה לשיטה ReadData בקריאה לשיטה הרשומה להלן:

VB

```
Private Sub ReadDataNoInternals()
    Dim daDepartments As New SqlDataAdapter
    Dim conn As New SqlConnection( _
      "server=localhost;uid=sa;database=novelty")
    Dim cmdSelect As New SqlCommand( _
        "SELECT * FROM tblDepartment")

    dsEmployeeInfo = New DataSet

    cmdSelect.Connection = conn
    daDepartments.SelectCommand = cmdSelect
```

```
        ' Open the connection before starting operations
        conn.Open()

        daDepartments.Fill(dsEmployeeInfo, "Departments")
        ' Do other database operations here
        ' ...

        DisplayDataSet(dsEmployeeInfo)
        conn.Close()
    End Sub
```

C#

```
private void ReadDataNoInternals()
        {
          SqlDataAdapter daDepartments = null;
          SqlConnection conn =
          new SqlConnection("server=localhost;uid=sa;database=novelty");
          SqlCommand cmdSelect =
                    new SqlCommand("SELECT * FROM tblDepartment");

          dsEmployeeInfo = new DataSet();

          cmdSelect.Connection = conn;
          daDepartments.SelectCommand = cmdSelect;

          // Open the connection before starting operations
          conn.Open();

          daDepartments.Fill(dsEmployeeInfo, "Departments");
          // Do other database operations here
          //

          DisplayDataSet(dsEmployeeInfo);
          conn.Close();
        }
```

כדי שהדבר באמת ישתלם, עליך לבצע פעולות נוספות במסד הנתונים, באמצעות שימוש באותה מחרוזת ההתחברות.

העברנו לשיטה Fill קישור אל ה-DataSet ואת שם הטבלה שברצוננו למלא. נוכל גם להעביר קישור אל הטבלה במקום להעביר שם. אפשרות נוספת היא לציין את ה-DataSet בלבד ולאחר מכן את ברירות המחדל של השיטה Fill, כדי לטעון את הנתונים אל הטבלה Table.

הערה

בדרך כלל תעדיף להשתמש במתאם הנתונים כדי למלא טבלה שב-DataSet, אך קיימת אפשרות לטעון נתונים לאובייקט DataTable נפרד באמצעות השיטה Fill.

אם נרצה לטעון טבלה שנייה, נוכל להוסיף DataAdapter שני עם פקודה SELECT אחרת. כדי לטעון את הטבלאות Department ו-Employees מטבלאות מסדי הנתונים המתאימות שלהן, עלינו להחליף את השגרה ReadData שבקוד 4.1 בקוד שלהלן:

VB

```
Private Sub ReadData()
    Dim daDepartments As SqlDataAdapter = New SqlDataAdapter( _
        "SELECT * FROM tblDepartment", _
        "server=localhost;uid=sa;database=novelty")

    Dim daEmployees As SqlDataAdapter = New SqlDataAdapter( _
        "SELECT * FROM tblEmployee", _
        "server=localhost;uid=sa;database=novelty")

    dsEmployeeInfo = New DataSet()

    daDepartments.Fill(dsEmployeeInfo, "Departments")
    daEmployees.Fill(dsEmployeeInfo, "Employees")

    DisplayDataSet(dsEmployeeInfo)
  End Sub
```

C#

```
private void ReadData()
    {
      SqlDataAdapter daDepartments = new SqlDataAdapter(
         "SELECT * FROM tblDepartment",
         "server=localhost;uid=sa;database=novelty");

       SqlDataAdapter daEmployees = new SqlDataAdapter(
         "SELECT * FROM tblEmployee",
         "server=localhost;uid=sa;database=novelty");

      dsEmployeeInfo = new DataSet();

      daDepartments.Fill(dsEmployeeInfo, "Departments");
      daEmployees.Fill(dsEmployeeInfo, "Employees");
```

```
        DisplayDataSet(dsEmployeeInfo);
    }
```

הפעלת הפרויקט DataSetCode ולחיצה על DataAdapterFill תמלא את תיבת הרשימה בתוצאות הדומות לאלו שהושגו קודם לכן. ההבדל הוא שהנתונים אשר נטענים כעת ל-DataSet ומוצגים בתיבת הרשימה, מגיעים ממסד הנתונים של SQL Server ולא נוצרים באופן מקומי על ידי קוד.

גם במקרה זה נוכל ליצור DataRelation (קישורי נתונים) בין שתי הטבלאות, כפי שעשינו בפרק הקודם.

לא תמיד צריך להשתמש ב-DataAdapter שונה עבור כל טבלה ב-DataSet. ניתן גם לעשות שימוש חוזר באותו מתאם על ידי שינוי הפקודות שהוא משתמש בהן. גישה זו שימושית בעיקר כאשר יש קריאות אחדות לצורך מילוי טבלאות. יצירה ועדכון של כל הפקודות (הכנסה, עדכון ומחיקה) לכל DataAdapter דורשות יותר עבודת תכנות.

לכן, נוכל לשכתב את הקוד הקודם ולהשתמש ב-DataAdapter יחיד, כמפורט להלן (שים לב: בקוד המצורף עליך לשנות את השיטה btnDataAdapterFill_Click, באופן שתקרא ל- ReadData2 במקום הקריאה ל-ReadData):

VB

```
Private Sub ReadData2()

    Dim da As SqlDataAdapter = New SqlDataAdapter( _
        "SELECT * FROM tblEmployee", _
        "server=localhost;uid=sa;database=novelty")

    dsEmployeeInfo = New DataSet

    da.Fill(dsEmployeeInfo, "Employees")

    da.SelectCommand.CommandText = _
      "SELECT * FROM tblDepartment"
    da.Fill(dsEmployeeInfo, "Departments")

    DisplayDataSet(dsEmployeeInfo)
  End Sub
```

C#

```
private void ReadData2()
        {
          SqlDataAdapter da = new SqlDataAdapter(
                    "SELECT * FROM tblEmployee",
                    "server=localhost;uid=sa;database=novelty");

          dsEmployeeInfo = new DataSet();

          da.Fill(dsEmployeeInfo, "Employees");

          da.SelectCommand.CommandText =
                    "SELECT * FROM tblDepartment";
          da.Fill(dsEmployeeInfo, "Departments");

          DisplayDataSet(dsEmployeeInfo);
        }
```

שים לב: בקוד המצורף עליך לשנות את השיטה btnDataAdapterFill_Click, באופן שתקרא ל-ReadData2 במקום ל-ReadData.

הערה

ניתן, ולפעמים עדיף, למלא טבלת DataSet בתוצאות השימוש ב- SQL Join לקישור בין שתי טבלאות. לאחר מכן תהיה לנו טבלת DataSet יחידה ולא יהיה צורך ליצור כל קישור. עם זאת, השימוש בטבלאות נפרדות המקושרות באמצעות DataRelation גמיש יותר ברוב המקרים. דבר זה נכון במיוחד כאשר מעדכנים את מקור הנתונים, מכיון שיש הגבלות על עדכון טבלאות מקושרות (joined tables). זכור שמגבלות אלו אינן קיימות כאשר הטבלאות מעודכנות בנפרד.

הערה

דרך יעילה יותר לטעינת שתי טבלאות ב-DataSet היא לספק את השיטה SelectCommand, אשר קוראת לשגרה מאוחסנת שמחזירה מספר ערכות תוצאות או שמפעילה אצוה (batch) של פקודות SQL. פעולה זו גורמת לאחזור כל הנתונים על ידי השרת במהלך אחד, במקום מספר מהלכים אשר דרושים בקוד המוצג. עם זאת, אחזור מספר טבלאות בשיטה זו נעשה בצורה ישירה, ולכן עדכון מקור הנתונים בשינויים שבוצעו בטבלאות DataSet יהיה מורכב יותר במידה מסוימת, אם קיימים קשרים בין הטבלאות. בתסריט עסקי 4.1, אשר מופיע בהמשך פרק זה, נסביר כיצד לעדכן טבלאות כאלו.

קוד 4.2 מדגים כיצד למזג מספר Fills אל טבלה יחידה.

בקוד המצורף, עליך לשנות את השיטה btnDataAdapterFill_Click, כך שתקרא ל- ReadData2 במקום ל-ReadData3.

קוד 4.2-VB ביצוע מספר פעולות Fill בטבלה יחידה באמצעות SqlDataAdapter יחיד

```
Private Sub ReadData3()
    Dim daEmployees As SqlDataAdapter = New SqlDataAdapter( _
        "SELECT * FROM tblEmployee WHERE DepartmentID = 1", _
        "server=localhost;uid=sa;database=novelty")

    dsEmployeeInfo = New DataSet

    daEmployees.Fill(dsEmployeeInfo, "Employees")

    daEmployees.SelectCommand.CommandText = _
      "SELECT * FROM tblEmployee WHERE DepartmentID = 4"
    daEmployees.Fill(dsEmployeeInfo, "Employees")

    DisplayDataSet(dsEmployeeInfo)
  End Sub
```

קוד #4.2-C ביצוע מספר פעולות Fill בטבלה יחידה באמצעות SqlDataAdapter יחיד

```
private void ReadData3()
    {
      SqlDataAdapter daEmployees = new SqlDataAdapter(
            "SELECT * FROM tblEmployee WHERE DepartmentID = 1",
            "server=localhost;uid=sa;database=novelty");

      dsEmployeeInfo = new DataSet();
      daEmployees.Fill(dsEmployeeInfo, "Employees");

      // Change where clause from DepartmentID = 1 to
      // DepartmentID = 4
      daEmployees.SelectCommand.CommandText =
            "SELECT * FROM tblEmployee WHERE DepartmentID =4";
      daEmployees.Fill(dsEmployeeInfo, "Employees");

      DisplayDataSet(dsEmployeeInfo);
    }
```

נוכל להשתמש בגישה המוצגת בקוד 4.2 ולהמשיך בכיוון זה על ידי הפעלה של אותה הצהרת SELECT מספר פעמים, כדי לרענן את הטבלה בנתונים העדכניים ביותר (אשר אולי השתנו על ידי משתמש שונה) הנמצאים במקור הנתונים.

הערה

ערכים קיימים בטבלה מתעדכנים רק בפעולת Fill אשר תבוא לאחר מכן, אם מוגדר מפתח ראשי עבור הטבלה. פעולת ברירת המחדל של השיטה Fill היא מילוי הטבלה בנתוני סכמת עמודות ורשומות נתונים, ללא הגדרת אילוצים אשר עשויים להיות מוגדרים במקור הנתונים. כדי להגדיר את התכונה PrimaryKey בצורה נכונה – באופן שיופעלו ריענונים מסוג זה (וגם השיטה Find) – יש לבצע את אחת הפעולות הבאות לפני קריאה לשיטה Fill:

- קריאה לשיטה FillSchema של ה-DataAdapter.
- קביעת ערך התכונה MissingSchema של ה-DataAdapter ל-AddWithKey.
- קביעה מפורשת של התכונה PrimaryKey כעמודה או כעמודות המתאימות, אם עמודה או עמודות אלו ידועות בזמן העיצוב.

עדכון מקור הנתונים

לאחר ביצוע כל השינויים הרצויים בטבלאות DataSet, תרצה בדרך כלל לשמור את השינויים האלה במקור הנתונים. לשם כך, קרא לשיטה Update של ה-DataAdapter. כאשר תקרא לשיטה Update, ה-DataAdapter ינתח את השינויים שבוצעו בטבלה הרצויה ב-DataSet (או בכל הטבלאות אם שם הטבלה לא נרשם). עבור כל רשומה שהשתנתה, מבוצעת הפקודה המתאימה (Insert, Update או Delete) על מקור הנתונים, כדי לעדכן אותו בערכי הנתונים הנוכחיים. פקודות אלו מוצגות באמצעות התכונות InsertCommand, UpdateCommand ו-DeleteCommand.

הערה

היכולת לציין בקלות פקודות SQL מותאמות או להשתמש בשגרות מאוחסנות בצורה אוטומטית בזמן עדכון מקור נתונים מ-DataSet, היא שיפור **משמעותי** בהשוואה לשיטות הזמינות ב- ADO 2.X. שים לב שב-ADO.NET ניתן לשנות את אופן העדכון של רשומות שהשתנו, וגם תוכל להשתמש בשגרות מאוחסנות במשימה זו. פעולה זו משפרת את הביצועים ואת הלוגיקה (העסקית) המותאמת, בנוסף ליכולת לציין פקודות SQL לפעולות העדכון. מייד נציג דוגמה לגישה זו.

בנוסף לכך, מנגנון העדכון באצווה פועל גם במקורות נתונים שאינם SQL, שלא כמו ב- ADO 2.X שבו עדכון האצווה פעל במקור נתונים מבוסס SQL בלבד.

כל רשומה שהשתנתה מעודכנת בנפרד ואינה חלק של תנועה או של אצווה. בנוסף, סדר העיבוד של השורות נקבע על ידי הסדר שלהן בטבלה.

כדי לשלוט בצורה מפורשת בסדר הפעולות בטבלה מסוימת, נוכל להשתמש בשיטה GetChanges או בשיטה Select. שתי השיטות זמינות ברמת ה-DataSet או ברמת הטבלה. נשתמש בשיטות אלו כדי להחזיר קבוצות נפרדות של רשומות, שכל אחת מהן מתאימה למצב רשומה שונה.

נניח שברצוננו להשתמש ב-daDepartments כדי לעדכן את מסד הנתונים novelty מ-DataSet בשם dsEmployeeInfo. על פי הדרישות שלנו, עלינו לבצע תחילה את כל ההוספות, לאחר מכן את כל העדכונים ולבסוף – את כל המחיקות. נוכל לעשות זאת על ידי קריאה לשיטה GetChanges שלוש פעמים ולציין מצב רשומה שונה בכל פעם. לאחר כל קריאה ל-GetChanges, נקרא לשיטה Update של המתאם ונעביר לה את הטבלה שהוחזרה על ידי השיטה GetChanges.

VB

```
dt = dsEmployeeInfo.Tables("Departments")
' Get each type of change and update accordingly
dtChanged = dt.getChanges(DataRowState.Added)
daDepartments.Update(dtChanged)

dtChanged = dt.getChanges(DataRowState.Modified)
daDepartments.Update(dtChanged)

dtChanged = dt.getChanges(DataRowState.Deleted)
daDepartments.Update(dtChanged)
```

C#

```
dt = dsEmployeeInfo.Tables["Departments"];

// Get each type of change and update accordingly
dtChanged = dt.getChanges(DataRowState.Added);
daDepartments.Update(dtChanged);

dtChanged = dt.getChanges(DataRowState.Modified);
daDepartments.Update(dtChanged);

dtChanged = dt.getChanges(DataRowState.Deleted);
daDepartments.Update(dtChanged);
```

נוכל לכתוב קוד זה בצורה קומפקטית יותר. הנה כך:

VB

```
daDepartments.Update(dt.GetChanges(DataRowState.Added))
daDepartments.Update(dt.GetChanges(DataRowState. Modified))
daDepartments.Update(dt.GetChanges(DataRowState. Deleted))
```

C#

```
daDepartments.Update(dt.GetChanges(DataRowState.Added));
daDepartments.Update(dt.GetChanges(DataRowState. Modified));
daDepartments.Update(dt.GetChanges(DataRowState. Deleted));
```

נוכל להגיע לאותן התוצאות על ידי שימוש בשיטה Select:

VB

```
dt = dsEmployeeInfo.Tables("Departments")
da.Update(dt.Select(Nothing , Nothing , _
    DataViewRowState.Added))
da.Update(dt.Select(Nothing , Nothing , _
    DataViewRowState. Modified))
da.Update(dt.Select(Nothing , Nothing , _
    DataViewRowState. Deleted))

daDepartments.Update(dt.GetChanges(DataRowState. Modified))
daDepartments.Update(dt.GetChanges(DataRowState. Deleted))
```

C#

```
dt = dsEmployeeInfo.Tables["Departments"];
da.Update(dt.Select(null , null,
    DataViewRowState.Added));
da.Update(dt.Select(null , null , DataViewRowState. ModifiedCurrent));
da.Update(dt.Select(null , null , DataViewRowState. Deleted));
```

היתרון של השימוש בשיטה Select ולא בשיטה GetChanges הוא בכך ששיטה זו מאפשרת סינון ומיון נוספים (אם דרוש).

הסרת רשומה לעומת מחיקת רשומה

כעת כדאי להזכיר מהו ההבדל בין הסרת רשומה לבין מחיקת רשומה מטבלה, כפי שתואר בפרק 3. כאשר אתה **מסיר** (Remove) רשומה, מוציאים אותה מאוסף הרשומות של הטבלה והיא אינה קיימת יותר. כאשר אתה **מוחק** (Delete) רשומה, הפעולה היא סימון מחיקה של הרשומה ולא הוצאתה מהטבלה. לכן, כאשר אתה

משתמש בטבלה יחד עם מתאם לעדכון מקור נתונים, עליך להשתמש תמיד בשיטה Delete ולא בשיטה Remove כדי להסיר שורה. כאשר המתאם מגלה רשומה שסומנה למחיקה, הוא יידע להפעיל את DeleteCommand על מסד הנתונים כדי לסנכרן אותו עם הטבלה. לעומת זאת, אם תשתמש בשיטה Remove, המתאם לא יראה אף פעם את הרשומה שהוסרה כאשר מתבצעת קריאת Update, והרשומה לא תימחק לעולם ממקור הנתונים.

הגדרת פקודות Update

מתאם הנתונים אינו יוצר באופן אוטומטי את פקודות SQL, אשר דרושות לעדכון מקור הנתונים בשינויים שבוצעו בנתוני ה-DataSet. אם הפקודות Insert, Update או Delete לא נקבעו והשיטה Update קוראת להן, תיווצר שגיאת חריגה – exception. תוכל לציין פקודות אלו במספר דרכים:

- השתמש באובייקט CommandBuilder כדי ליצור באופן אוטומטי פקודות בזמן ההפעלה.
- תכנת בצורה מפורשת את הפקודות בקוד.
- השתמש ברכיב העיצוב DataAdapter (DataAdapter Design-Time Component) וב- DataAdapter Configuration Wizard.

שימוש באובייקט CommandBuilder

גישה זו היא הפשוטה ביותר, אך במידה מסוימת מוגבלת; היא דומה ל-BatchUpdate של ADO 2.X. האובייקט CommandBuilder ייצור באופן אוטומטי את התכונות InsertCommand, UpdateCommand ו-DeleteCommand, אשר דרושים למתאם נתונים זה על ידי קישור אובייקט CommandBuilder לאובייקט DataAdapter מסוים. אם אחת מתכונות אלו ריקה (Nothing ב- Visual Basic), קיים כבר אובייקט פקודה עבור תכונה זו. CommandBuilder לא ידרוס פקודה קיימת.

הערה

כצפוי, אובייקט DataAdapter צריך להיות ספציפי לספק נתוני NET. איתו הוא עובד. לכן, עלינו להשתמש באובייקט נגזר ספציפי בקוד שלנו, כגון SqlDataAdapter, OleDataAdapter או OdbcDataAdapter.

כדי שיצירת הפקודות האוטומטיות תתבצע בהצלחה, יש להגדיר את SelectCommand של מתאם הנתונים. CommandBuilder משתמש בסכמת הטבלה אשר הושגה באמצעות הפקודה Select של SelectCommand ליצירת הפקודות Insert, Update ו-Delete מתאימות. שים לב לכך שהעמודות המוחזרות על ידי SelectCommand חייבות לכלול לפחות מפתח ראשי אחד או עמודה ייחודית.

שינוי ה-CommandText של הפקודה SELECT, לאחר יצירה אוטומטית של פקודות עדכון, עלול לגרום לשגיאות חריגה (exceptions), כאשר אחת מפקודות העדכון מופעלת. אם הפקודה SELECT המקורית, שפעולת יצירת הפקודה התבססה עליה, הכילה עמודות שאינן קיימות בפקודה שהשתנתה, הפעלת אחת מפקודות העדכון על ידי השיטה Update של מתאם הנתונים עשויה לגרום לגישה אל עמודות שאינן קיימות ולשגיאה. כדי להימנע מבעיה זו, עליך לקרוא לשיטה RefrechSchema של CommandBuilder לאחר ששינית את התכונה SelectCommand של מתאם הנתונים, או לאחר ששינית את ה-CommandText של אובייקט פקודה זה.

הערה

גם לאחר ש-CommandBuilder יצר פקודות Insert, Update ו/או Delete, התכונות המתאימות של ה-DataADapter אינן משתנות. CommandBuilder שומר על אובייקטי הפקודה הנוצרים באופן פנימי. תוכל לקבל הפניות לאובייקטים אלה על ידי קריאה לשיטות GetInsertCommand, GetUpdateCommand או GetDeleteCommand של CommandBuilder.

קל ופשוט להשתמש ב-CommandBuilder, אך חשוב להבין את מגבלותיו, אשר העיקרית ביניהן היא שאין שליטה על פעולתו. אובייקט זה יוצר את פקודות העדכון על פי פקודת SELECT שסופקה, ללא אפשרויות אחרות. האובייקט מיועד ליצור פקודות לטבלת מסד נתונים יחידה ונפרדת. במילים אחרות, לא ניתן להשתמש בטבלאות הממולאות בתוצאות פעולת JOIN של SQL יחד עם CommandBuilder. בנוסף לכך, הפקודות נוצרות מבלי להתחשב בכך שעשוי להיות קשר מטבלה אל טבלה אחות במסד הנתונים, דבר שעלול לגרום לפגיעה בכללי רישום מפתח זר בעת ביצוע פעולות עדכון באחד ממסדי הנתונים.

