Microsoft SQL Server 2005

שאילתות T-SQL

תרגום: **איציק בן-גן**

עריכה ועיצוב: **