מגבלה נוספת של CommandBuilder היא שהוא אינו יוצר את פקודות העדכון, כאשר שם טבלה או שם עמודה כלשהם כוללים סימנים מיוחדים כגון רווח, נקודה או סימן אחר שאינו אלפאנומרי – גם אם השם תחום בסוגריים. עם זאת, קיימת תמיכה בשמות טבלה מלאים (כגון, database.owner.table).

לאחזור ושמירת נתונים במסד הנתונים באמצעות מתאם נתונים, הוסף טופס לפרויקט DataSetCode.

1. תחילה, הוסף לחצן מתחת ללחצן btnDataAdapterFill. קבע את ערך התכונה Name של הלחצן ל-**btnDataAdapterUpdates** ואת ערך התכונה Text ל- **DataAdapter Updates**.
2. הוסף טופס חדש **frmUpdates** לפרויקט DataSetCode.
3. בחלון Properties של frmUpdates, קבע את ערך התכונה Text ל- **DataAdapter Updates**.
4. הגדל את frmUpdates.

5. מהכרטיסיה Windows Forms של תיבת הכלים, הוסף DataGrid ל-frmUpdates והצב אותו בצד ימין של הטופס.
6. בחלון Properties, קבע את ערך התכונה Name של ה-DataGrid ל-**grdDataGrid**.
7. הגדל את DataGrid באופן שיכסה כ-80% משטח הטופס.
8. הוסף לחצן מהכרטיסיה Windows Forms של תיבת הכלים בחלק השמאלי העליון של frmUpdates.
9. בחלון Properties, קבע את ערך התכונה Name של הלחצן החדש ל-**btnLoad** ואת ערך התכונה Text ל-**Load**.
10. הוסף לחצן מתחת ללחצן Load.
11. בחלון Properties, קבע את ערך התכונה Name של הלחצן החדש ל-**btnUpdate**, את ערך התכונה Text ל-**Update** ואת ערך התכונה Enabled ל-**False**.
12. פתח את הטופס frmUpdates ב- Code View והוסף את שורות הקוד שלהלן לתחילת הקובץ:

```
Imports System
Imports System.Data
Imports System.Data.SqlClient
```

13. הוסף את קוד 4.3 ל-frmUpdates.

קוד 4.3-VB יצירת פקודות עדכון אוטומטיות באמצעות SqlCommandBuilder עבור SqlDataAdapter

```
Private dsEmployeeInfo As DataSet
Private daEmployees As SqlDataAdapter
Private conn As New SqlConnection( _
    "server=localhost;uid=sa;pwd=;database=novelty")

' Use SqlCommandBuilder to automatically generate the update commands
Private cbEmployees As SqlCommandBuilder

Private Sub btnLoad_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnLoad.Click
    dsEmployeeInfo = New DataSet()

    LoadCommandBuilder()

    'Config grid
    grdDataGrid.PreferredColumnWidth = 110
    grdDataGrid.AllowSorting = True
```

```
        'Fill Data Set
        'daEmployees.Fill(dsEmployeeInfo, "Employees")
        SqlDataAdapter1.Fill(dsEmployeeInfo, "Employees")

        'Assign DataSet to DataGrid
        grdDataGrid.DataSource = dsEmployeeInfo.Tables("Employees")

        btnUpdate.Enabled = True
    End Sub

    Private Sub btnUpdate_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles btnUpdate.Click
        Try
            daEmployees.Update(dsEmployeeInfo, "Employees")
            'SqlDataAdapter1.Update(dsEmployeeInfo, "Employees")

        Catch ec As DBConcurrencyException
            ' Do something !!
        Catch es As SqlException
            MessageBox.Show(es.Message)

        End Try
    End Sub

    Private Sub LoadCommandBuilder()
        If conn.State = ConnectionState.Closed Then
            conn.Open()
        End If

        'Create New DataAdapter Object
        Dim SQL As String
        SQL = "SELECT FirstName, LastName, DepartmentID, Salary, ID FROM
tblEmployee"
        daEmployees = New SqlDataAdapter(SQL, conn)

        ' Use SqlCommandBuilder to automatically
        ' generate the update commands
        cbEmployees = New SqlCommandBuilder(daEmployees)

    End Sub
```

קוד #C-4.3 יצירת פקודות עדכון אוטומטיות באמצעות SqlCommandBuilder עבור SqlDataAdapter

```
private DataSet dsEmployeeInfo;
private SqlDataAdapter daEmployees;
private SqlConnection conn = new SqlConnection
  ("server=localhost;uid=sa;pwd=;database=novelty");
private SqlCommandBuilder cbEmployees;
private System.Windows.Forms.DataGrid grdDataGrid;
private void btnLoad_Click(object sender, System.EventArgs e)
{
  dsEmployeeInfo = new DataSet();

  LoadCommandBuilder();

  // Config grid
  grdDataGrid.PreferredColumnWidth = 110;
  grdDataGrid.AllowSorting = true;

  //Fill Data Set
  daEmployees.Fill(dsEmployeeInfo, "Employees");

  // Assign DataSet to DataGrid
  grdDataGrid.DataSource =
          dsEmployeeInfo.Tables["Employees"];
  btnUpdate.Enabled = true;

}

private void LoadCommandBuilder()
{
  if (conn.State == ConnectionState.Closed)
  {
          conn.Open();
  }

  String sql;
  sql = "SELECT FirstName, LastName, DepartmentID, Salary, ID"
        + " FROM "
        + "          tblEmployee";
  daEmployees = new SqlDataAdapter(sql,conn);
```

```
    // Use SqlCommandBuilder to automatically
    // generate the update command
    cbEmployees = new SqlCommandBuilder(daEmployees);
  }
```

הקריאה לשגרה הראשית LoadCommandBuilder מתבצעת כאשר הלחצן Load נלחץ. שגרה זו מדגימה כיצד לפתוח אובייקט התחברות בצורה מפורשת (וכיצד להימנע משגיאה בלחיצה על Load מספר פעמים), וגם – כיצד להגדיר את האובייקטים SqlDataAdapter (dsEmployees) ו-SqlCommandBuilder (cbEmployees). שני אובייקטים אלה נוצרים ומאותחלים עם טפסים בבנאים שלהם, אשר מקבלים את ההגדרות החיוניות כפרמטרים. הבנאי של daEmployees מקבל את מחרוזת SELECT ואת מחרוזת ההתחברות, ואילו cbEmployees מקבל את האובייקט SqlDataAdapter.

הערה

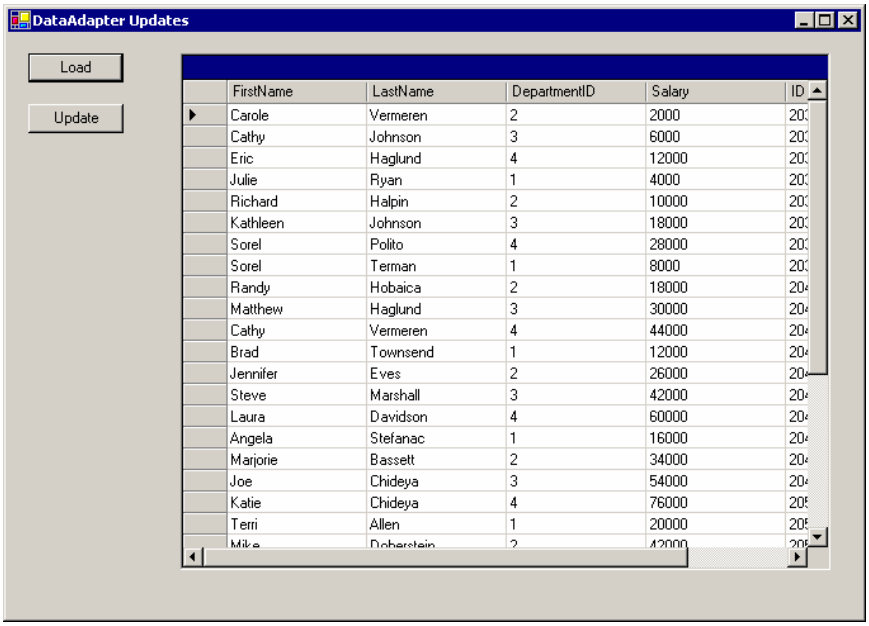
ניתן לקשר זה אל זה אובייקט DataAdapter או CommandBuilder אחד בלבד בכל זמן נתון.

כעת, נותר לנו רק להגדיר את הרשת (grid), לקבוע את ערך התכונה DataSource של הרשת כטבלה Employees ב-DataSet ולקרוא לשיטה Fill אשר תטען את ה-DataSet ותגרום לרשת להציג את הנתונים באופן אוטומטי.

המטפל באירועים (event handler) עבור לחיצה על הלחצן Update מכיל שורת קוד אחת, אשר קוראת לשיטה Update של מתאם הנתונים daEmployees.

הפעל את הפרויקט DataSetCode ולחץ על **DataAdapter Updates** בטופס frmDataSets. כאשר יוצג הטופס frmUpdates, לחץ **Load**. פעולה זו תגרום לנתונים להיקרא מטבלת מסד הנתונים tblEmployee, להיטען לטבלה Employee שב-DataSet בשם dsEmployeeInfo ולהופיע ברשת – כפי שמוצג בתרשים 4.2.

כעת תוכל לבדוק את הטופס בעצמך. בצע שינויים כרצונך, הוסף רשומות על ידי גלילה מטה אל השורה האחרונה בטבלה המתקבלת (grid), מחק שורות על ידי בחירה של שורה אחת או יותר ולאחר מכן הקש על **Delete**, או שנה ערכי עמודה על ידי עריכת ערכים בתוך תאי הטבלה. זכור ששינויים אלה לא יישמרו במסד הנתונים עד שתלחץ Update. תוכל לבדוק את השינויים על ידי שימוש בכלי המועדף עליך לצפייה בטבלאות מסדי נתונים, או על ידי לחיצה על Load אשר תגרום לטעינה חוזרת של נתוני מסד הנתונים ל-DataSet ולטבלה של הטופס.



תרשים 4.2 נתונים מטבלת הנתונים tblEmployee מוצגים בטבלת התוצאה DataGrid.

הערה

כדי להשתמש ב-CommandBuilder ליצירת פקודות עדכון, עליך לכתוב כמות קטנה של קוד, אך יש לכך חיסרון רב, גם אם מגבלותיו אינן מהוות קושי מיוחד. CommandBuilder צריך לבצע מהלך נוסף של תקשורת עם שרת מסד הנתונים כדי לאחזר את המטא-נתונים הדרושים ליצירת הפקודות. יכולת זו שימושית וגמישה למדי כאשר אתה מפתח שאילתות במהלך עבודתך, אך אם השאילתות כבר ידועות בזמן העיצוב, תוכל לשפר את הביצועים במידה ניכרת על ידי ציון מפורש של הפקודות והפרמטרים שלהן בקוד עצמו, באמצעות שימוש בפקודות עדכון מפורשות או באשף DataAdapter Configuration Wizard.

פקודות עדכון מפורשות (explicit)

אם השימוש ב-CommandBuilder הוא פשוט למדי ביצירת פקודות העדכון הדרושות, הרי שתכנות מפורש של פקודות אלו בקוד הינו גמיש למדי – אך מחייב השקעת עבודה רבה יותר. יש לתכנן ולקודד ידנית את כל אחת מארבע הפקודות (SELECT, INSERT, UPDATE ו-DELETE). ברוב המקרים, לאחר התחלת הקידוד בצורה מפורשת, תמשיך לכתוב שגרות מאוחסנות של SQL Server עבור כל אחת מהן.

קוד 4.4 מציג את תסריט SQL Server ליצירת ארבע השגרות המאוחסנות. השגרה המאוחסנת Select Employee בוחרת את כל העמודות מהטבלה tblEmployee. השגרה המאוחסנת Insert Employee מצפה לארבעה פרמטרים – פרמטר אחד עבור כל אחת

מהעמודות הניתנות לעדכון. העמודה ID אינה ניתנת לעדכון מכיון שהיא עמודת זיהוי. השגרה המאוחסנת UpdateEmployeee מצפה לאותם ארבעה פרמטרים בעמודות הניתנות לעדכון, וגם לפרמטר חמישי המכיל את הערך המקורי של העמודה ID. ערך זה של העמודה ID משמש במשפט WHERE לבחירה נכונה של הרשומה לעדכון (על פי המפתח הראשי). השגרה המאוחסנת DeleteEmployee דורשת את הערך המקורי של העמודה ID בלבד, כפרמטר לבחירת השורה הנכונה למחיקה.

התסריט לביצוע הפעולה נמצא בקובץ crtEmployeePRO.sql שבספריה DatabaseScripts השייכת לפרק זה.

קוד 4.4 תסריט SQL Server ליצירת שגרות מאוחסנות עבור הטבלה tblEmployee

```
use [Novelty]
GO

IF EXISTS (SELECT * FROM sysobjects WHERE name =
    'SelectEmployees' AND user_name(uid) = 'dbo')
   DROP PROCEDURE [dbo].[SelectEmployees];
GO
CREATE PROCEDURE [dbo].[SelectEmployees]
AS
  SET NOCOUNT ON;
SELECT FirstName, LastName, DepartmentID, Salary, ID FROM
  tblEmployee;
GO
IF EXISTS (SELECT * FROM sysobjects WHERE name = 'InsertEmployee'
   AND user_name(uid) = 'dbo')
   DROP PROCEDURE [dbo].[InsertEMployee];
GO
CREATE PROCEDURE [dbo].[InsertEmployee]
(
   @FirstName varchar (50),
   @LastName varchar (70),
   @DepartmentID int,
   @Salary money
)
AS
   SET NOCOUNT OFF;
INSERT INTO tblEmployee(FirstName, LastName, DepartmentID, Salary)
   VALUES (@FirstName, @LastName, @DepartmentID, @Salary);
GO
IF EXISTS (SELECT * FROM sysobjects WHERE name = 'UpdateEmployee'
   AND user_name(uid) = 'dbo')
```

```
  DROP PROCEDURE [dbo].[UpdateEmployee];
Go

CREATE PROCEDURE [dbo].[UpdateEmployee]
(
   @FirstName varchar (50),
   @LAstName varchar (70),
   @DepartmentID int,
   @Salary money,
   @Original_ID int
)
AS
   SET NOCOUNT OFF;

UPDATE tblEmployee SET FirstName = @FirstName, LastName = @LastName ,
   DepartmentID = @DepartmentID, Salary = @salary
WHERE (ID = @Original_ID);
GO

IF EXISTS (SELECT * FROM sysobjects WHERE name = 'DeleteEmployee'
   AND user_name(uid) = 'dbo')

  DROP PROCEDURE [dbo].[DeleteEmployee];
Go

CREATE PROCEDURE [dbo].[DeleteEmployee]
(
   @Original_ID int
)
AS
   SET NOCOUNT OFF;
DELETE FROM tblEmployee WHERE (ID = @Original_ID);
GO
```

כעת נחזור לקוד היישום שלנו. תחילה נשנה את שורת הקוד הראשונה בשגרה btnLoad_Click, כך שבמקום לקרוא ל-LoadCommandBuilder, היא תקרא ל-LoadExplicitCode. בנוסף, כאשר נכתב הקוד לביצוע עדכונים, היו שגיאות תוכנה (בגים). כדי שניתן יהיה לפתור את השגיאות, הוספנו בלוק Try-Catch לשגרה btnUpdate_Click. כעת, השגרה תיראה כך:

הקוד בגירסה זו יהיה בשיטה btnUpdate_Click2.

VB

```
Private Sub btnUpdate_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles btnUpdate.Click
    Try
        daEmployees.Update(dsEmployeeInfo, "Employees")

    Catch es As SqlException
        MessageBox.Show(es.Message)

    End Try
  End Sub
```

C#

```
private void btnUpdate_Click(object sender, System.EventArgs e)
{
        try
        {
                daEmployees.Update(dsEmployeeInfo, "Employees");
        }
        catch (SqlException es)
        {
                MessageBox.Show(es.Message);
        }
}
```

לבסוף, קוד 4.5 מכיל את הגדרת הפקודות עבור מתאם הנתונים daEmployees.

קוד 4.5-VB שגרה לקביעת ארבע הפקודות המותאמות עבור מתאם הנתונים dsEmployees

```
Private Sub LoadExplicitCode()
    Dim param As SqlParameter

    If conn.State = ConnectionState.Closed Then
        conn.Open()
    End If

    'Create New DataAdapter Object
    daEmployees = New SqlDataAdapter()

    ' Set up custom Select Command (SP)
    daEmployees.SelectCommand = New SqlCommand()
    With daEmployees.SelectCommand
```

```
    .Connection = conn
    .CommandType = CommandType.StoredProcedure
    .CommandText = "SelectEmployees"
End With

' Set up custom Insert Command (SP)
daEmployees.InsertCommand = New SqlCommand()
With daEmployees.InsertCommand
    .Connection = conn
    .CommandType = CommandType.StoredProcedure
    .CommandText = "InsertEmployee"
End With

param = daEmployees.InsertCommand.Parameters.Add( _
 New SqlParameter("@FirstName", SqlDbType.VarChar, 50))
param.Direction = ParameterDirection.Input
param.SourceColumn = "FirstName"
param.SourceVersion = DataRowVersion.Current

param = daEmployees.InsertCommand.Parameters.Add( _
 New SqlParameter("@LastName", SqlDbType.VarChar, 70))
param.Direction = ParameterDirection.Input
param.SourceColumn = "LastName"
param.SourceVersion = DataRowVersion.Current

param = daEmployees.InsertCommand.Parameters.Add( _
  New SqlParameter("@DepartmentID", SqlDbType.Int))
param.Direction = ParameterDirection.Input
param.SourceColumn = "DepartmentID"
param.SourceVersion = DataRowVersion.Current

param = daEmployees.InsertCommand.Parameters.Add( _
  New SqlParameter("@Salary", SqlDbType.Money))
param.Direction = ParameterDirection.Input
param.SourceColumn = "Salary"
param.SourceVersion = DataRowVersion.Current

' Set up custom Update Command (SP)
daEmployees.UpdateCommand = New SqlCommand()
With daEmployees.UpdateCommand
    .Connection = conn
    .CommandType = CommandType.StoredProcedure
    .CommandText = "UpdateEmployee"
End With
```

```
    param = daEmployees.UpdateCommand.Parameters.Add( _
     New SqlParameter("@FirstName", SqlDbType.VarChar, 50))
    param.Direction = ParameterDirection.Input
    param.SourceColumn = "FirstName"
    param.SourceVersion = DataRowVersion.Current

    param = daEmployees.UpdateCommand.Parameters.Add( _
     New SqlParameter("@LastName", SqlDbType.VarChar, 70))
    param.Direction = ParameterDirection.Input
    param.SourceColumn = "LastName"
    param.SourceVersion = DataRowVersion.Current

    param = daEmployees.UpdateCommand.Parameters.Add( _
      New SqlParameter("@DepartmentID", SqlDbType.Int))
    param.Direction = ParameterDirection.Input
    param.SourceColumn = "DepartmentID"
    param.SourceVersion = DataRowVersion.Current

    param = daEmployees.UpdateCommand.Parameters.Add( _
      New SqlParameter("@Salary", SqlDbType.Money))
    param.Direction = ParameterDirection.Input
    param.SourceColumn = "Salary"
    param.SourceVersion = DataRowVersion.Current

    param = daEmployees.UpdateCommand.Parameters.Add( _
     New SqlParameter("@Original_ID", SqlDbType.Int))
    param.Direction = ParameterDirection.Input
    param.SourceColumn = "ID"
    param.SourceVersion = DataRowVersion.Original

    ' Set up custom Delete Command (SP)
    daEmployees.DeleteCommand = New SqlCommand()
    With daEmployees.DeleteCommand
        .Connection = conn
        .CommandType = CommandType.StoredProcedure
        .CommandText = "DeleteEmployee"
    End With

    param = daEmployees.DeleteCommand.Parameters.Add( _
     New SqlParameter("@Original_ID", SqlDbType.Int))
    param.Direction = ParameterDirection.Input
    param.SourceColumn = "ID"
    param.SourceVersion = DataRowVersion.Original

End Sub
```

קוד #C-4.5 שגרה לקביעת ארבע הפקודות המותאמות עבור מתאם הנתונים dsEmployees

```
private void LoadExplicitCode()
        {
          SqlParameter param;

          if (conn.State == ConnectionState.Closed)
          {
                    conn.Open();
          }

          // Create new DataAdapter Object
          daEmployees = new SqlDataAdapter();

          // Set up custom select command (SP)
          SqlCommand sqlCommand = new SqlCommand("SelectEmployees",
                    conn);
          sqlCommand.CommandType = CommandType.StoredProcedure;

          daEmployees.SelectCommand = sqlCommand;

          // Set up custom Insert Command (SP)
          sqlCommand = new SqlCommand("InsertEmployee", conn);
          sqlCommand.CommandType = CommandType.StoredProcedure;

          daEmployees.InsertCommand = sqlCommand;

          param = daEmployees.InsertCommand.Parameters.Add(
                    new SqlParameter("@FirstName", SqlDbType.VarChar, 50));
          param.Direction = ParameterDirection.Input;
          param.SourceColumn = "FirstName";
          param.SourceVersion = DataRowVersion.Current;

          param = daEmployees.InsertCommand.Parameters.Add(
                    new SqlParameter("@LastName", SqlDbType.VarChar, 70));
          param.Direction = ParameterDirection.Input;
          param.SourceColumn = "LastName";
          param.SourceVersion = DataRowVersion.Current;

          param = daEmployees.InsertCommand.Parameters.Add(
                    new SqlParameter("@DepartmentID", SqlDbType.Int));
          param.Direction = ParameterDirection.Input;
          param.SourceColumn = "DepartmentID";
```

```
param.SourceVersion = DataRowVersion.Current;

param = daEmployees.InsertCommand.Parameters.Add(
          new SqlParameter("@Salary", SqlDbType.Money));
param.Direction = ParameterDirection.Input;
param.SourceColumn = "Salary";
param.SourceVersion = DataRowVersion.Current;

// Set up custom Update Command (SP)
sqlCommand = new SqlCommand("UpdateEmployee", conn);
sqlCommand.CommandType = CommandType.StoredProcedure;

daEmployees.UpdateCommand = sqlCommand;

param = daEmployees.UpdateCommand.Parameters.Add(
          new SqlParameter("@FirstName", SqlDbType.VarChar, 50));
param.Direction = ParameterDirection.Input;
param.SourceColumn = "FirstName";
param.SourceVersion = DataRowVersion.Current;

param = daEmployees.UpdateCommand.Parameters.Add(
          new SqlParameter("@LastName", SqlDbType.VarChar, 70));
param.Direction = ParameterDirection.Input;
param.SourceColumn = "LastName";
param.SourceVersion = DataRowVersion.Current;

param = daEmployees.UpdateCommand.Parameters.Add(
          new SqlParameter("@DepartmentID", SqlDbType.Int));
param.Direction = ParameterDirection.Input;
param.SourceColumn = "DepartmentID";
param.SourceVersion = DataRowVersion.Current;

param = daEmployees.UpdateCommand.Parameters.Add(
          new SqlParameter("@Salary", SqlDbType.Money));
param.Direction = ParameterDirection.Input;
param.SourceColumn = "Salary";
param.SourceVersion = DataRowVersion.Current;

param = daEmployees.UpdateCommand.Parameters.Add(
          new SqlParameter("@Original_ID", SqlDbType.Int));
param.Direction = ParameterDirection.Input;
param.SourceColumn = "ID";
param.SourceVersion = DataRowVersion.Original;
```

```
    // Set up custom Delete Command (SP)
    sqlCommand = new SqlCommand("DeleteEmployee", conn);
    sqlCommand.CommandType = CommandType.StoredProcedure;

    daEmployees.DeleteCommand = sqlCommand;

    param = daEmployees.DeleteCommand.Parameters.Add(
            new SqlParameter("@Original_ID", SqlDbType.Int));
    param.Direction = ParameterDirection.Input;
    param.SourceColumn = "ID";
    param.SourceVersion = DataRowVersion.Original;
}
```

הערה

ניתן לכתוב בצורה קומפקטית יותר (פחות שורות קוד) את הקוד המקצה ערכים לכל אחד מאובייקטי הפרמטרים, על ידי קריאה להעמסת יתר (overloaded) של השיטה Add. גירסה חילופית זו מקבלת ערכים עבור כל תכונות הפרמטרים הדרושים בקריאה לשיטה אחת עם רשימה ארוכה של פרמטרים. היתרון בגישה שהשתמשנו בה בקוד הקודם הוא הקריאות אשר טובה יותר מאשר בחלופה.

הקוד בשגרה LoadExplicitCode ארוך למדי, אך הוא מובן מעצמו. קל להבין אותו לאחר שנקבע הממשק (פרמטרים וסוגים) לשגרות המאוחסנות. עבור כל אחד מתכונות Command, נוצר מופע חדש של האובייקט SQLCommand. נקצה את האובייקט המשותף Connection לאובייקט זה, ונקבע את התכונות CommandType ו-CommandText. לאחר מכן, עלינו ליצור ולהגדיר את כל פרמטרי הפקודה הדרושים לכל פקודה.

המשך לבצע פעולות בטופס frmUpdates, כפי שעשית קודם לכן, וודא שהוא פועל כשורה. נראה כי הטופס פועל כפי שפעל לפני כן, אך ההבדל הוא בכך שעל ידי קריאה ל-LoadExplicitCode הוא משתמש בפקודות המותאמות שלנו לביצוע עדכונים במסד הנתונים. גישה זו דורשת קידוד נוסף, אך עם זאת היא מקנה גמישות רבה יותר, ביצועים משופרים וניהול מרכזי של שגרות מאוחסנות במסד הנתונים.

הטמעת לוגיקה עסקית בפקודות עדכון

השימוש בשגרות מאוחסנות כפקודות עדכון מותאמות עבור ה-DataAdapter מאפשר להטמיע לוגיקה עסקית בשגרות מאוחסנות, אשר מופעלות באופן אוטומטי. החידוש בהשוואה לגרסאות קודמות של ADO ומודלים אחרים לגישה לנתונים אינו בהטמעת לוגיקה בשגרות מאוחסנות – דבר זה נעשה תמיד. החידוש הוא בכך ששגרות

מאוחסנות אלו נקראות באופן אוטומטי כאשר מבצעים עדכון אצווה (batch), ואין צורך להפעיל אותן בצורה מפורשת מקוד היישום.

כדי לצפות בהדגמה של גישה זו, עליך לשנות את השגרה המאוחסנת. נניח שהלוגיקה העסקית שלנו היא: אם אנו מוסיפים רשומת עובד חדשה והעמודה Salary ריקה (null) או 0, משכורת העובד תקבע כפונקציה של המחלקה שהוא נמצא בה. למשימה זו נשתמש בלוגיקה פשוטה. העובד יקבל אוטומטית משכורת אשר תהיה DepartmentID אשר נגזרת מהמחלקה שהעובר נמצא בה, כפול 1,000$. כעת, השגרה המאוחסנת תיראה כך:

```
CREATE PROCEDURE dbo.InsertEmployee
(
  @FirstName varchar (50),
  @LastName varchar (70),
  @DepartmentID int,
  @Salary money
)
AS
  SET NOCOUNT OFF;

  if (@Salary = 0 or @Salary is null)
  begin
    /* Do salary calculation */
    set @Salary = @DepartmentID * 1000
  end
  INSERT INTO tblEmployee(FirstName, LastName, DepartmentID, Salary)
    VALUES (@FirstName, @LastName, @DepartmentID, @Salary);
GO
```

הערה

השגרה המאוחסנת InsertEmployee כבר קיימת, ולכן עליך למחוק (להשמיט) את השגרה המאוחסנת הקיימת או לשנות בה את השורה הראשונה בתסריט, כדי שתהיה

```
ALTER PROCEDURE dbo.InsertEmployee
```

אם ברצונך להפעיל את התסריט הקודם מ- SQL Server Query Analyzer.

השיטה הנוספת האפשרית היא להכניס את הקוד החדש לתסריט שהשתמשת בו ליצירת השגרות המאוחסנות. התסריט crtEmployeePRO2.sql מכיל כבר שיטה זו בגרסתה החדשה.

כעת תוכל להפעיל את הפרויקט DataSetCode מבלי לערוך שינויים בקוד היישום. הוסף רשומות עובד חדשות בטופס frmUpdates, ובדוק אם השגרה המאוחסנת מקצה משכורות נכונות לשורות שהוכנסו באופן אוטומטי. זכור שאת המשכורת החדשה תראה רק לאחר שתעלה את הנתונים מחדש.

שימוש ברכיב העיצוב DataAdapter

לאחר הטיפול בקוד שבסעיף הקודם, אנו מניחים שאתה כבר מעוניין בדרך יותר קלה לביצוע המשימה. כעת תלמד על DataAdapter Configuration Wizard.

ה- DataAdapter Configuration Wizard מציע מבחר אפשרויות להגדרת אובייקט DataAdapter, אשר נוסף לטופס כרכיב עיצוב. למעשה, רכיב זה הינו מתקדם יותר מרכיבי העיצוב שעסקנו בהם עד עתה; הוא אינו מספק ממשק גרפי בלבד לקביעת אובייקטים ותכונות רבים של הרכיב, אלא גם מציע אפשרויות שונות המשפיעות על הקוד אשר נוצר באופן אוטומטי. התחל בביצוע פעולות אלו:

1. פתח את הטופס frmUpdates במעצב הטפסים.
2. מהכרטיסיה Data של תיבת הכלים, גרור את הרכיב SqlDataAdapter לשטח העיצוב של frmUpdates. רכיב זה אינו גלוי בזמן ההפעלה, ולכן הוא יופיע ב-tray הרכיבים מתחת לשטח העיצוב של הטופס. DataAdapter Configuration Wizard יופעל אוטומטית. כדי להמשיך, לחץ **Next** בתיבת הדו-שיח Welcome.
3. בתיבת הדו-שיח Data Connection, בחר התחברות למסד הנתונים Novelty. אם לא קיימות כרגע התחברויות, תוכל ליצור התחברות באמצעות לחיצה על New Connection, כדי להציג את תיבת הדו-שיח OLEDB Data Link properties. לאחר בחירת התחברות תקפה, לחץ **Next**.
4. ב- Query Type, תוכל לציין פקודת SQL, יצירה של שגרות חדשות או שימוש בשגרות מאוחסנות קיימות. נמשיך בברירת המחדל של שימוש בפקודות SQL, למרות שבפועל ייתכן שיהיה שימושי יותר לאפשר לאשף ליצור שגרות מאוחסנות עבורך. לחץ **Next**.
5. בתיבת הדו-שיח הבאה, עליך להזין פקודת Select לשימוש DataAdapter כבסיס לשלוש פקודות העדכון האחרות, אם הן נוצרות. הזן את השורה הבאה לתיבת הטקסט:

```
SELECT FirstName, LastName, DepartmentID, Salary, ID FROM tblEmployee
```

בתיבת דו-שיח זו תמצא שני לחצנים נוספים. הלחצן Advanced Options מציג תיבת דו-שיח ובה אפשרויות לשליטה בדרך יצירת הפקודות. תיבת הסימון הראשונה מציינת אם יש ליצור את שלוש פקודות העדכון (או רק להשתמש במתאם הנתונים למילוי DataSet). תיבת הסימון השנייה מציינת אם על הפקודות Update ו-Delete לכלול משפט WHERE, אשר קובע אם הרשומה שונתה במסד הנתונים מאז שנטענה ל-DataSet. התיבה השלישית מציינת אם יש להוסיף פקודת

Select לפקודות Insert ו-Update, כדי להחזיר את השורה כשהיא מלאה בערכי עמודה המחושבים בשרת, כגון ערכי העמודה Identity וערכי עמודת ברירת המחדל. הלחצן Query Builder מציג חלון סטנדרטי לבניית שאילתה, אשר מאפשר לעצב הצהרת שאילתת Select בצורה גרפית במקום להזין את השאילתה באופן ידני לתיבת הטקסט, כפי שעשינו בשלב הקודם.

6. לחץ **Next** כדי לצפות בסיכום השלבים שבוצעו על ידי האשף.

7. לחץ **Finish** כדי להחיל את ההגדרות על הרכיב DataAdapter.

הערה

לאחר שנוצר הרכיב DataAdapter, תוכל לשנות את תכונותיו והגדרותיו באמצעות החלון Properties, או באמצעות הפעלה נוספת של Configuration Wizard. ניתן להפעיל את האשף מתוך רכיב DataAdapter קיים באמצעות לחיצה ימנית על הרכיב במגש (tray) הרכיבים, ומתפריט הקיצור לבחור את הפריט ...Configure Data Adapter. תוכל גם לאתחל את האשף באמצעות בחירת הרכיבים במגש הרכיבים ולחיצה על הקישור ...Configure Data Adapter מהחלון Properties (בחלונית שבין רשימת התכונות לבין החלונית Description).

בדומה ל-CommandBuilder, גם האשף DataAdapter מיועד ליצירת פקודות עבור טבלת מסד נתונים יחידה ונפרדת. עם זאת, אשף זה מציע מספר אפשרויות הגדרה, כגון שימוש בשגרות מאוחסנות חדשות או קיימות, כדי להקנות גמישות ויעילות רבה יותר בכתיבת קוד מסחרי.

כעת עלינו לקשר את מתאם הנתונים (DataAdapter), אשר קיבל באופן אוטומטי את השם SqlDataAdapter1, עם התוכנית הקיימת שלנו. כדי לשמור על העקביות, נפתח בצורה מפורשת מהקוד שלנו את ההתחברות SqlConnection1 שנוצרה.

יש לשנות את השגרה btnLoad_Click, כך שתקרא ל-LoadWizardCode במקום ל-LoadExplicitCode. בנוסף, עליה לקרוא לשיטה Fill של מתאם הנתונים החדש שנוצר. יש לשנות גם את השגרה btnUpdate_Click, כך שתשתמש ברכיב החדש SqlDataAdapter1. לבסוף, עלינו להוסיף את השגרה LoadWizardCode, אשר נותר לה רק לפתוח את ההתחברות החדשה. שלוש שגרות אלו מוצגות בקוד 4.6.

קוד 4.6-VB שגרות שהשתנו ונוספו לטיפול ברכיב SqlDataAdapter ביישום הקיים

```
Private Sub btnLoad_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnLoad.Click
    dsEmployeeInfo = New DataSet()

    'LoadCommandBuilder()
    'LoadExplicitCode()
    'LoadWizardCode()

    'Config grid
    Me.grdDataGrid.PreferredColumnWidth = 110
    Me.grdDataGrid.AllowSorting = True

    'Fill Data Set
    'daEmployees.Fill(dsEmployeeInfo, "Employees")
    SqlDataAdapter1.Fill(dsEmployeeInfo, "Employees")

    'Assign DataSet to DataGrid
    Me.grdDataGrid.DataSource = dsEmployeeInfo.Tables("Employees")
    Me.btnUpdate.Enabled = True
End Sub

Private Sub btnUpdate_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnUpdate.Click
    Try
        'daEmployees.Update(dsEmployeeInfo, "Employees")
        SqlDataAdapter1.Update(dsEmployeeInfo, "Employees")

    Catch es As SqlException
        MessageBox.Show(es.Message)

    End Try
End Sub

Private Sub LoadWizardCode()
    If SqlConnection1.State = ConnectionState.Closed Then
        SqlConnection1.Open()
    End If
End Sub
```

קוד #C-4.6 שגרות שהשתנו ונוספו לטיפול ברכיב SqlDataAdapter ביישום הקיים

```
private void btnLoad_Click(object sender, System.EventArgs e)
        {
          dsEmployeeInfo = new DataSet();

          // LoadCommandBuilder();

          LoadWizardCode();
          // Config grid
          grdDataGrid.PreferredColumnWidth = 110;
          grdDataGrid.AllowSorting = true;

          //Fill Data Set
          sqlDataAdapter1.Fill(dsEmployeeInfo, "Employees");

          // Assign DataSet to DataGrid
          grdDataGrid.DataSource =
                    dsEmployeeInfo.Tables["Employees"];
          btnUpdate.Enabled = true;

        }

private void btnUpdate_Click(object sender, System.EventArgs e)
        {
          try
          {
                    // daEmployees.Update(dsEmployeeInfo, "Employees");
                    sqlDataAdapter1.Update(dsEmployeeInfo, "Employees");
                    MessageBox.Show("Finish work");
          }
          catch (SqlException es)
          {
                    MessageBox.Show(es.Message);
          }
        }

private void LoadWizardCode()
        {
          if (sqlConnection1.State == ConnectionState.Closed)
          {
                    sqlConnection1.Open();
          }
        }
```

אם אתה תוהה כיצד נראה הקוד באשף, תוכל לצפות בו על ידי פתיחת frmUpdates בעורך הקוד של Visual Studio, ופתיחת איזור הקוד Windows Form Designer שנוצר. קוד זה דומה למדי לקוד 4.5 שכתבנו בעצמנו.

הערה

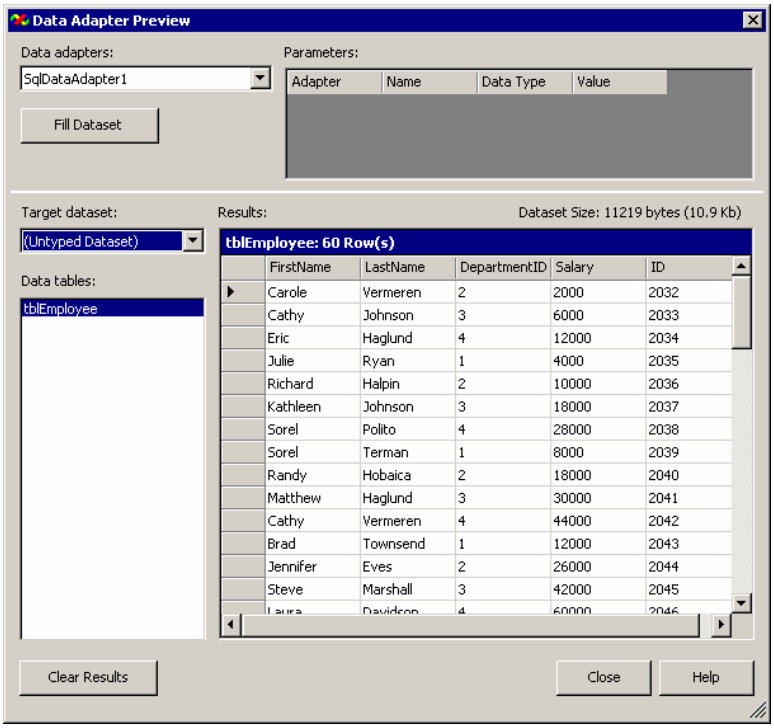
לא שינינו את שמות ברירת המחדל של האובייקטים ב-**VB** (כגון, SqlDataAdapter1 ו-SqlSelectCommand1), וגם לא ב-**C#** (sqlDataAdapter1 ו-sqlConnection1), אשר הוקצו כברירת מחדל על ידי DataAdapter Configuration Wizard. עם זאת, תוכל לשנות את השמות לשמות משמעותיים יותר או לשמות שיתאימו לשיטת הקצאת השמות שלך. ניתן לשנות את שמות רכיבי העיצוב על ידי בחירת הרכיב במגש הרכיבים, מתחת לשטח עיצוב הטופס והגדרת ערך התכונה Name בחלון Properties.

לשינוי שמות פקודות מסוימות (כגון, SelectCommand ו-InsertCommand), הישאר בחלון Properties של הרכיב DataAdapter. אתר את הפקודה שברצונך לשנות ולחץ על הסמל **+** כדי להרחיב את אובייקט הפקודה הרצוי. פעולה זו תחשוף את כל תכונות אובייקט הפקודה הנבחר ותאפשר לשנות את שמו ותכונותיו האחרות.

השיטה בגרסתה החדשה נמצאת בקוד המצורף בשם אחר: btnLoad_Click2. כדי לעבוד בצורה החדשה, החלף את השמות בין שתי השיטות.

הפעל את התוכנית כדי לוודא שגירסה זו גם פועלת כשורה, כפי שעשית בגרסאות הקודמות של הפרויקט DataSetCode באמצעות CommandBuilder ופקודות SQL מפורשות. זכור באיזו מהירות ניתן ליצור קוד מלא זה. תוכל אף ליצור שגרות מאוחסנות!

לשימוש ברכיבי עיצוב מסוג DataAdapter יש תכונה שימושית נוספת, Preview Data, אשר בעזרתה תוכל לצפות בנתונים המוחזרים מפקודת Select של מתאם הנתונים בזמן העיצוב. כדי לנצל תכונה זו, לחץ לחיצה ימנית על הרכיב **DataAdapter** במגש הרכיבים ומתפריט הקיצור בחר **Preview Data**. תוכל גם להציג חלון תיבת דו-שיח זה על ידי בחירה ברכיב הרצוי לך ממגש הרכיבים ולאחר מכן לחיצה על הקישור **Preview Data...** בחלון Properties (בחלונית שבין רשימת התכונות לחלונית Description). בחר במתאם הנתונים הרצוי מתוך תיבת הרשימה של DataAdapter ולאחר מכן לחץ על **Fill DataSet**. תוצאות ה-SqlDataAdapter שהוספנו ל-frmUpdates מוצגות בתרשים 4.3.

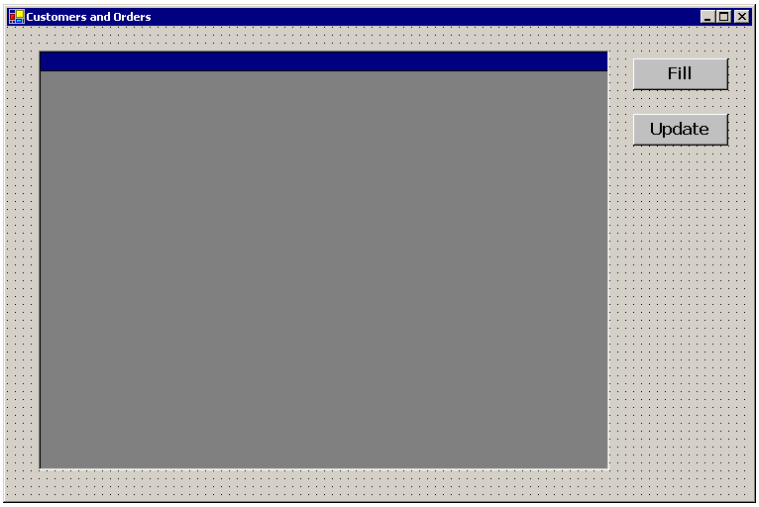


תרשים 4.3 נתוני tblEmployee מוצגים בחלון DataAdapter Preview.

תסריט עסקי 4.1:
שילוב מספר טבלאות עם קשרים

כבר נאמר בפרק זה שלא ניתן להשתמש בקלות באף אחת מהטכניקות הזמינות להגדרת פקודות עדכון של טבלאות אחדות. דבר זה נכון במיוחד אם קיים **קשר** (Relation) מוגדר בין הטבלאות, כמו במקרה שבו קשר אב-בן קיים בין הטבלאות. האם המשמעות היא ש-ADO.NET אינו מסוגל לטפל במצב מעין זה? המפתחת שפיתחה את מסדי הנתונים ב- Jones Novelties השתמשה ביכולות ADO.NET והוכיחה כי זה אינו המקרה; היא פיתחה טופס אשר מציג ומאפשר עדכונים בנתוני לקוחות ובנתוני הזמנות קשורות. שיטה זו מדגימה גם את השימוש באצוות של פקודות SQL Server למילוי שתי הטבלאות על ידי ביצוע מהלך אחד של התחברות לשרת. המפתחת המשיכה לבצע את הפעולות שלהלן כדי ליצור את הטופס:

1. הפעלת Visual Studio.NET.
2. יצירת פרויקט Visual Basic Windows Application חדש.
3. קריאה לפרויקט BusinessCase6.
4. ציון נתיב לשמירת קבצי הפרויקט.
5. הגדלת הטופס Form1 וקביעת ערך התכונה Name שלו ל-**frmCustomersOrders** ואת ערך התכונה Text ל- **Customers and Orders**.
6. שינוי שם הקובץ ל-**frmCustomersOrders** ב-FileName שבחלונית properties של הקובץ הראשוני שנוצר (Form1.cs).
7. הוספת לחצן btnFill עם ערך **Fill** לתכונה Text, הוספת לחצן btnUpdate עם ערך **Update** לתכונה Text והוספת DataGrid בשם **grdCustomersOrders**.
8. סידור הפקדים כפי שמוצג בתרשים 4.4.



תרשים 4.4 סידור פקדים ב-frmCustomersOrders.

לאחר הוספת השורות בתחילת הקובץ, נקבל:

VB

```
Imports System.Data
Imports System.Data.SqlClient
```

C#

```
Using System.Data.SqlClient;
```

שורות קוד אלו גורמות להוספת קוד 4.7 לגוף הגדרת המחלקה של frmCustomersOrders.

קוד 4.7-VB שגרות לטעינה ועדכון של מספר טבלאות בעלות התייחסות

```
Private ds As DataSet
Private cn As New SqlConnection( _
        "server=localhost;uid=sa;database=Novelty")
  Private Sub btnFill_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles btnFill.Click
    Dim da As New SqlDataAdapter()

    grdCustomersOrders.DataSource = Nothing
    ds = New DataSet()

    ' Set up batch select command
    da.SelectCommand = New SqlCommand()
    da.SelectCommand.Connection = cn
```

```
    da.SelectCommand.CommandType = CommandType.Text
    da.SelectCommand.CommandText = _
          "select * from tblCustomer; SELECT * FROM tblOrder"

    ' Table mappings for clear names
    da.TableMappings.Add("Table", "Customers")
    da.TableMappings.Add("Table1", "Orders")

    ' Load Data
    da.Fill(ds)

    ' Manually add relation
    ds.Relations.Add("Customer_Orders", _
       ds.Tables("Customers").Columns("ID"), _
       ds.Tables("Orders").Columns("CustomerID"))

    ' Display the data
    grdCustomersOrders.DataSource = ds

  End Sub
```

קוד #C-4.7 שגרות לטעינה ועדכון של מספר טבלאות בעלות התייחסות

```
      private DataSet ds = new DataSet();
      private SqlConnection cn = new
SqlConnection("server=localhost;uid=sa;database=Novelty");
      private void btnFill_Click(object sender, System.EventArgs e)
      {
        SqlDataAdapter da = new SqlDataAdapter();


        grdCustomersOrders.DataSource = null;

        // Setup batch select command
        da.SelectCommand = new SqlCommand();
        da.SelectCommand.Connection = cn;
        da.SelectCommand.CommandType = CommandType.Text;
        da.SelectCommand.CommandText =
                "select * from tblCustomer; SELECT * FROM tblOrder";

        //Table mapping for clear names
        da.TableMappings.Add("Table", "Customers");
        da.TableMappings.Add("Table1", "Orders");
```

```
        da.Fill(ds);

        ds.Relations.Add("Customer_Orders",
                ds.Tables["Customers"].Columns["ID"],
                ds.Tables["Orders"].Columns["CustomerID"]);
        grdCustomersOrders.DataSource = ds;
    }
```

השגרה הראשונה, btnFill_Click, קוראת את שתי הטבלאות ממסד הנתונים במהלך יחיד, על ידי הפעלת אצוות פקודות SQL SERVER. הפקודות השונות מופרדות באמצעות נקודה פסיק (';') במחרוזת CommandText.

שמות ברירת המחדל לטבלאות שב-DataSet, Table ו-Table1 ממופים לשמות משמעותיים יותר – Customers ו-Orders, הנה כך:

```
' Table mappings for clear names
da.TableMappings.Add("Table", "Customers")
da.TableMappings.Add("Table1", "Orders")
```

הערה

בפרק 5 נדון בהרחבה במיפויי טבלאות ועמודות.

לאחר מילוי מסד הנתונים ds בנתונים, נוצר אובייקט DataRelation אשר מקשר בין שתי הטבלאות. כאן, הטבלה Customers היא טבלת האב והטבלה Orders היא טבלת הבן. שורת הקוד האחרונה כורכת את ה-DataSet לרשת לשם הצגת הנתונים.

השגרה השנייה, btnUpdate_Click, גורמת שהשינויים יבוצעו בשתי הטבלאות שבמסד הנתונים. בפעולה זו יש להבטיח את שלמות הנתונים בקשר אב-בן. לרוע המזל, דבר זה אינו מתבצע באופן אוטומטי. מפתחת מסד הנתונים צריכה לקבץ סוגי שינויים ולאחר מכן להפעיל אותם בסדר הנכון; עליה להפעיל את השינויים בסדר הבא עבור שתי טבלאות עם קשר אב-בן:

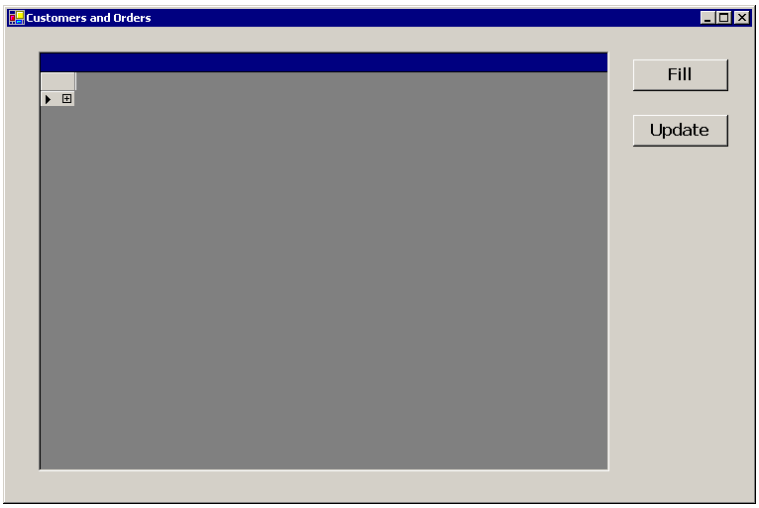
1. למחוק רשומות בטבלת הבן.
2. להכניס, לעדכן ולמחוק רשומות בטבלת האב.
3. להכניס ולעדכן רשומות בטבלת הבן.

כדי לבצע את השינויים המתאימים, השגרה קוראת לשיטה GetChanges של הטבלה המתאימה; היא מציינת את המסנן הרצוי במצב השורה, על פי הנדרש. כל קריאה ל-GetChanges מחזירה טבלה עם הרשומות שהשתנו בלבד (לפי מסנן מצב השורה). אם לא קיימות שורות שהשתנו, מוחזר הערך Nothing. אם קיימת לפחות רשומה אחת שהשתנתה, מתבצעת קריאה לשיטה Update של מתאם הנתונים כדי לבצע את העדכון

במסד הנתונים. הקוד בפונקציה זו נמצא בתוך בלוק Try-Catch לטיפול בשגיאות שעלולות להיות במהלך עדכון מסד הנתונים.

כעת הגיע הזמן לבדוק את הטופס שנוצר על ידי מפתחת מסדי הנתונים. לשם כך, בצע את הפעולות שלהלן:

1. הפעל את הפרויקט BusinessCase6 ולחץ **Fill**. פעולה זו תגרום למילוי מסד הנתונים בנתונים ממסד הנתונים Novelty. עם זאת, שורת הקוד grdCustomersOrders.DataSource = ds כורכת לרשת (grid) את כל מסד הנתונים ולא טבלה מסוימת בלבד, ולכן התצוגה תכיל את תחילת הרשת ואת הסימן +, כפי שמוצג בתרשים 4.5, אשר מציין כי ניתן להרחיב את התצוגה.



תרשים 4.5 התצוגה ההתחלתית של frmCustomersOrders, לאחר מילוי נתונים במסד הנתונים.

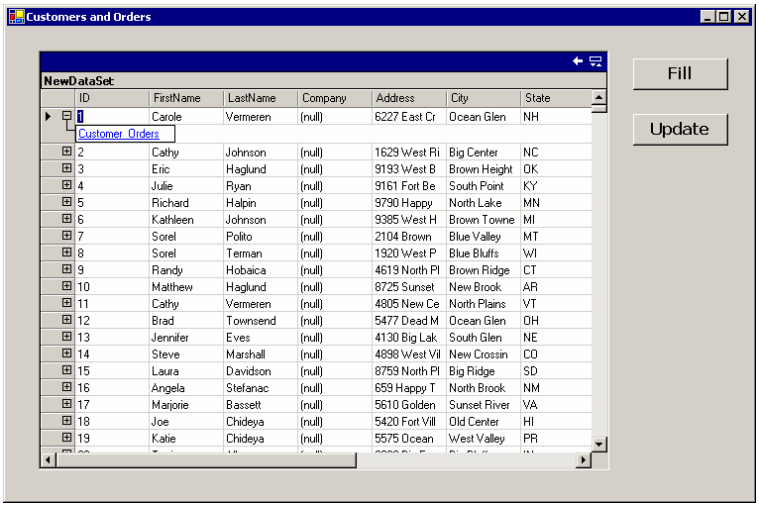
2. לחץ על הסמל + כדי להרחיב את תצוגת הרשת. כעת הרשת תציג שני קישורים דמויי Web – קישור אחד עבור כל טבלה שבמסד הנתונים.

3. לחץ על הקישור **Customers**. כעת הרשת תציג את רשומות הנתונים בטבלה Customers. שים לב לכך שכל רשומה בטבלה Customers מכילה משמאלה את הסמל +, אשר מציין שהטבלה קשורה לטבלה אחרת – אחת או יותר. לחיצה על הסמל + תרחיב את רשימת קשרי הנתונים (DataRelations) של אותה טבלה. במקרה זה, נוצר קשר אחד בלבד בשגרה btnFillClick – הקשר Customers_Orders, כפי שמוצג בתרשים 4.6.

4. לחץ על הקשר **Customer_Orders** ברשומה הראשונה. פעולה זו תשיג ותציג את השורות בטבלה Orders אשר קשורות לרשומת Customers הנוכחית, באמצעות שימוש בהגדרות ה-DataRelation בשם Customer_Orders.

הערה

כאשר אתה מנווט בטבלה וב-DataRelations ברשת, תוכל לחזור אחורה בכל שלב באמצעות לחיצה על החץ Navigate Back בראש הרשת.



תרשים 4.6 הקישור CustomersOrders Relation המופנה לשורה בטבלה Customers.

התצוגה אינה אמורה להציג רשומות כלשהן מ-Orders, מכיון שנתוני ההזמנות של חברת Jones Novelties נמצאים עדיין במסד נתוני Access ולא הועברו עדיין למסד הנתונים החדש של SQL Server. חברת Jones מעוניינת לבדוק את המערכת החדשה שנמצאת בשלבי פיתוח, ולכן מפתחת מסדי הנתונים שלה צריכה להזין נתוני בדיקה באמצעות טופס זה. פעולה זו תגרום ליצירת נתוני בדיקה וגם תבטיח כי ניתן להוסיף שורות חדשות! הרשת תציג שורה חדשה אשר תתווסף לטבלה. ערך השדה CustomerID כבר נקבע לשדה הזיהוי (ID) של רשומת Customers הנוכחית, מכיון שהרשת מזהה כי זהו הערך בשורת הלקוח הקשור. המשך להוסיף ערכים בשדות OrderDate ו-Amount. אין טעם להוסיף ערך עבור השדה ID של Orders, מכיון שזו עמודת זיהוי אשר תקבל הקצאה אוטומטית של ערך ממסד הנתונים, כאשר הרשומה תתווסף אליו.

5. לחץ **Update** כדי להפעיל את השגרה btnUpdate_Click מקוד 4.7. פעולה זו תגרום לעדכון מסד הנתונים.

6. תוכל לוודא שפעולת העדכון (הוספת שורה חדשה) בוצעה על מסד הנתונים באמצעות לחיצה על Fill, כדי לגרום לטעינה חוזרת של נתונים ממסד הנתונים אל ה-DataSet ולרשת. נווט לטבלה Order שבשורת Customer הראשונה, ואתר שם את הרשומה החדשה שהוספת זה עתה.

בצע שינויים נוספים במסד הנתונים על ידי הוספה, מחיקה ושינוי של שורות בשתי הטבלאות ובדיקה שהעדכונים אכן בוצעו.

הערה

אתה בוודאי תוהה מדוע הצלחת למחוק בהצלחה שורת Customer למרות העובדה שהיא מכילה עדיין שורות Orders? הסיבה לכך היא שהגדרת ברירת המחדל של ה-ForeignKeyConstraint, אשר נוצר על ידי הקשר Customer_Orders, היא לבצע בצורה מדורגת (cascade) מחיקות (ועדכונים) המבוצעים על טבלת אב גם בטבלת הבן.

סיכום

בפרק זה בחנו מקרוב את מתאם הנתונים, DataAdapter, שהוא אובייקט עיקרי של ADO.NET. מתאם הנתונים הוא הגורם המתאם בין העולם המנותק של מסד הנתונים, ה-DataSet, והאובייקטים המשויכים אליו, לבין העולם המחובר של אובייקטי ספקי נתוני NET., אשר מתחברים בפועל ויוצרים קשר עם מקור נתונים פיסי.

מתאם הנתונים משמש למילוי מסד נתונים בנתונים ממקור נתונים כלשהו, באמצעות שימוש בפקודות מפורשות או בשגרות מאוחסנות. מתאם הנתונים מבצע גם עדכון אוטומטי של מקור הנתונים בשינויים שבוצעו במסד הנתונים, ומאפשר להתאים בצורה מלאה את הפקודות שמשתמשים בהן בפעולות הוספה, עדכון ומחיקה, המבוצעות על מקור הנתונים.

שאלות ותשובות

שאלה: לעיתים ברצוני ליצור ולמלא רשומות נתונים על ידי כתיבת קוד ולאחר מכן לעדכן את מסד הנתונים בנתונים אלה. כאשר ניסיתי לעשות זאת ב- ADO 2.X, הצלחתי ליצור את רשומות הנתונים, אך לא הצלחתי לעדכן את מסד הנתונים. האם יש שינוי כלשהו בנושא זה ב-ADO.NET?

תשובה: כמובן! כפי שהזכרנו בפרק 3, ה-DataSet הוא מכולת נתונים שאינה יודעת וגם אינה צריכה לדעת מהיכן הגיעו הנתונים האלה. אם תרצה לעדכן מסד נתונים מ-DataSet, חבר אותו אל מתאם הנתונים המכיל הגדרות מתאימות וקרא לשיטה Update של ה-DataAdapter כדי לעדכן את מסד הנתונים. דבר זה נכון גם אם נתוני ה-DataSet נוצרים בתכנות ולא באמצעות פקודת Select של מתאם הנתונים, אשר ממלאת את ה-DataSet. כאשר תהיה מוכן להעביר את נתוני מכולת הנתונים למקור נתונים, חבר אותו אל מתאם הנתונים המוגדר כראוי ובצע את העדכון.

פרק 5

ADO.NET –
תכונות וטכניקות נוספות

מה בפרק זה?

- גילוי התנגשויות
- מיפויים של עמודות וטבלאות
- טבלאות מדומות (DataViews)
- Strongly Typed DataSets

בפרקים הקודמים, בחנו מקרוב את התפיסה ואת ארכיטקטורת הגישה לנתונים של ADO.NET. כמו כן, הצגנו והדגמנו את השימוש באובייקטים רבים של ADO.NET, ובכלל זה שיטות ותכונות עיקריות. בפרק זה נבחר ארבע תכונות וטכניקות נוספות של ADO.NET, אשר בדרך כלל לא תשתמש בהן עד שתתחיל לכתוב קוד מסחרי. נקבץ ונציג נושאים אלה כדי לקדם אותך לנקודת הפתיחה, כאשר תתחיל לפתח יישומים.

גילוי התנגשויות

אם כתבת (או השתמשת) ביישום מסד נתונים מרובה משתמשים, סביר שעסקת במצבי התנגשות (concurrency conflicts), בעת שמספר משתמשים עובדים על אותו מסד נתונים בו-זמנית. התנגשות מתרחשת כאשר משתמשים אחדים מנסים לשנות בו-זמנית את אותן שורות נתונים במסד הנתונים. לדוגמה, משתמש A קורא נתונים ומשתמש B קורא את אותם הנתונים. עכשיו, למי יש זכות לעדכן אותם? אם משתמש A ישנה את הנתונים, הם יהיו שונים מאלה שמשתמש B עוסק בהם. כעת משתמש B רוצה גם לשנות את הנתונים, אבל איזה נתונים? האם את המקוריים שקרא, או את אלה שעודכנו על ידי A? כיצד ניתן להתמודד עם בעיה זו? קיימות שתי גישות בסיסיות לבקרה של שימוש בו-זמנית.

הגישה הראשונה הינה בקרה בו-זמנית פסימית, pessimistic concurrency control, הכרוכה בנעילת מסד הנתונים מפני גישה. על פי גישה זו, אפשרות הדריסה של השינויים שביצע משתמש A נמנעת על ידי הנתונים של B, מכיון שהיא אינה מתירה אף פעם מצב כזה. בגישה זו, כאשר המשתמש קורא בכוונה נתונים כדי לשנות אותם, מבוצעת נעילה של הנתונים. הנעילה גורמת לכך שהנתונים לא יהיו זמינים עבור משתמשים אחרים, עד שהמשתמש הראשון יסיים את משימתו וישחרר את הנעילה.

גישה זו שימושית במקרים בהם ישנה תחרות על השימוש באותם הנתונים, או במקרים שהמשתמש זקוק לנתונים המעודכנים ביותר. יישום טיפוסי של תרחיש מעין זה עשוי להיות ניהול מלאי בזמן אמת או מערכת ניהול הזמנות. המשתמש אינו רוצה לאשר הזמנה, אלא אם הוא בטוח שהפריט או הפריטים אכן נמצאים במלאי.

החסרונות העיקריים של גישת pessimistic concurrency control הם התקורה העודפת של ניהול בלתי פוסק של נעילת נתונים; הצורך בהתחברות מתמשכת למסד הנתונים; וחוסר תפקוד, בעיקר כאשר משתמשים בסביבה מבוזרת (כגון האינטרנט) ונועלים רשומות למספר רב של שניות ואף דקות.

הגישה השנייה הינה בקרה בו-זמנית אופטימית, optimistic concurrency control, שגם היא כרוכה בנעילה. על פי גישה זו, לא מתבצעת נעילה של שורות נתונים, אלא רק לפרק זמן קצר מאוד שבו נעשה העדכון שלהן בפועל. גישה זו מונעת את בעיית ניהול הנעילות ואת בעיית חסימת הגישה למסד הנתונים עקב נעילות; היא פועלת כשורה ואף מיועדת למצבים שבהם המשתמש מבצע פעולות עריכה בזמן שהוא מנותק ממסד הנתונים. עם זאת, מה קורה כאשר משתמש B חוזר ומעדכן נתונים שהשתנו

כבר על ידי משתמש A? אפשרות אחת היא לקבוע כי העדכון האחרון הוא הקובע. ראוי לציין כי ישנם יישומים מעטים היכולים לפעול בדרך זו.

שיפור דרך פעולה זו הוא על ידי הבדיקה של משתמש B האם הנתונים שלו שונו מאז שאוחזרו לראשונה. בדיקה זו נקראת concurrency violation. קיימות שתי גישות בסיסיות ליישום פעולת הבדיקה והגילוי:

1. הגדרת חותמת זמן (timestamp) או מספר גירסה ייחודי לכל שורה, אשר יעודכן בכל פעם שהשורה משתנה. הערך המקורי של תווית הזמן או מספר הגירסה נכלל כחלק מהמשפט WHERE של הצהרת העדכון.
2. שמירת הערכים המקוריים של השדות. ערכים אלה הופכים לתנאים נוספים במשפט WHERE של הצהרת העדכון.

בכל אחד מהמקרים, אם השורה המקורית משתנה, יתקיים התנאי במשפט WHERE, השורה לא תימצא ולא יבוצע העדכון שלה.

הערה

אם השתמשת בנעילה אופטימית ב- ADO 2.X ותהית מדוע הודעת השגיאה הקשורה ל- concurrency violation מציינת כי לא ניתן למצוא את השורה שצוינה, במקום לציין שמצב זה התרחש, הרי שגילית זה עתה את התשובה לכך.

ב-ADO.NET קיימת תמיכה לנעילה אופטימית, אך לא קיימת עדיין תמיכה מובנית לנעילה פסימית. Visual Studio מציע מספר אפשרויות ליישום נעילה אופטימית, אשר תואמות לתמיכה המקיפה בארכיטקטורת יישומים מבוזרים, מנותקים וא-סינכרוניים.

משפטי SQL עבור הפקודות Update ו-Delete, אשר נוצרו באמצעות CommandBuilder ו- DataAdapter Configuration Wizard, כוללים בשני המקרים את הפסוקית WHERE, אשר תפקידה לגלות התנגשויות הנובעות מפעולות בו-זמניות – concurrency conflicts. כעת נבחן את הקוד הרלוונטי אותו יצרנו בפרק 4 באמצעות DataAdapter Configuration Wizard. תוכל להשיג את הקוד שנוצר בכך שתרחיב את אזור הקוד ב- Windows Forms Designer, כאשר תצפה ב-frmUpdates בחלון הקוד. בחן תחילה את פקודות Update (מוצגות בעריכה שונה כדי לשפר את הקריאות) בקוד 5.1.

קוד 5.1 הצהרת SQL Update שנוצרה על ידי DataAdapter Configuration Wizard

```
UPDATE tblEmployee
SET FirstName = @FirstName,
LastName = @LastName,
DepartmentID = @DepartmentID,
Salary = @Salary
WHERE
(ID = @Original_ID) AND
(DepartmentID = @Original_DepartmentID OR
   @Original_DepartmentID IS NULL AND DepartmentID IS NULL) AND
(FirstName = @Original_FirstName) AND
(LastName = @Original_LastName) AND
(Salary = @Original_Salary OR @Original_Salary IS NULL AND Salary IS NULL);
SELECT FirstName, LastName, DepartmentID, Salary, ID
FROM tblEmployee WHERE (ID = @ID)
```

המשפט מתחיל כמו פקודת Update רגילה, אשר מגדירה את ערכי ארבע העמודות שניתן לעדכן בערכים החדשים שמועברים כפרמטרים לאובייקט UpdateCommand. פסוקית התנאי WHERE מכילה את שדה המפתח הראשי (ID), ואת הערכים המקוריים של כל אחת מהעמודות הנוספות. נערכת בדיקה האם ערכים מקוריים אלה תואמים לערכים הנוכחיים של השורה במסד הנתונים. בהמשך, נבדקים ערכים ריקים (NULL) במסד הנתונים ובערכים הנוכחיים בעמודות אשר יכולים להתקיים בהן ערכים ריקים.

לאחר הנקודה-פסיק, מופיעה הצהרת SELECT (ההצהרה שהזכרנו בעת הגדרת ה-DataAdapter). נקודה-פסיק הוא סימן מפריד בין פקודות באצווה, והפקודה SELECT מתווספת כברירת מחדל כדי להחזיר ליישום את השורה לאחר השינוי.

כעת נבחן את קוד 5.2, אשר מגדיר פרמטרים עבור האובייקט UpdateCommand.

קוד 5.2-VB קוד להגדרת פרמטרי פקודה הנוצרים על ידי DataAdapter Configuration Wizard

```
Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@FirstName",
System.Data.SqlDbType.VarChar, 50, "FirstName"))
Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LastName",
System.Data.SqlDbType.VarChar, 70, "LastName"))
Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@DepartmentID",
System.Data.SqlDbType.Int, 4, "DepartmentID"))
::
::
```

קוד #C-5.2 קוד להגדרת פרמטרי פקודה הנוצרים על ידי DataAdapter Configuration Wizard

```
this.sqlUpdateCommand1.Parameters.Add(new
System.Data.SqlClient.SqlParameter("@FirstName",
System.Data.SqlDbType.VarChar, 50, "FirstName"));
this.sqlUpdateCommand1.Parameters.Add(new
System.Data.SqlClient.SqlParameter("@LastName",
System.Data.SqlDbType.VarChar, 70, "LastName"));
this.sqlUpdateCommand1.Parameters.Add(new
System.Data.SqlClient.SqlParameter("@DepartmentID",
System.Data.SqlDbType.Int, 4, "DepartmentID"));
::
::
```

לאובייקט זה מוגדרים עשרה פרמטרי פקודה. ארבעת הפרמטרים הראשונים הם הערכים הנוכחיים (ערכים שייתכן שהשתנו) של העמודות שאנו רוצים לעדכן ברשומה שבמסד הנתונים. בפרק 3 הזכרנו את העובדה כי כל שורה יכולה להכיל עד ארבע גרסאות שונות של ערכים. כברירת מחדל, אם לא תציין אחרת, יופיע הערך הנוכחי של העמודה שקראת.

חמשת הפרמטרים הבאים הם הערכים המקוריים של כל העמודות המשמשות כערכים במשפט WHERE. שים לב לכך שכדי לאחזר את הערך המקורי של עמודה (ולא את הערך הנוכחי המוגדר כברירת מחדל), עליך לציין כך את גרסת השורה

System.Data.DataRowVersion.Original

שנמצאת בבנאי של SqlParameter אשר מתווסף לאובייקט הפקודה.

הערה

אין חובה לכלול בפסוקית WHERE את הערכים המקוריים של כל העמודות. תוכל להתאים לצרכיך את כל אחד מאובייקטי פקודת העדכון, ולקבוע כי בעת עדכון שורה, תוצג התראה רק אם משתמש אחר שינה עמודה מסוימת אחת או יותר. עם זאת, תוכל להמשיך בעדכון מסד הנתונים אם שונתה אחת או יותר מהעמודות האחרות.

הפרמטר האחרון הוא הערך הנוכחי של העמודה ID, שמשתמשים בו כפרמטר לפקודה SELECT, אשר מחזירה את הערכים המעודכנים של השורה.

לאחר כל פעולת הוספה, עדכון או מחיקה, מתאם הנתונים בוחן את מספר השורות שהושפעו מהפעולה. אם מספר זה הוא אפס, תיגרם שגיאה מסוג DBConcurrency Exception, מכיון שהנחה היא שמדובר לרוב בתוצאה של פגיעה בפעולה בו-זמנית (concurrency violation). נוכל להוסיף קוד לטיפול בשגיאה זו בבלוק Try-Catch בשגרה btnUpdate_Click, כפי שמוצג בקוד 5.3 (בקובץ frmUpdates).

קוד 5.3-VB בלוק Try-Catch עם מנגנון טיפול בשגיאות עבור DBConcurrencyException

```
Private Sub btnUpdate_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles btnUpdate.Click
    Try
        daEmployees.Update(dsEmployeeInfo, "Employees")
        'SqlDataAdapter1.Update(dsEmployeeInfo, "Employees")

    Catch ec As DBConcurrencyException
        ' Do something !!
    Catch es As SqlException
        MessageBox.Show(es.Message)

    End Try
End Sub
```

קוד 5.3-C# בלוק Try-Catch עם מנגנון טיפול בשגיאות עבור DBConcurrencyException

```
private void btnUpdate_Click(object sender, System.EventArgs e)
    {
        try
        {
                // daEmployees.Update(dsEmployeeInfo, "Employees");
                sqlDataAdapter1.Update(dsEmployeeInfo, "Employees");
                MessageBox.Show("Finish work");
        }
        catch (DBConcurrencyExption ec)
        {
                MessageBox.Show(es.Message);
        }
        catch (SqlException es)
        {
                MessageBox.Show(es.Message);
        }
    }
```

מיפויי טבלאות ועמודות

לאובייקט DataAdapter יש אוסף אובייקטים מסוג DataTableMapping שהוא חושף כתכונה TableMappings. מיפויים אלה מבוצעים כדי לאפשר מיפוי שמות טבלאות, ו/או שמות עמודות משמותיהם במקור הנתונים אל שמותיהם ב-DataSet. כמובן שלאחר הגדרתה, שיטה זו פועלת בשני הכיוונים: בקריאה ממקור הנתונים אל ה-DataSet ובכתיבה מה-DataSet אל מקור הנתונים.

בדוגמאות שהצגנו ואשר בהן נעשה שימוש בשיטה Fill של DataAdapter, נציין את ה-DataSet ואת שם הטבלה המתאימה שנמלא אותה בנתונים הנקראים, כגון:

```
dsEmployees.Fill(dsEmployeeInfo, "Employees")
```

עם זאת, הארגומנט השני של השיטה Fill הוא למעשה השם TableMapping. מתאם הנתונים בודק אם קיים מיפוי מוגדר עבור שם זה. אם נמצא מיפוי, הוא משתמש במידע כדי להשלים את פעולת Fill. אם לא קיים מיפוי מתאים, הוא יוצר טבלה בשם הפרמטר שהועבר וממלא אותה בנתונים.

כלומר, אם נוסיף שורת קוד עבור מיפוי MappingName עבור הטבלה empDataSetTable, כגון:

```
dsEmployees.TableMappings.Add("MappingName", "empDataSetTable")
```

נוכל לקרוא לשיטה Fill באמצעות שורת הפקודה:

```
dsEmployees.Fill(dsEmployeeInfo, "MappingName")
```

פעולה זו תגרום לנתונים שנקראו להירשם בטבלה empDataSetTable ב- DataSet הנקרא dsEmployeeInfo.

לאחר הגדרת מיפוי הטבלה, נוכל להוסיף לה מיפויי עמודות. גישה זו שימושית בעיקר כאשר קוד היישום אמור להשתמש בשמות עמודות שונים מאלה שבמקור הנתונים. כאשר משתמשים במיפוי טבלה למילוי מסד נתונים, מתאם הנתונים יחפש מיפויי עמודות עבור אותו מיפוי טבלה וישתמש בהם למיפוי שמות העמודות. עמודות שאין להן הגדרת מיפוי עמודה, ישתמשו בשמות העמודה ממקור הנתונים כשמות עבור ה-DataSet.

לדוגמה, האם תהית באיזו שנה נוצר מסד נתוני הדוגמה של SQL Server בשם pubs, ומה היו מגבלות השמות אשר בגללן הוגדרו בו שמות קצרים? מיפוי עמודות מאפשר טעינה של טבלאות מסד הנתונים pubs אל ה-DataSet שלנו עם שמות עמודות קריאים יותר. אם עליך להשתמש בשמות עמודות שונים מהשמות המקוריים, תוכל להשתמש במיפוי שמות.

נמשיך בדוגמת מיפוי הטבלה שלנו. בנוסף למיפוי הטבלה, ברצוננו למפות את כל שמות העמודות באופן שיתאימו לדרישת מנהל עיצוב התוכנה, אשר קובע כי כל שמות האובייקטים יתחילו בקידומת בת שלוש אותיות, המציינת למי הם שייכים. קוד המיפוי שלנו יהיה כפי שמוצג בקוד 5.4.

קוד 5.4-VB מיפוי שמות עמודות וטבלאות

```
' Configure Table and Column mappings
daEmployees.TableMappings.Add("MappingName", "empDataSetTable")
With daEmployees.TableMappings("MappingName").ColumnMappings
    .Add("ID", "empEmployeeID")
    .Add("FirstName", "empFirstName")
    .Add("LastName", "empLastName")
    .Add("DepartmentID", "empDepartmentID")
    .Add("Salary", "empSalary")
End With

daEmployees.Fill(dsEmployeeInfo, "MappingName")
```

קוד #C-5.4 מיפוי שמות עמודות וטבלאות

```
// Configure Table and Column mappings
daEmployees.TableMappings.Add("MappingName", "Employees");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("ID",
"empEmployeeID");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("FirstName",
"empFirstName");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("LastName",
"empLastName");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("DepartmentID",
"empDepartmentID");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("Salary",
"empSalary");

daEmployees.Fill(dsEmployeeInfo, "MappingName");
```

בפרק 4 (קודים 4.1 ו-4.2), כתבנו פונקציה בשם ReadData אשר מילאה DataSet בנתונים מטבלת מסד נתונים, ולאחר מכן הציגה את תוכן ה-DataSet בתיבת רשימה ב-fmDataSets. אם נגרום ל-btnDataAdapterFill_Click לקרוא לגירסה חדשה של הפונקציה ReadDataMapped, אשר מכילה את קוד 5.5, נוכל להפעיל את הפרויקט DataSetCode ולצפות בתוצאות מיפויי העמודות והטבלאות. תוצאות אלו מוצגות בתרשים 5.1.

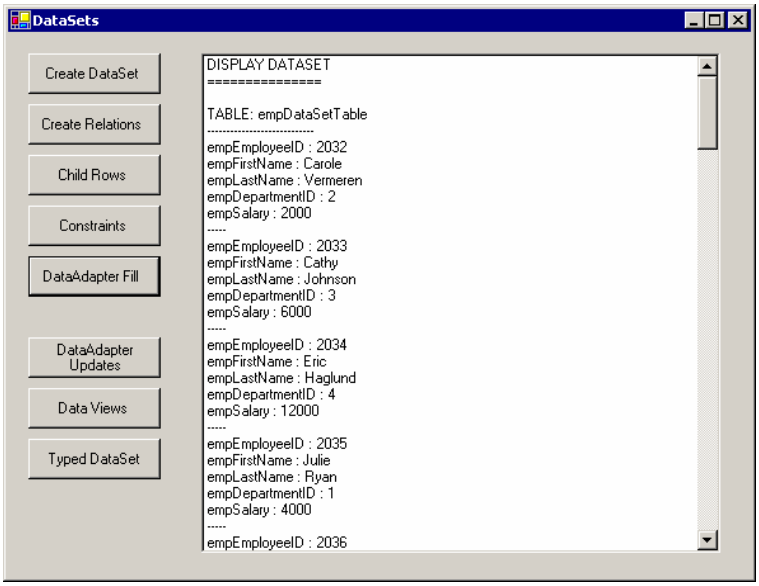
קוד 5.5-VB ReadData אשר שונתה לשימוש במיפויי טבלאות ועמודות

```
Private Sub ReadDataMapped()
        Dim daEmployees As SqlDataAdapter = _
         New SqlDataAdapter("SELECT * FROM tblEmployee", _
         "server=localhost;uid=sa;database=novelty")

        dsEmployeeInfo = New DataSet

        ' Configure Table and Column mappings
        daEmployees.TableMappings.Add("MappingName", "empDataSetTable")
        With daEmployees.TableMappings("MappingName").ColumnMappings
            .Add("ID", "empEmployeeID")
            .Add("FirstName", "empFirstName")
            .Add("LastName", "empLastName")
            .Add("DepartmentID", "empDepartmentID")
            .Add("Salary", "empSalary")
        End With

        daEmployees.Fill(dsEmployeeInfo, "MappingName")

        DisplayDataSet(dsEmployeeInfo)

    End Sub
```

קוד 5.5-C# ReadData אשר שונתה לשימוש במיפויי טבלאות ועמודות

```
private void ReadDataMapped() {
   SqlDataAdapter daEmployees = new  SqlDataAdapter("SELECT * FROM
tblEmployee", "server=localhost;uid=sa;database=novelty");
    DataSet dsEmployeeInfo =  new DataSet();
    // Configure Table and Column mappings
    daEmployees.TableMappings.Add("MappingName", "Employees");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("ID",
"empEmployeeID");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("FirstName",
"empFirstName");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("LastName",
"empLastName");
```

```
daEmployees.TableMappings["MappingName"].ColumnMappings.Add("DepartmentID",
"empDepartmentID");

daEmployees.TableMappings["MappingName"].ColumnMappings.Add("Salary",
"empSalary");

   daEmployees.Fill(dsEmployeeInfo, "MappingName");
   DisplayDataSet(dsEmployeeInfo);

}
```



תרשים 5.1 הצגת תוכן ה-DataSet בשם dsEmployeeInfo על ידי מיפויי טבלאות ועמודות.

הערה

שם מיפוי ברירת המחדל של הטבלה נקרא Table. במיפוי זה משתמשים רק אם צוין שם ה-DataSet בקריאה לשיטה Fill (או Update). לכן, טבלה אשר ממולאת באמצעות קריאה כזו ל-Fill תיקרא בשם Table ב-DataSet. עם זאת, תוכל להגדיר בצורה מפורשת מיפוי בשם Table ולציין את שם הטבלה הרצוי. לכן, שורות הקוד הבאות ייצרו וימלאו את הטבלה MyTableName:

```
dsEmplyees.TableMappings.Add("Table", "MyTableName")
daEmployees.Fill(dsEmployeeInfo)
```

טבלאות מדומות – DataViews

האובייקט DataView מאפשר ליצור תצוגות שונות של הנתונים המאוחסנים בטבלאות הנתונים מסוג DataTable, וגם ליצור בו-זמנית מספר תצוגות של אותם הנתונים. כלומר, אנו יכולים ליצור טבלה מדומה, אשר אינה קיימת באופן פיסי ומשמשת לתצוגת נתונים בלבד. ל-DataView יש תכונות המאפשרות התאמה של הנתונים הנחשפים, על פי מספר גורמים:

- **סדר מיון** – עמודה אחת או יותר, בסדר עולה או יורד.
- **ביטוי מסנן שורה** – ביטוי המציין קריטריון לחשיפת השורות, על פי ערכי עמודות.
- **מסנן מצב שורה** – ביטוי המציין קריטריון לחשיפת שורות, על פי מצב השורה (ראה רשימת ערכים DataViewRowState בטבלה 3.3).

נראה כי שיטה זו דומה לשיטה Select של ה-DataTable, אך קיים ביניהן הבדל חשוב. מצד אחד, DataView היא תצוגה דינמית מלאה של הנתונים. בנוסף לשינויים בערכי עמודות, תוספות ומחיקות של שורות בטבלה מופיעות באופן מיידי בנתונים שנחשפים על ידי ה-DataView. מצד שני, השיטה Select מחזירה מערך באורך קבוע של התייחסויות לשורות, אשר משקף שינויים שבוצעו בערכי עמודות בטבלה אך אינו משקף שינויים של הוספה, מחיקה או מיון. ההתייחסות הדינמית ל-DataView מתאימה במיוחד לתרחישי כריכת נתונים (data binding).

הערה

הטבלה המדומה, ה-DataView, דומה לתצוגה קלאסית של מסד נתונים, אך היא שונה ממנה:

- לא ניתן להשתמש בה לפעולות שנעשות בטבלה רגילה.
- אינה מאפשרת צירוף (join) עם טבלה או טבלאות אחרות.
- אינה יכולה לבטל בחירה של עמודות שקיימות בטבלה.
- לא ניתן להוסיף לה עמודות (לדוגמה, עמודות חישוב) שאינן קיימות בטבלה.

תוכל להתחיל להשתמש מייד בתצוגות, באמצעות טבלה מדומה על ידי גישה לתכונה DefaultView של ה-DataTable. אם ברצונך ליצור תצוגה עבור הטבלה Customers, אשר חושפת לקוחות ששם המשפחה שלהם מתחיל באות 'C' בלבד ולמיין את תצוגת הנתונים לפי מיקוד, עליך להגדיר את שני ערכי התכונות המתאימות:

```
dsCustomers.Tables("Customers").DefaultView().RowFilter =
    "LastName = 'Like C*'"
dsCustomers.Tables("Customers").DefaultView().Sort = "Zip"
```

אם לחילופין, תרצה לחשוף את הערכים הנוכחיים של השורות שהשתנו בלבד (אך לא נשמרו עדיין), תוכל לאפס את התכונה RowFilter ולקבוע את ערך התכונה RowState מחדש:

```
dsCustomers.Tables("Customers").DefaultView().RowFilter = " "
dsCustomers.Tables("Customers").DefaultView().RowStateFilter = _
   DataViewRowState.ModifiedCurrent
```

הערה

הטבלה המדומה, DataView, גם מכילה את השיטה Find אשר מחפשת שורה יחידה ואת השיטה FindRows אשר מחפשת ומחזירה מספר שורות. שיטות אלו נעזרות בהגדרות הנוכחיות של התכונה Sort שבמפתח. אם תרצה לאחזר את השורה או את ערכת השורות המתאימות לקריטריון במקום להשתמש בתצוגה דינמית של הנתונים, השימוש בשיטה Find או FindRows (במקום להגדיר את התכונה RowFilter) יחזיר רק את השורות הרצויות. כמו כן, לשיטה זו יש ביצועים משופרים מאלה של הגדרת התכונה RowFilter. הסיבה לכך היא שהגדרת התכונה RowFilter גורמת לבנייה מחדש של אינדקס התצוגה, ואילו השיטות Find ו-FindRows משתמשות באינדקס הקיים.

ניתן ליצור טבלאות מדומות נוספות עבור טבלה כלשהי רצויה. אם ברצונך להגדיר תצוגה נוספת בטבלה Customers, תוכל ליצור אותה ולהגדיר את התכונות הבאות:

```
dvView2 = New DataView(dsCustomers.Tables("Customers"), " ", "LastName", _
   DataViewRowState.CurrentRows)
```

לאחר יצירת התצוגה, תוכל לשנות את הגדרותיה:

```
dvView2.RowFilter = "LastName > 'F' "
dvView2.Sort = "LastName DESC"
dvView2.RowStateFilter = DataViewRowState.Current
```

הערה

קיים גם האובייקט DataViewManager, אשר מציע דרך נוחה ומרכזית לניהול הגדרות עבור תצוגות ברירת המחדל של כל הטבלאות ב-DataSet.

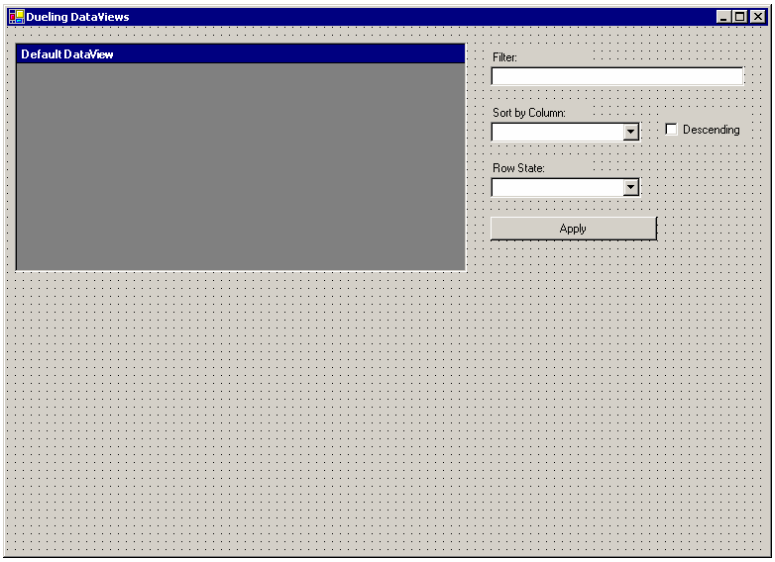
מבחינות רבות, הטבלה המדומה – DataView, דומה לטבלת נתונים רגילה – DataTable. הגישה בנפרד לערכי שורות ועמודות בתצוגה מתבצעת באמצעות האובייקט DataRowView. אובייקט זה תומך גם בניווט קשרים (navigating relations) שהוגדרו בין טבלאות ה-DataSet.

לטבלה המדומה יש מודל עריכה דומה לזה של טבלת נתונים רגילה. לאחר שאפשרת את העריכה על ידי הגדרת ערכי התכונות AllowNew, AllowEdit או AllowDelete ל-True, תוכל לבצע פעולות עריכה מתאימות. השיטות BeginEdit, EndEdit ו-CancelEdit של ה-DataRowView שולטות ביישום השינויים בטבלת הנתונים. השיטה EndEdit מכניסה

את השינויים לגרסת השורה הנוכחית (Current) של שורת הנתונים, ה-DataRow המתאימה שב-DataRowView. לאחר מכן, שינויים אלה מתקבלים (או נדחים) על ידי אובייקט הטבלה כאשר מתבצעת קריאה ל-AcceptChanges (או RejectChanges).

כעת נבחן תפיסות אלו בפעולה. נוסיף טופס חדש לפרויקט DataSetCode, אשר יספק שתי תצוגות שונות של טבלת Customers יחידה. עבור כל שורה, DataGrid יציג את תצוגת הנתונים וגם פקדים אחדים לציון תכונות המיון והסינון של התצוגה. בצע את הפעולות שלהלן כדי לקבל את התוצאה המבוקשת:

1. בטופס frmDataSets, הוסף לחצן מתחת ללחצן DataAdapterUpdates.
2. קרא ללחצן החדש **btnDataViews**, וקבע את ערך התכונה Text שלו ל- **Data Views**.
3. הוסף טופס frmDataViews לפרויקט DataSetCode.
4. בחלון Properties של frmDataViews, קבע את ערך התכונה Text ל- **Dueling DataViews**.
5. הגדל את frmDataViews.
6. הוסף DataGrid בשם **DataGrid1**, תיבת טקסט בשם **txtFilter1**, ComboBox בשם **cboSort1**, תיבת סימון בשם **chkDesc1**, ComboBox בשם **cboRowState1**, לחצן בשם **btnApply1** ושלוש תוויות בחלק העליון של הטופס frmDataViews.
7. קבע את ערך התכונה Text של תיבת הסימון ל-**Descending** ואת ערך התכונה Text ל-**Apply**. קבע את ערך התכונה DropDownStyle של שתי תיבות ה-ComboBox ל-**DropDownList**. קבע את ערך התכונה Text של התוויות ל-**Filter:**, **Sort by Column:** ו- **Row State**.
8. קבע את ערך התכונה CaptionText של DataGrid1 ל- **Default DataView**.
9. סדר את הפקדים כפי שמוצג בתרשים 5.2.

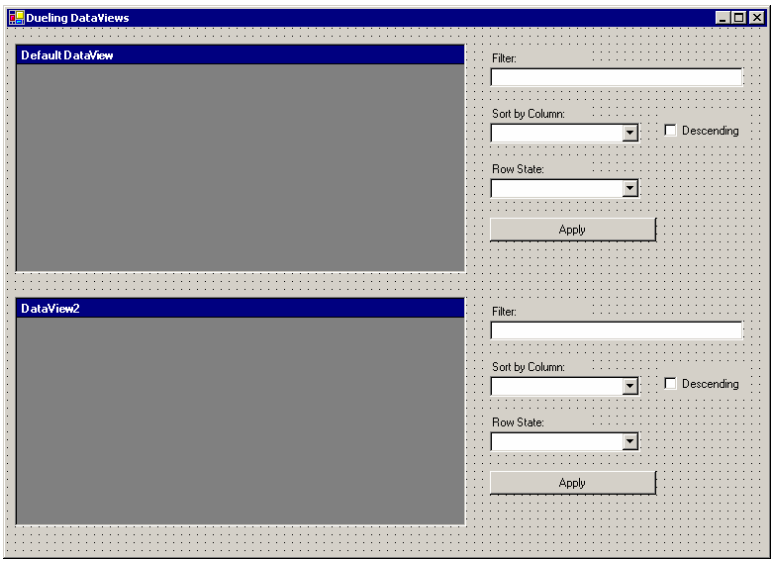


תרשים 5.2 סידור הפקדים בחלק העליון של frmDataViews.

10. בחר בכל הפקדים והעתק אותם לחלק התחתון של frmDataViews. שנה את שמות כל הפקדים באופן שיסתיימו בספרה 2 ולא ב-1, כגון btnApply2.

11. קבע את ערך התכונה Caption Text של DataGrid2 ל-**DataView2**.

העיצוב הסופי של הטופס מוצג בתרשים 5.3, והקוד עבור טופס זה מוצג בקוד 5.6.



תרשים 5.3 עיצוב סופי של frmDataViews.

קוד 5.6-VB קוד ליצירת שתי תצוגות שונות של רשת עבור אותה טבלת נתונים

```
Private dsCustomers As New DataSet()
Private dvView2 As DataView

Private Sub frmDataViews_Load(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles MyBase.Load
      Dim i As Integer
      Dim col As DataColumn

      ' Initialize DataAdapter
      Dim daCustomers As SqlDataAdapter = New SqlDataAdapter( _
       "SELECT * FROM tblCustomer", _
       "server=localhost;uid=sa;database=novelty")

      ' Fill only ONE table
      daCustomers.Fill(dsCustomers, "Customers")

      ' create second DataView
      dvView2 = New DataView(dsCustomers.Tables("Customers"), _
       " ", "LastName", DataViewRowState.CurrentRows)

      ' Fill list of column names
      For Each col In dsCustomers.Tables("Customers").Columns
         cboSort1.Items.Add(col.ColumnName)
         cboSort2.Items.Add(col.ColumnName)
      Next

      ' Fill list of DataViewRowState enumeration
      Dim names As String()
      names =
DataViewRowState.None.GetNames(DataViewRowState.None.GetType)
      For i = 0 To names.GetUpperBound(0)
         cboRowState1.Items.Add(names(i))
         cboRowState2.Items.Add(names(i))
      Next

      ' set to default values
      txtFilter1.Text = ""
      txtFilter2.Text = ""
      cboSort1.SelectedItem = "ID"
      cboSort2.SelectedItem = "ID"
      chkDesc1.Checked = False
      chkDesc2.Checked = False
```

```
    cboRowState1.SelectedItem = "CurrentRows"
    cboRowState2.SelectedItem = "CurrentRows"

    dsCustomers.Tables("Customers").DefaultView.Sort = "ID"
    dvView2.Sort = "ID"

    ' Bind grids to table
    DataGrid1.DataSource = dsCustomers.Tables("Customers").DefaultView
    DataGrid2.DataSource = dvView2

End Sub

Private Sub btnApply1_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnApply1.Click
    Dim sort As String
    Dim rowState As DataViewRowState

    ' Set Filter
    dsCustomers.Tables("Customers").DefaultView.RowFilter = txtFilter1.Text

    ' Set sort
    sort = cboSort1.SelectedItem
    If chkDesc1.Checked Then
        sort = sort & " DESC"
    End If
    dsCustomers.Tables("Customers").DefaultView.Sort = sort

    ' Set row state
    dsCustomers.Tables("Customers").DefaultView.RowStateFilter = _
            rowState.Parse(rowState.GetType, cboRowState1.SelectedItem)
End Sub

Private Sub btnApply2_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnApply2.Click
    Dim sort As String
    Dim rowState As DataViewRowState

    ' Set Filter
    dvView2.RowFilter = txtFilter2.Text

    ' Set sort
    sort = cboSort2.SelectedItem
    If chkDesc2.Checked Then
        sort = sort & " DESC"
    End If
```

```
    dvView2.Sort = sort

    ' Set row state
    dvView2.RowStateFilter = rowState.Parse(rowState.GetType, _
      cboRowState2.SelectedItem)
  End Sub
```

קוד #C-5.6 קוד ליצירת שתי תצוגות שונות של רשת עבור אותה טבלת נתונים

```
 private DataView dvView2;
private DataSet dsCustomers = new DataSet();
private void frmDataViews_Load(object sender, System.EventArgs e)
        {

          // Initialize DataAdapter
          SqlDataAdapter daCustomers = new SqlDataAdapter
            ("SELECT * FROM tblCustomer",
             "server=localhost;uid=sa;database=novelty");

          // Fill only ONE table
          daCustomers.Fill(dsCustomers, "Customers");

          // create second DataView
          dvView2 = new DataView(dsCustomers.Tables["Customers"],
             "", "LastName", DataViewRowState.CurrentRows);

         // Fill list of column names
           foreach (DataColumn col in dsCustomers.Tables["Customers"].Columns)
           {
               cboSort1.Items.Add(col.ColumnName);
               cboSort2.Items.Add(col.ColumnName);
           }

           // Fill list of DataViewRowState enumeration
           String []names;
           names = Enum.GetNames(DataViewRowState.None.GetType());
            for (int i =0; i<names.Length;++i)

           {
              cboRowState1.Items.Add(names[i]);
              cboRowState2.Items.Add(names[i]);
           }
```

```
 // set to default values
 txtFilter1.Text = "";
 txtFilter2.Text = "";
 cboSort1.SelectedItem = "ID";
 cboSort2.SelectedItem = "ID";
 chkDesc1.Checked = false;
 chkDesc2.Checked = false;
 cboRowState1.SelectedItem = "CurrentRows";
 cboRowState2.SelectedItem = "CurrentRows";

 dsCustomers.Tables["Customers"].DefaultView.Sort = "ID";
 dvView2.Sort = "ID";

 // Bind grids to table
 DataGrid1.DataSource = dsCustomers.Tables["Customers"].DefaultView;
 DataGrid2.DataSource = dvView2;

}

private void btnApply1_Click(object sender, System.EventArgs e)
{
 String sort;
 DataViewRowState rowState;

 // Set Filter
 dsCustomers.Tables["Customers"].DefaultView.RowFilter = txtFilter1.Text;

 // Set sort
 sort = cboSort1.SelectedItem.ToString();
if (chkDesc1.Checked == true)
  {
     sort +=  " DESC";
  }

 dsCustomers.Tables["Customers"].DefaultView.Sort = sort;

 // Set row state
 dsCustomers.Tables["Customers"].DefaultView.RowStateFilter =

(System.Data.DataViewRowState)Enum.Parse(DataViewRowState.None.GetType(),
cboRowState1.SelectedItem.ToString(), true);
```

```
}

private void btnApply2_Click(object sender, System.EventArgs e)
{
  string sort;

  // Set Filter
  dvView2.RowFilter = txtFilter2.Text.ToString();

  // Set sort
  sort = cboSort2.SelectedItem.ToString();
  if ( chkDesc2.Checked) {
     sort +=  " DESC";
   }

  dvView2.Sort = sort;

  // Set row state
  dvView2.RowStateFilter =

(System.Data.DataViewRowState)Enum.Parse(DataViewRowState.None.GetType(),
cboRowState2.SelectedItem.ToString(), true);
}
```

השגרה frmDataViews_Load מאתחלת את האובייקטים השונים שבטופס. האובייקט DataAdapter נוצר ולאחר מכן משמש לטעינת נתוני tblCustomer לטבלה Customers של ה-DataSet. ניצור תצוגה חדשה dvView2 עבור שתי הטבלאות המדומות, ונשתמש בתצוגת ברירת המחדל עבור האחרות. לאחר מכן, dvView2 מאותחלת כדי לחשוף את כל השורות הנוכחיות ולמיין אותן על פי LastName.

כעת מאותחלות שתי ערכות תיבות הסימון. פקדי cboSort נטענים ברשימת שמות העמודות של הטבלה Customers. פקדי cboRowState ממולאים ברשימת ערכי המנייה DataViewRowState.

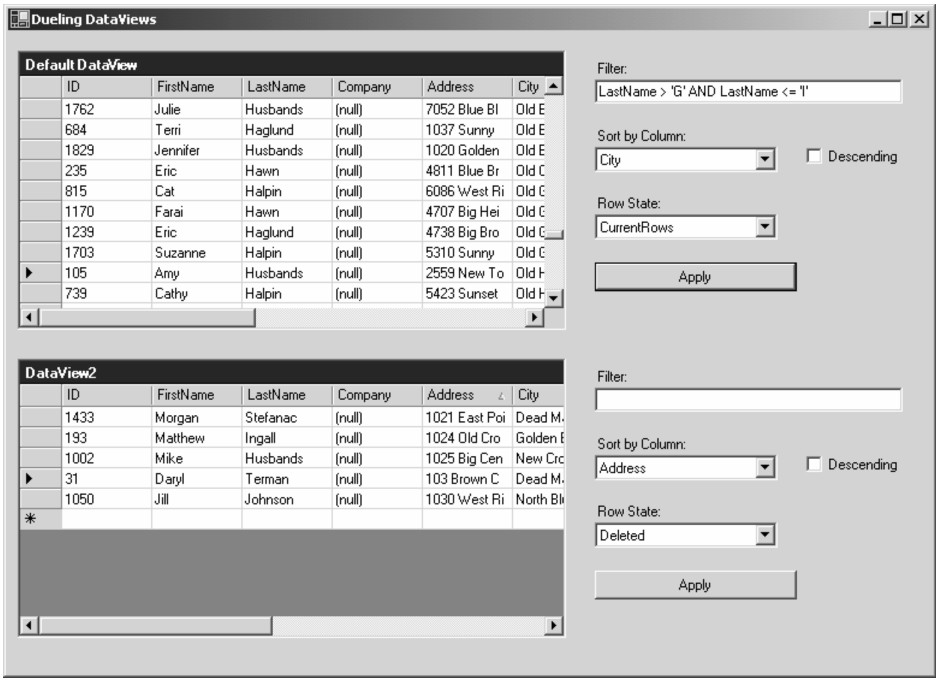
הערה

ב- Visual Basic.NET לא קיימת יותר תמיכה בתכונה ItemData. לכן, השתמשנו בשיטה GetNames של רשימת ערכי המנייה (enum) להמרה ממספר אל מחרוזת, בעת הטעינה של התיבות המשולבות. בדומה לכך, השיטה Parse של המנייה משמשת להמרה ממחרוזת לערכים מספריים, כאשר נקצה לאחר מכן את הערכים הנבחרים לתכונה RowStateFilter.

לאחר מכן, הגדרות ברירת מחדל מוקצות לפקדי הקריטריון וגם סדר מיון ראשוני עבור שתי התצוגות, לפי השדה ID. לבסוף, כל אחת משתי התצוגות נכרכת (binding) לאחד מה-DataGrids. פעולה זו גורמת להצגת כל הנתונים הנוכחיים בשתי הרשתות.

כאשר נלחץ על הלחצן Apply, הגדרות הקריטריון שנבחרו מוחלות על התצוגה המתאימה. שתי השגרות btnApply1_Click ו-btnApply2_Click זהות, פרט לכך שהן מטפלות בערכה שונה של פקדים. RowFilter נקבע מהטקסט שבתיבת הטקסט txtFilter; התכונה Sort נקבעת מהעמודה שנבחרה בתיבה המשולבת (בתוספת אפשרית של DESC להגדיר מיון בסדר יורד); ו-RowStateFilter נקבע מהערך הנבחר בתיבה המשולבת (ComboBox). שינוי של תכונות בתצוגות הקשורות לרשתות גורם להצגת נתונים ברשת על פי הגדרות התצוגה החדשות, באופן אוטומטי.

כעת תוכל להפעיל את הפרויקט DataSetCode. לחץ על הלחצן **Data Views**, אשר מציג טופס חדש בשם frmDataViews. בצע כרצונך שינויים בקריטריוני הרשתות. ודא שלחצת על Apply כדי להחיל את השינויים ברשתות המתאימות. תרשים 5.4 מציג תצוגה לדוגמה.



תרשים 5.4 תצוגת דוגמה של תוצאות בטופס frmDataViews.

נסה "לשחק" עם התיבות המשולבות ונסה למיין על פי עמודות שונות והצגה על פי מצבי שורה שונים. נסה מסננים שונים והשתמש גם בביטויים מורכבים, כגון: "ID > 10 and ID <=18" או "LastName Like 'c*" (אל תקליד בתיבת הטקסט את הגרשיים!). למידע נוסף אודות כללי הכתיבה של ביטויים עבור המסנן, פנה לתיעוד העזרה של Visual Studio בנושא DataColumn.Expression Property.

בנוסף, הרשתות תומכות באופן אוטומטי בהוספה, שינוי ומחיקת שורות. לכן, נסה לערוך שורות ולבחור במצב שורה מתאים (לדוגמה, Added, ModifiedCurrent או Deleted), כדי להציג את השורות שהשתנו בלבד.

הערה

ראה כיצד שתי התצוגות שברשת מראות למעשה את אותה הטבלה הבסיסית. אם תוסיף, תשנה או תמחק שורה בתצוגה אחת (ותקבל את השינויים על ידי מעבר לשורה אחרת), השינוי יופיע באופן אוטומטי בתצוגה השנייה (אלא אם השורה מסוננת).

תסריט עסקי 5.1:
הצגת נתונים משולבים ממקורות שונים

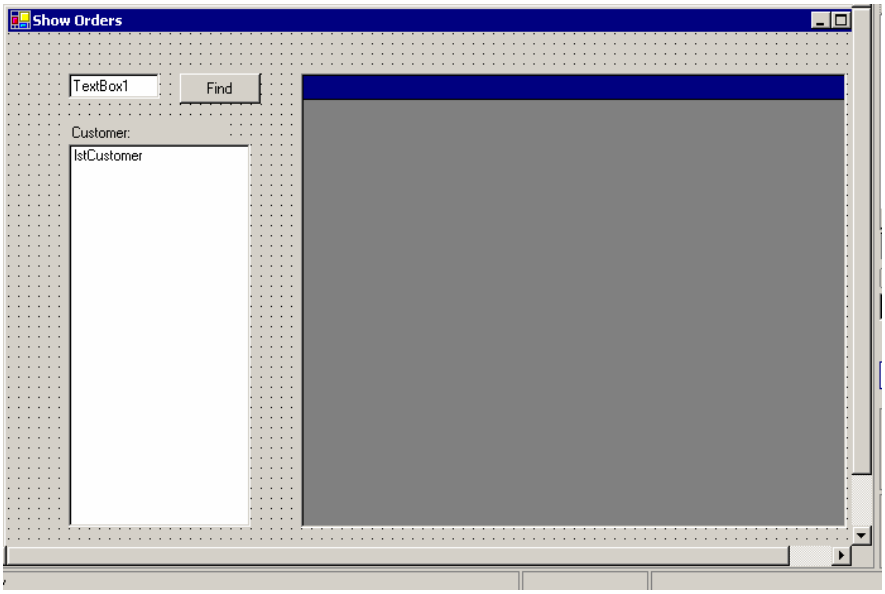
כאשר חברת Jones Novelties החליטה ליצור מערכת חדשה לעיבוד נתונים, כבר היו בידה חלקים ממנה בצורות שונות. לדוגמה, למנכ"ל Brad Jones היו נתוני הזמנות במסד נתוני Access; הוא הבין את היתרונות שבפיתוח מערכת חדשה המבוססת על מסד נתונים של SQL Server, אך רצה לוודא שהנתונים לא יאבדו בשלב ההגירה. לפי דעתו, חשוב לערוך את השינויים בצורה הדרגתית, במיוחד כאשר מדובר בנתונים קריטיים לחברה כמו נתוני הזמנות היסטוריים.

Jones רצה לפתח מערכת חדשה ב- SQL Server, אך עם זאת להמשיך ולהשתמש במסד נתוני Access שלו עד לסיום המעבר. במצב כזה, נתונים מסוימים צריכים להישמר במסד נתונים אחד ונתונים אחרים – במסד נתונים אחר. על שני מסדי הנתונים לשתף ולאחד נתונים, למרות שהם לא מאותו הסוג!

למרבה המזל, דרישה זו אינה מורכבת מדי עבור ADO.NET. מבחינת האובייקט DataSet, לא איכפת לו והוא גם אינו מודע למקור הנתונים הכלולים בטבלאות שלו. דבר זה נכון גם אם הנתונים מגיעים ממקורות שונים. מי שפיתח את מסד הנתונים של Jones יכול לפתח את היישום על ידי טעינת הטבלה tblOrder ממסד נתוני Access ואת הטבלה tblCustomer ממסד נתוני SQL Server. בעתיד, כאשר הטבלה tblCustomer הנמצאת ב- SQL Server תהיה מוכנה לשימוש, יהיה צורך לשנות את מחרוזת ההתחברות שנמצאת באובייקט DataAdapter, באופן שתטען את tblCustomer. שאר הפעולות ימשיכו להתבצע ללא שינוי.

כעת ניצור טופס אשר מדגים כיצד מפתח הנתונים בחברת Jones יכול להגיע לתוצאה זו. לשם כך, בצע את הפעולות שלהלן:

1. הפעל את Visual Studio.NET.
2. צור פרויקט Visual Basic Windows Application חדש.
3. קרא לפרויקט בשם **BusinessCase71**.
4. ציין נתיב לשמירת קבצי הפרויקט.
5. הגדל את Form1.
6. בחלון Properties של Form1, קבע את ערך התכונה Name ל-**frmShowOrders** ואת ערך התכונה Text ל- **Show Orders**.
7. הוסף תיבת טקסט txtCustID, לחצן btnFind, תיבת רשימה lstCustomer ו-DataGrid בשם **grdOrders**. קבע את ערך התכונה Text של הלחצן ל-**Find**.
8. סדר את הפקדים כפי שמוצג בתרשים 5.5.



תרשים 5.5 סידור פקדים בטופס frmShowOrders.

9. מתיבת הכלים, הוסף רכיב DataSet מסוג Untyped DataSet וקבע את שמו ל-**dsCustOrders**.

10. הוסף SqlDataAdapter ל-tblCustomer, אשר נמצאת במסד הנתונים Novelty של SQL Server. כאשר יוצג Configuration Wizard, בחר בהתחברות למסד נתוני Novelty אשר בשימוש בפרק זה. ב- Query Type, בחר **Use SQL Statements**.

11. השתמש ב- SELECT * FROM tblCutomer כפקודת SQL לטעינת נתונים ל-DataSet.

12. בסיום Configuration Wizard, שנה את השם SqlDataAdapter ל-**daCustomers**. Configuration Wizard יציב גם רכיב SqlConnection (עם הגדרות מתאימות) במגש הרכיבים.

13. לטבלה tblOrder, אשר נמצאת במסד נתוני Access בשם Novelty.mdb, הוסף OledbDataAdapter. כאשר Configuration Wizard יחל לפעול, הוסף התחברות. לאחר לחיצה על New Connection והצגת תיבת הדו-שיח Data Link בעלת הכרטיסיות, בחר בכרטיסיה **Provider** ובחר באפשרות **Microsoft Jet 4.0 OLE DB Provider**.

14. בחר בכרטיסיה **Connection**, הזן את שם קובץ מסד הנתונים Novelty.MDB או אתר אותו.

15. בחר באפשרות **Use SQL Statements** עבור Query Type.

16. בחר באפשרות **SELECT * FROM tblOrder** כפקודת SQL לטעינת נתונים ל-DataSet.

17. בסיום הפעולה של Configuration Wizard, שנה את השם OledbDataAdapter ל-**daOrders**. אשף ההתקנה יציב גם רכיב OledbConnection (עם הגדרות מתאימות) במגש הרכיבים.

לפניך קטע הקוד הראשון שיש להוסיף בתחילת הקובץ:

```
Imports System
Imports System.Data
Imports System.Data.SqlClinet
Imports System.Data.Oledb
```

לאחר מכן, נוסיף את קוד 5.7 לגוף הגדרת המחלקות שב-frmOrders.

קוד 5.7-VB קוד לאיחוד נתונים משני סוגים שונים של מקורות נתונים

```
Private Sub btnFind_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles btnFind.Click
    Dim RowNum As Integer
    Dim dvRow As DataRowView
    Dim i As Integer

    If IsNumeric(txtCustID.Text) Then
      RowNum = _
          dsCustOrders.Tables("Customers").DefaultView.Find(txtCustID.Text)

      If RowNum <> -1 Then
        dvRow = dsCustOrders.Tables("Customers").DefaultView(RowNum)
        ' Fill Listbox with Customer data fields
        lstCustomer.Items.Clear()
        For i = 0 To dsCustOrders.Tables("Customers").Columns.Count - 1
          lstCustomer.Items.Add(dvRow.Item(i))
        Next
        grdOrders.CaptionText = "Orders for customer #" & txtCustID.Text
        ' Get related Child rows of selected Customer
        dvOrders = dvRow.CreateChildView("relCustOrders")
        grdOrders.DataSource = dvOrders
      Else
        MessageBox.Show("CustomerID not found - Please try again.")
        txtCustID.Clear()
      End If
    Else
      Beep()
    End If
  End Sub
```

```
Private dvOrders As New DataView()
  Private Sub frmShowOrders_Load(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles MyBase.Load
      Dim rel As DataRelation

      ' Fill the DataSet table with data from database
      daCustomers.Fill(dsCustOrders, "Customers")
      daOrders.Fill(dsCustOrders, "Orders")

      ' Create relation between the tables
      rel = dsCustOrders.Relations.Add("relCustOrders", _
        dsCustOrders.Tables("Customers").Columns("ID"), _
        dsCustOrders.Tables("Orders").Columns("CustomerID"))

      ' Set Primary Key on Customers table
      Dim pk(0) As DataColumn
      pk(0) = dsCustOrders.Tables("Customers").Columns("ID")
      dsCustOrders.Tables("Customers").PrimaryKey = pk

      ' Set Default Sort to allow Find method to work
      dsCustOrders.Tables("Customers").DefaultView.Sort = "ID"

    End Sub
```

קוד #C-5.7 קוד לאיחוד נתונים משני סוגים שונים של מקורות נתונים

```
private DataView dvOrders = new DataView();
private void btnFind_Click(object sender, System.EventArgs e)
 {
      int RowNum;
      DataRowView dvRow;

      RowNum = dsCustOrders.Tables["Customers"].DefaultView.Find(txtCustID.Text);

          if (RowNum != -1)
          {
             dvRow = dsCustOrders.Tables["Customers"].DefaultView[RowNum];
             String data =  dsCustOrders.Tables["Customers"].Rows[0]["ID"].ToString();
             // Fill Listbox with Customer data fields
             lstCustomer.Items.Clear();
             for (int i=0; i<dsCustOrders.Tables["Customers"].Columns.Count;++i)
              lstCustomer.Items.Add(dvRow[i]);
              grdOrders.CaptionText = "Orders for customer #" + txtCustID.Text;
```

```
        // Get related Child rows of selected Customer
        dvOrders = dvRow.CreateChildView("relCustOrders");
        grdOrders.DataSource = dvOrders;
    }
    else
    {
        MessageBox.Show("CustomerID not found - Please try again.");
        txtCustID.Clear();
    }
}

private void frmShowOrders_Load(object sender, System.EventArgs e)
{
    DataRelation rel = null;

    // Fill the DataSet table with data from database
    daCustomers.Fill(dsCustOrders, "Customers");
    daOrders.Fill(dsCustOrders, "Orders");

    // Create relation between the tables
    rel = dsCustOrders.Relations.Add("relCustOrders",
       dsCustOrders.Tables["Customers"].Columns["ID"],
       dsCustOrders.Tables["Orders"].Columns["CustomerID"]);

    // Set Primary Key on Customers table
    DataColumn []pk = new DataColumn[1];
    pk[0] = dsCustOrders.Tables["Customers"].Columns["ID"];
    dsCustOrders.Tables["Customers"].PrimaryKey = pk;

    // Set Default Sort to allow Find method to work
    dsCustOrders.Tables["Customers"].DefaultView.Sort = "ID";
}
```

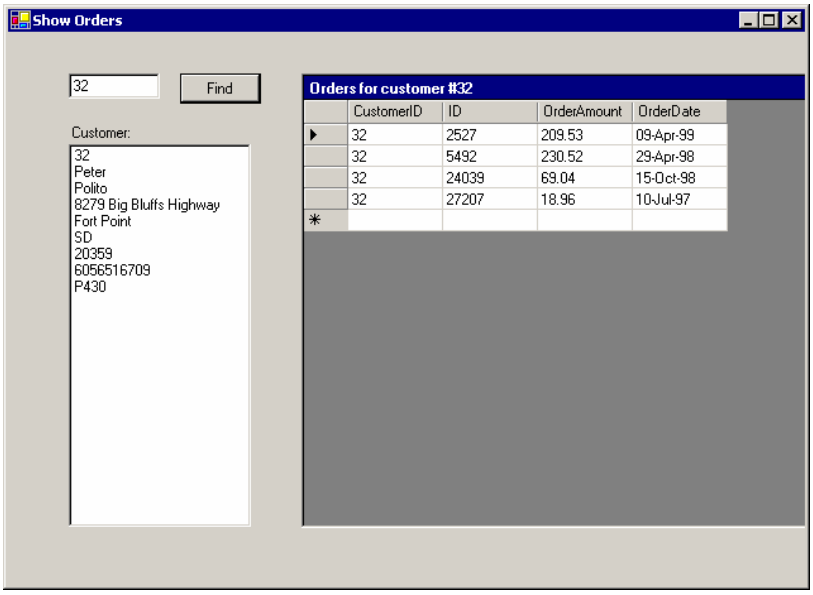
הגדרנו את כל הנחוץ בשגרה frmShowOrders_Load. לאחר מכן נמלא את שתי טבלאות ה-DataSet וניצור DataRelation כדי לקשר ביניהן. לבסוף, נקבע את המפתח הראשי, PrimaryKey, ואת תכונות המיון Sort בטבלה Customers ו-DefaultView, באופן שנוכל להשתמש בשיטה Find של התצוגה, כפי שנדגים בהמשך.

מעניין לצפות בתגובה ללחיצה על Find, אשר מיושמת בשגרה btnFind_Click. לאחר בדיקה כי הערך בתיבת הטקסט txtCustID הוא מספרי, יבוצע חיפוש של ערך זה באמצעות Customers DefaultView. אם נמצאה התאמה, כל אחת מהעמודות של התצוגה Customer DataRowView מוצגת בתיבת הרשימה. ניצור תצוגה חדשה של

שורות מהטבלה tblOrder, אשר שייכות לתצוגת השורות DataRowView שנבחרה, ונכרוך את התצוגה ל-DataGrid.

אני מקווה שאתה עוקב אחר הפעולות שנעשו. באמצעות הפעולות האחרונות, יצרת וגם ניווטת בצירוף של שתי טבלאות משני מסדי נתונים מסוגים שונים!

כעת תוכל להפעיל את הפרויקט BusinessCase7 ולבדוק כיצד נתוני הלקוח מוצגים בתיבת הטקסט, וכיצד ההזמנות הקשורות ללקוח זה מוצגות ברשת. תרשים 5.6 מכיל תצוגה לדוגמה.



תרשים 5.6 תצוגה לדוגמה של תוצאות בטופס frmShowOrders.

הערה

אל תיבהל ואל תתבלבל מכך שהטבלה Orders בתרשים 5.6 מכילה עמודה בשם OrderAmount במקום בשם Amount, כפי שהגדרנו אותה בטבלת SQL Server שלנו. זכור כי למרות שנתוני הלקוח מגיעים ממסד נתוני SQL Server, נתוני ההזמנות מגיעים ממסד נתונים "ישן" ושונה מסוג MDB. פעמים רבות תיתקל בשמות של אובייקטי מסד נתונים אשר השתנו בין גרסאות שונות של יישומים.

אם אינך מעוניין בשם OrderAmount, תוכל לשנות אותו על ידי משפט AS. משפט זה ישנה את שם העמודה בנתונים המוחזרים על ידי הצהרת SELECT עבור DataAdapter בשם daOrders, בצורה שלהלן:

```
SELECT CustomerID, ID, OrderAmount AS Amount, OrderData FROM tblOrder
```

אל תיבהל אם אתה מקבל שגיאה בתחילת ההרצה. בדוק אם כל הנתונים נמצאים בשתי הטבלאות ולא מחקת חלק מהם.

בהמשך עבודתך תיתקל בצורך לשלב נתונים ממקורות נתונים שונים. חברות יוצרות שותפויות עסקיות או התמזגויות, והנתונים עשויים להגיע במבנים שונים. במצב כזה, קידוד XML הופך לדרך קלה ונפוצה להעברת נתונים, כמו רשימת הזמנות ומוצרים בין חברות. עבור ה-DataSet, XML הוא דרך נוספת למילוי טבלאות בנתונים.

Strongly Typed DataSets

עד עתה, עסקנו במכולות של טבלאות, DataSets, חסרות סוג (untyped DataSets). עם זאת, ב-ADO.NET וגם ב- Visual Studio, נוכל ליצור מכולות בעלות סיווג מוגדר – typed DataSets. אובייקט זה נגזר מהמחלקה untyped DataSet ומתווספים לו אובייקטים, שיטות, תכונות ואירועים שייחודיים לסכמת (schema) מסד הנתונים שלך. הסכמה מוגדרת בקובץ XML Schema (קובץ XSD) ובכלי עיצוב ליצירת ערכת מחלקות המבוססות על סכמה זו.

מכיון שהאובייקט typed DataSet נגזר מהאובייקט DataSet, הוא יורש ממנו את הפונקציונליות שלו, תכונות, שיטות ואירועים, וניתן להשתמש בו בכל מקום שניתן להשתמש בו באובייקט מסוג untyped DataSet. עם זאת, מכיון שאלמנטים מסוימים מוגדרים באופן ייחודי למסד הנתונים, יש תכונות נוספות אשר משפרות את מהירות ואמינות הפיתוח. לכן, נקרא ל- typed DataSet בשם Strongly typed DataSet.

- DataSets, DataTables ו-DataRows הם אובייקטים המיוחדים לסכמה שמטפלים בה.
- DataColumns ו-DataRelations נחשפים כתכונות בעלות שם ספציפי, ולא כאלמנטים באוסף כללי (generic).
- ניתן לבצע בדיקת תקינות בזמן הידור.
- השלמת משפטים באמצעות IntelliSense ניתנת בעורך הקוד של Visual Studio.
- באופן כללי, הקוד קריא הרבה יותר.

לדוגמה, בטבלה Customers שיצרנו, הגישה אל ערך של עמודה בשורה כלשהי נעשה כך:

VB

```
daCustomers.Tables("Customers").Rows(row)("FirstName") = NewValue
```

C#

```
dsCustOrders.Tables["Customers"].Rows[0]["ID"] = NewValue;
```

במשפטים אלה, row הוא האינדקס באוסף Rows של הטבלה Customers. FirstName הוא שם העמודה שניגשים אליה, ו-NewValue הוא הערך החדש. יש כמה שגיאות אפשריות שמדווחות בזמן ריצה בלבד ולא בזמן העיצוב. האם המשתנה NewValue

הוא מהסוג הנכון אשר מתאים לסוג העמודה? האם הטבלה Customers קיימת? האם העמודה FirstName קיימת? האם שם העמודה או הטבלה נכון? שאלות אלו חלות גם בקריאת ערך עמודה.

עם זאת, אם ניצור typed DataSet בשם dsCustomers, יהיו לו תכונות ייחודיות המותאמות לסכמה שלנו, והוא יוכל לבצע את כל בדיקות הסוג הנדרשות בזמן העיצוב. להלן הקוד המתאים עבור typed DataSet עם הטבלה Customers:

VB

```
dsCustomers.Customers(row).FirstName = NewValue
```

C#

```
dsCustomers.Customers[row].FirstName = NewValue;
```

שים לב לכך שהטבלה Customers היא תכונה ספציפית של ה- typed DataSet, והעמודה FirstName היא תכונה ספציפית של typed DataTable בשם Customers. נדגים גם בקצרה כיצד IntelliSense בעורך הקוד משתמש בתכונות אלו.

כעת נחזור לטבלה Departments כדי לבדוק גישה זו. בצע תחילה את הפעולות שלהלן:

1. פתח את הפרויקט DataSetCode ב- Visual Studio.
2. בטופס frmDataSets, הוסף לחצן מתחת הלחצן Data Views.
3. קרא ללחצן החדש בשם **btnTypedDataSet** וקבע את ערך התכונה Text שלו ל- **Typed DataSet**.

כעת עליך להוסיף את הסכמה הרצויה לפרויקט שלנו. אם כבר קיים קובץ XSD, ניתן להשתמש בו. במקרה שלנו, לא קיים עדיין קובץ כזה ולכן יש ליצור אותו. תוכל לעשות זאת בקלות בעזרת רכיבים שנמצאים בכרטיסיה Data של תיבת הכלים, בצורה הבאה:

1. הוסף לטופס רכיב SqlDataADapter. באשף התצורה, השתמש בהתחברות למסד נתוני Novelty (SQL Server), השתמש ב- SELECT * FROM tblDepartment במחרוזת ההתחברות ובהגדרות ברירת המחדל עבור האפשרויות האחרות. פעולה זו תוסיף DataAdapter חדש וחיבור חדש לטופס. שנה את שם האובייקט DataAdapter ל-**daDepartments**.
2. הצג את תיבת הדו-שיח **Generate DataSet**, על ידי בחירתה מתפריט Data הראשי או על ידי בחירתה מהתפריט הנפתח, אשר מוצג באמצעות לחיצה ימנית בטופס. בחר באפשרות **New dataset**, קרא למסד הנתונים בשם **DepartmentsDS**, השאר את התיבה Add this dataset to the designer מסומנת ולחץ **OK**.
3. שנה את שם הרכיב DataSet שהוספת זה עתה ל-**dsDepartments**.

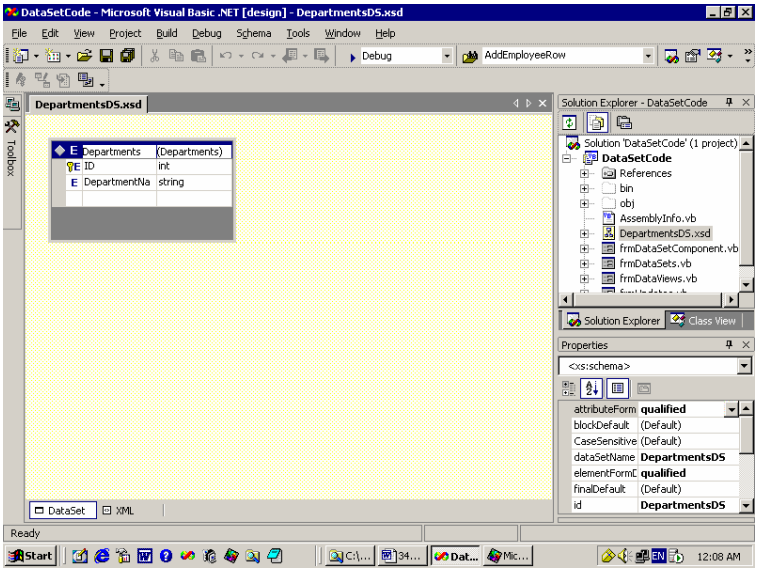
ה- Dataset Generator הוסיף גם את קובץ סכמת XML בשם DepartmentsDS.xsd אל הפרויקט ב- Solution Explorer. בנוסף, התווסף הקובץ אשר מיישם את מחלקות ה- typed DataSet המותאם בשם DepartmentsDS.vb, אל הפרויקט שבתשתית קובץ הסכמה. הקובץ DepartmentsDS.vb אינו גלוי, אלא אם תלחץ על Show All Files בראש Solution Explorer.

הערה

Generator, DataSet ו- Schema Editor מטפלים בקלות בסכמות עם טבלאות בעלות קשרים, כאילו היו טבלאות יחידות.

1. לחץ לחיצה כפולה על הקובץ **DepartmentsDS.xsd** ב- Solution Explorer, כדי להציג את XML Schema Editor. כעת תבצע שינוי של שם האלמנט בסכמה המוצגת.

2. שנה את שם האלמנט מ-tblDepartment ל-**Departments**, על ידי עריכת התא העליון השמאלי. התוצאות מוצגות בתרשים 5.7.



תרשים 5.7 הסכמה DepartmentsDS ב- Schema Editor.

3. לחץ לחיצה ימנית בשטח העיצוב של Schema Editor, כדי לוודא שהאפשרות Generate DataSet נבחרה. זוהי הגדרת ברירת המחדל. אם אפשרות זו לא נבחרה מסיבה כלשהי, בחר בה כעת.

4. שמור וסגור את Schema Editor. הקוד שבקובץ DepartmentsDS.vb ייווצר באופן אוטומטי.

הערה

תוכל לפתוח את הקובץ DepartmentsDS.vb בעורך הקוד, כדי לצפות בקוד שנוצר. הדבר החשוב שעליך לזכור הוא שעבור טבלה נתונה (אשר שינית מ-tblDepartments ל-Departments), DataSet Generator מייצר שלוש מחלקות אובייקטים

DepartmentsDataTable

DepartmentsDataRow

DepartmentsRowChangeEvent

בנוסף למחלקה type DataSet עצמה, שכבר ציינו כ-DepartmentsDS.

כעת נוכל להמשיך ולהוסיף את קוד 5.8 ל- typed DataSet בשם DepartmentsDS.

קוד 5.8-VB קוד להצגת תוכן ה- typed DataSet בשם DepartmentsDS

```
Private Sub btnTypedDataSet_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnTypedDataSet.Click

    daDepartments.Fill(dsDepartments, "Departments")

    DisplayDepartments(dsDepartments)
End Sub

Private Sub DisplayDepartments(ByVal ds As DepartmentsDS)
    Me.lstOutput.Items.Clear()
    Me.lstOutput.Items.Add("DISPLAY TYPED DATASET")
    Me.lstOutput.Items.Add("====================")

    ' Each Column is now a property of the DepartmentsDS DataSet
    Dim row As DepartmentsDS.DepartmentsRow

    For Each row In ds.Departments.Rows
        Me.lstOutput.Items.Add( _
            ds.Departments.IDColumn.ColumnName _
            & " : " & row.ID.ToString)

        Me.lstOutput.Items.Add( _
            ds.Departments.DepartmentNameColumn.ColumnName _
            & " : " & row.DepartmentName)
    Next
End Sub
```

קוד #C-5.8 קוד להצגת תוכן ה- typed DataSet בשם DepartmentsDS

```
private void btnTypedDataSet_Click(object sender, System.EventArgs e)
    {
      daDepartments.Fill(dsDepartments, "Departments");
      DisplayDepartments(dsDepartments);

    }
    private void DisplayDepartments(DepartmentsDS ds)
    {
      lstOutput.Items.Clear();
      lstOutput.Items.Add("DISPLAY TYPED DATASET");
      lstOutput.Items.Add("====================");

      // Each Column is now a property of the DepartmentsDS DataSet
      //DepartmentsDS.DepartmentsRow row;

      foreach (DepartmentsDS.DepartmentsRow row in ds.Departments.Rows)
      {
            lstOutput.Items.Add(
                  ds.Departments.IDColumn.ColumnName
                  + " : " + row.ID.ToString());

            lstOutput.Items.Add(
                  ds.Departments.DepartmentNameColumn.ColumnName
                  + " : " + row.DepartmentName);
      }
    }
```

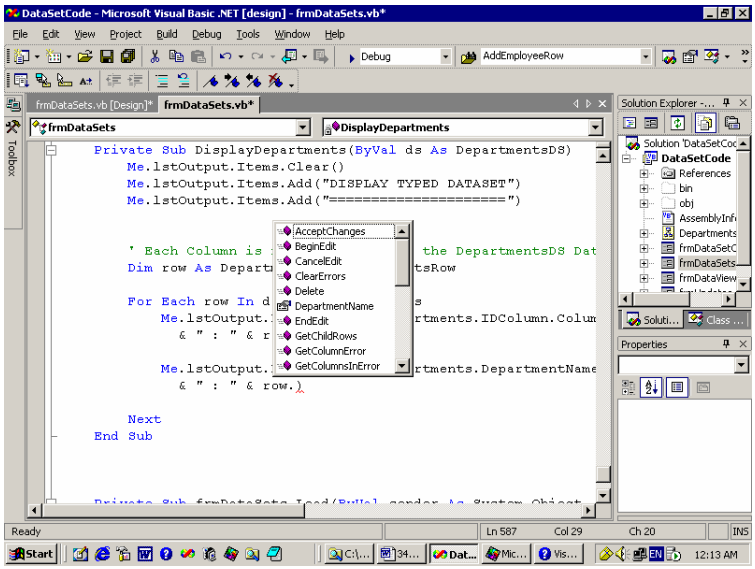
באירוע Click של btnTypedDataSet, נקרא לשיטה Fill של אובייקט DataAdapter בשם daDepartments, כדי לטעון את הטבלה Departments בנתונים. לאחר מכן נקרא לשגרה DisplayDepartments, אשר תציג את תוכן הטבלה. להצגת תוכן הטבלה, נבצע לולאה על כל השורות שבה, ונציג את שם העמודה ואת הערך עבור כל עמודה. גישה זו אינה שונה במידה ניכרת מהפעולות שביצענו קודם, אך שים לב לנקודות הבאות:

- המשתנה row מוגדר כסוג ספציפי של row, DepartmentsDS.DepartmentsRows, ולא כ-DataRow גנרי.
- הגישה לטבלה Departments מתבצעת כתכונה של DataSet, dsDepartments, ולא כפריט באוסף Tables. כלומר, ds.Tables("Departments").
- הגישה לעמודות הטבלה מתבצעת כתכונות הטבלה, ds.Departments.IDColumn ו- ds.Departments.DepartmentNameColumn, ולא כפריטים באוסף Columns.

כלומר: ds.Tables("Departments").Columns("ID")
ו- ds.Tables("Departments").Columns("DepartmentName").

- הגישה לערכי העמודה מתבצעת כתכונות השורה DepartmentsRows, row.ID ו-row.DepartmentName, ולא כפריטים באוסף Items של השורה. כלומר, row("ID") ו-row("DepartmentName").

תרשים 5.8 מדגים כיצד מוצגות תכונות מסוג מסוים (לדוגמה, DepartmentName) בתפריט הנפתח IntelliSense, אשר מסייע לכתיבת הקוד ומאיץ את כתיבתו.



תרשים 5.8 תפריט IntelliSense, המציג את תכונות typed DataSet.

סיכום

בפרק זה דנו במספר נושאים חשובים עבור המתכנת בעבודה מעשית, ואשר קשורים לפיתוח יישומי מסד נתונים ב-ADO.NET. חשוב להתייחס לשימוש בו-זמנית על ידי משתמשים אחדים ולהטמיע זאת בזמן העיצוב, במיוחד כאשר אתה עובד במצב מנותק עם DataSet. מיפויי טבלאות ועמודות הן דרכים נוחות לשיפור הקריאות ויכולת הטיפול בקוד יישום. השימוש באובייקטי DataView מאפשר שימוש במספר תצוגות של אותם הנתונים, כאשר לכל תצוגה הגדרות מיון וסינון משלה. לבסוף, אובייקטים מסוג strongly typed DataSet מאפשרים שימוש בפונקציונליות ואמינות מתקדמות, כאשר נעשה בהם שימוש במצבים מתאימים.

שאלות ותשובות

ש: כעת, לאחר שהבנתי כיצד לגלות concurrency violation, כיצד אפתור בעיה זו?

ת: התשובה לשאלה זו תלויה ביישום שאתה עוסק בו. תוכל לקבוע שבמקרה של גילוי שימוש בו-זמנית, המשתמש צריך לבצע שאילתה חדשה במסד הנתונים ולהתחיל מההתחלה. תוכל לאפשר דריסה של השינויים שבוצעו על ידי המשתמש הראשון באמצעות השינויים שביצע המשתמש השני, אך תצטרך לתעד את המקרה ביומן (או לשלוח הודעה). אפשרות שלישית היא לאפשר למשתמש לקבוע על פי הצגת שלוש ערכות הנתונים: הערכים המקוריים, הערכים אשר נמצאים כרגע במסד הנתונים והערכים שהוא מנסה להוסיף למסד הנתונים. לבסוף, תוכל לפתח אלגוריתם או קוד לוגי מסוים אשר יאפשר ליישום להחליט בעצמו איזה שינוי יש להשאיר. לדוגמה, נניח שהעדיפות או תפקיד המשתמש, כגון מפקח מול פקיד, יהיה הגורם המכריע. ADO.NET ו- Visual Studio.NET מספקים את הכלים הדרושים לגילוי concurrency violation. בכל מקרה, עליך מוטלת עדיין המשימה הקשה של פתרון הבעיה!

ש: האם עלי להשתמש תמיד ב- typed DataSets? אם לא, מהם השיקולים שיש להתחשב בהם בבחירה?

ת: כמו ברוב המקרים, התשובה היא "זה תלוי". הצגנו בפניך את יתרונות השימוש ב- typed DataSet, אך במקרים מסוימים עדיף להשתמש ב- untyped DataSet. זהו בדרך כלל המצב כאשר הסכמות הנטענות ל-DataSet אינן ידועות מראש, או שהן עשויות להשתנות. דוגמה קלאסית לכך היא שאילתות הנוצרות במהלך ההפעלה, או שאילתות שצוינו על ידי המשתמש בסוג מסוים של בונה השאילתה – query builder. ייתכן גם שתרצה לציין את הסכמה על ידי קריאת קובץ XML. במקרים אלה, לא תוכל לספק סכמה ליצירת typed DataSet בזמן העיצוב.

שיקול נוסף, או פשרה, היא הקביעה אם כתיבת שגרות גנריות היא דרישה עיקרית בפרויקט הפיתוח. היתרונות הטמונים ב- typed DataSets מתקבלים על חשבון קוד לא גנרי. תוצאה אחת של שימוש לא גנרי היא שכל שינוי שתבצע בסכמת מסד הנתונים יחייב אותך ליצור ולהדר מחדש את האובייקטים.

לבסוף, אובייקט DataSet הוא כה שימושי וקל לשימוש, עד שתרצה פעמים רבות להשתמש בו לניהול קוד שאינו מבוסס על סכמה. DataSet עשוי להוות חלופה טובה ליישום מבני נתונים מותאמים עם אלגוריתמי חיפוש, מיון וסינון מותאמים. יכולות XML הן בבחינת בונוס נוסף.

אינדקס

A

ADO 2.x see ADO
ADO 3.x see ADO
ADO vs. ADO.NET, 82
ADO, 80-83, 190
ADO.NET
    Command (object), 92-111
    Connection (object), 86-87
    Data Providers, 79-129
    DataAdapter, 179-221
    DataReader (object), 112-115
    DataSet, 131-178
    Objects within .NET Framework, 83-84
    vs. ADO, 82
ALTER PROCEDURE command, 208
ALTER TABLE command, 54
ASP.NET
    Books, 12
Authentication see SQL Server Authentication
Authentication see Windows Authentication
Authors, 14

B

Books
    SQL, 11
    SQL Server 2000, 11

C

C Sharp see C#
C#, 11
Classic ADO see ADO
Client/Server programming, 16
Column Mapping, 229-232
Command (object), 92-111
    ExcuteNonQuery, 101-109

ExcuteReader, 111
ExcuteScalar, 109-111
Using with Parameters, 95-100
Using with Stored Procedures, 95-100
CommandBuilder (object), 193-207
Common Language Runtime see CLR
Common Type System see CTS
Concurrency Conflicts, 224-228
Optimistic Concurrency Control, 224
Pessimistic Concurrency Control, 224
Configuration Wizard (DataAdapter), 209-214
Conflicts see Concurrency Conflicts
Connection (object), 86-87
Constraint (DataSet), 134
Constructors, 94, 112
Control Access to Data (View), 42
Create (Stored Procedure), 47-54
Create (Trigger), 53-54
CREATE PROCEDURE command, 50, 56, 105
CREATE TABALE command, 102
CREATE TRIGGER command, 53
Create View (SQL Query Analyzer Batch), 46-47
Create View (SQL Server Enterprise Manager), 43-45
CREATE VIEW command, 45, 105
Creating a Database (SQL Server Enterprise Manager), 31-36
Creating a Database (SQL Server Manager), 27-30
CS see C#
CTS, 83

D

Data Providers (ADO.NET), 79-129
ODBC, 85
OLEDB, 85
SqlClient, 84
Data Warehouse (MS SQL Server 2000), 18
DataAdapter, 179-221
Configuration Wizard, 209-214
Fill (method), 181-190
Update (method), 190-193
Database Roles, 61
DataColumn (DataSet), 133

DataReader (object), 112-115
DataRelation (DataSet), 134
DataRow (DataSet), 134
DataSet, 131-178
    Component, 171-177
    Constraint, 134
    DataColumn, 133
    DataRelation, 134
    DataRow, 134
    DataTable, 133
    Errors, 156-158
    Strongly Typed, 250-255
    Subsidiary objects, 132-133
    Table constraints, 165-171
    Table relations, 158-165
    Untyped, 250
    Updating, 153
DataTable, 133
    Adding Data, 141-144
    Schemas, 135-140
    Searching, 148-152
    Updating, 144-148
DDL see SQL DDL
DROP command, 67
DROP VIEW command, 46
DataRowVersion, 154
DataView (object), 233-243

E

Editions (Microsoft SQL Server 2000), 18
ExcuteNonQuery, 101-109
    vs. ExcuteScaler, 109
ExcuteScalar, 109-111
    vs. ExcuteNonQuery, 109
ExcuteReader, 111

F

Fill (DataAdapter), 181-190

G

GRANT command, 65

I
Identity column, 33
INSERT INTO command, 56
Instance, 94, 112

J
Jackie Goldstein, 14
Jeffrey P. McManus, 14
Jet, 16-17
JOIN command, 188

K
KILL command, 67

L
Login, 58

M
MSDE, 19
Managing Security in SQL Server Enterprise Manager, 57-64
    Login, 58
    SQL Server Authentication, 58-59
Managing Roles with SQL Server Enterprise Manger, 61-63
Managing Users in SQL Server Enterprise Manager, 57-64
Mapping
    Column, 229-232
    Table, 229-232
Microsoft.Data.Odbc, 13, 85
Microsoft SQL Server 2000, 15-78
    Editions, 18
    Installation, 12, 20
    System Requirements, 19
Multiuser Computing, 16

N
Namespace, 83-84
NET Framework
    ADO.NET objects, 83
    Integration with ADO.NET, 82

O

ODBC, 13, 85
OLEDB Data Provider, 85
Optimistic Concurrency Control (conflicts), 224
Overloading, 147, 207

P

Parameters (Command Object), 95-100
Permission set see Managing Security
Pessimistic Concurrency Control (conflicts), 224
Placeholder, 95
Primary Keys, 35-36, 150

R

REVOKE command, 65
Roles see Managing Roles with SQL Server Enterprise Manger
Roles
    Database, 61
    Server, 61
RowStates, 154
Running (Stored Procedure), 47-54

S

Scalar Value, 109
Schemas (DataSet), 135-140
Security see Managing Security
Security
    Applying Security in SQL Query Analyzer, 64-67
    Testing Security in SQL Query Analyzer, 63-64
SELECT command, 38
Server Roles, 61
SET command, 55
Source Code of the book, 12
SP see Stored Procedure
sp_addlogin, 64
sp_adduser, 64
sp_addrolemember, 65
sp_help, 39-40
sp_helpdb, 41
sp_helprole, 65
sp_helptext, 52

sp_lock, 67
sp_who, 66
SQL
Books, 11
Server 2000 see Microsoft SQL Server 2000
SQL Commands
ALTER PROCEDURE command, 208
ALTER TABLE command, 54
CREATE PROCEDURE command, 50, 56, 105
CREATE TABALE command, 102
CREATE TRIGGER command, 53
CREATE VIEW command, 45, 105
DROP command, 67
DROP VIEW command, 46
GRANT command, 65
INSERT INTO command, 56
JOIN command, 188
KILL command, 67
REVOKE command, 65
SELECT command, 38
SET command, 55
UPDATE command, 53
USE command, 37, 41, 65
With SQL Query Analyzer, 42
SQL Data Definition Language see SQL DDL
SQL DDL, 30
SQL Query Analyzer Manager, 37-40
Applying Security, 64-67
Create Stored Procedure, 50-51
Create View, 46-47
SQL Commands, 42
Testing Security, 63-64
SQL Server 2000 see Microsoft SQL Server 2000
SQL Server Authentication, 58-59
SQL Server Enterprise Manager, 25-27
Create Stored Procedure, 48-49
Creating a Database, 31-36
Managing Roles, 61-63
Managing Security, 57-64
Managing Users, 57-64

Running Stored Procedure, 49-50
SQL Server Manager, 23-24
Creating a Database, 27-30
SQL Service, 23-24
SqlClient Data Provider, 84
Stored Procedure, 39-40
Create, 47-54
In SQL Query Analyzer, 50-51
In SQL Server Enterprise Manager, 48-49
ReturnValue, 100
Running, 47-54
In SQL Server Enterprise Manager, 49-50
sp_addlogin, 64
sp_adduser, 64
sp_addrolemember, 65
sp_help, 39-40
sp_helpdb, 41
sp_helprole, 65
sp_helptext, 52
sp_lock, 67
sp_who, 66
Using with Command Object, 95-100
View, 52
Strongly Typed DataSets, 250-255
System Base Classes, 83-84
System.Data.Odbc, 13, 85-86
System.Data.OleDb, 85-86
System.Data.SqlClient, 86

T

Table constraints (DataSet), 165-171
Table Mapping, 229-232
Table relations (DataSet), 158-165
Tabulator Data Stream see TDS
TDS, 84
Trigger
Create, 53-54
Types, 55
Try-Catch, 92, 104, 128, 156, 228

U

Update (DataAdapter), 190-193
UPDATE command, 53
USE command, 37, 41, 65

V

VB see Visual Basic.NET
VB.NET see Visual Basic.NET
View, 42
    Control Access to Data, 42
    Create (SQL Server Enterprise Manager), 43-45
    Create (SQL Query Analyzer Batch), 46-47
Visual Basic.NET, 11
Visual C Sharp see C#
Visual C# see C#
Visual C#.NET see C#
VS see Visual Studio.NET
VS.NET see Visual Studio.NET

W

Web Application, 12
Web Forms, 84
Web Services, 12, 84
Windows Application, 11
Windows Authentication, 59
Windows Forms, 11, 84

X

XML, 81

ת.ד. 6108 הרצליה 46160 **הזמנות באתר**
info@hod-ami.co.il www.**hod-ami**.co.il

קטלוג מרץ 2013 *הנחות ומבצעים באתר*

	כולל	ת. הוצאה	עמ'	מחיר*
אינטרנט - מפתחי אתרים/גרפיקה				
אמא, אבא - בניתי אתר באינטרנט		9/2003	256	69
הגדל את הכנסות העסק שלך באמצעות פרסום בגוגל Google AdWords		6/2010	300	249
מבוא לתכנות בסביבת אינטרנט – הכל כלול בספר אחד HTML + JS + ASP		3/2006	192	69
מבוא לתכנות בסביבת אינטרנט - מבוא ו- **HTML** - חלק 1 מתוך 3 - מהד' 3		6/2003	320	59
מבוא לתכנות בסביבת אינטרנט - **JavaScript** - חלק 2 מתוך 3		5/2002	240	49
מבוא לתכנות בסביבת אינטרנט - **ASP** - חלק 3 מתוך 3 - מהד' 3		3/2006	192	49
HTML5 המדריך לבניית אתרים, הדור הבא (לא בחנויות) חדש		10/2012	352	149
HTML & CSS למפתחי אתרים באינטרנט - מהד' 5		4/2005	592	179
The **Java** Tutorial סדנת לימוד	CD	7/2001	768	179
JavaScript סדנת לימוד		2001	586	159
ASP.NET 3.5 סדנת לימוד בשפות #C ו-VB		10/2009	824	229
תכנות				
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3 (לא בחנויות)		7/2011	350	169
SAS (**Statistical Analysis System**) – ספר לימוד (לא בחנויות)		11/2010	256	169
Visual C# 3.0 סדנת לימוד		9/2008	656	219
ללמוד **C** - מהד' 3		2/2001	480	139
מבוא למדעי המחשב בשפת **C** - מהד' 2 (לא בחנויות)		5/2006	458	119
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 3 (לא בחנויות)		2012	152	95
PC - חומרה, תוכנה ורשתות				
מדריך Hacking ואבטחת מידע, מהד' 2		9/2011	428	169
מדריך **חומרה ותוכנה** לטכנאי PC - מהד' 5 כולל עדכון 2011	CD	2/2011	752	189
Windows 7/8 נושאים מתקדמים (לא בחנויות. המחיר כולל משלוח)		חדש	140	49
מדריך **רשתות** לטכנאי PC ולמנהלי רשת - מהד' 4		4/2007	608	219
Windows				
Windows 7 צעד-אחר-צעד		1/2010	272	89
Windows 7/8 נושאים מתקדמים (לא בחנויות. המחיר כולל משלוח)		חדש	140	49
Windows XP והכרת המחשב האישי למתחילים		3/2003	208	59
גרפיקה				
Flash – ספר הדרכה ותרגילים (לא בחנויות) חדש		2012	122	64
Illusatrator – ספר הדרכה ותרגילים (לא בחנויות) חדש		2012	120	64
Photoshop צעד אחר צעד (צבע מלא, למתחילים), מהד' 3		3/2012	200	159
Photoshop CS2 עיבוד תמונה דיגיטלית (צבע מלא)		1/2006	480	199
מדריך לתוכנת העיצוב והאנימציה **3ds max** (2 כרכים) (לא בחנויות)	CD	מהד' 5	1400	419

* מחיר מומלץ לצרכן כולל מע"מ. **יש להיכנס לאתר כדי לבדוק מבצעים ומחירים מיוחדים**

היכנס לאתר להתעדכן בספרים החדשים ומבצעים www.**hod-ami**.co.il

	כולל	הוצאה	עמ'	מחיר*
OFFICE 2010				
טבלאות ציר – ניתוח נתונים חכם (לא בחנויות)		2012	116	87
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 3 (לא בחנויות)		2012	152	95
Word 2010 צעד אחר צעד		11/2010	160	69
Excel 2010 סדנת לימוד		11/2010	344	129
PowerPoint 2010 צעד אחר צעד		11/2010	202	69
Outlook 2010 צעד אחר צעד		11/2010	190	69
Access 2010 סדנת לימוד		11/2010	360	179
OFFICE 2007				
PowerPoint 2007 למתחילים		8/2007	240	99
Outlook 2007 למתחילים		1/2008	224	99
OFFICE 2003				
Word 2003 צעד אחר צעד		3/2004	208	69
Excel 2003 הסדרה הידידותית למתחילים		2/2004	160	49
PowerPoint 2003 הסדרה הידידותית למתחילים		2/2004	96	49
Outlook 2003 הסדרה הידידותית למתחילים		2/2004	96	49
Access 2003 הסדרה הידידותית למתחילים		2/2004	144	19
OFFICE XP				
Office XP צעד אחר צעד	CD	7/2001	576	139
Word XP הסדרה הידידותית למתחילים		7/2001	144	49
ניהול, כלכלה ושונות				
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3 (לא בחנויות)		7/2011	350	169
חקר ביצועים כשירות ללקוח		2/2010	236	92
ניהול ממוקד לעשות יותר עם מה שיש (כריכה קשה) - מהד' 4 חדש		9/2012	358	133
לי זה עולה יותר (תמחיר) (כריכה קשה) - מהד' 3		10/2006	368	129
מערכות מידע				
Oracle SQL יכולות מתקדמות (לא בחנויות)		7/2011	626	249
SAS (**Statistical Analysis System**) – ספר לימוד (לא בחנויות)		11/2010	256	169
בסיסי נתונים ושפת **SQL** – עקרונות ועיצוב (לא בחנויות)			648	149
ניתוח מערכות מידע כולל מתודולוגיית ה- **UML** (לא בחנויות)		11/2004	818	229
המדריך העברי השלם **UML** (לא בחנויות)		1/2009	346	329

* מחיר מומלץ לצרכן כולל מע"מ. קטלוג 3/2013.

<u>יש להיכנס לאתר כדי לבדוק מבצעים ומחירים מיוחדים</u>

היכנס לאתר להתעדכן בספרים החדשים ומבצעים

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il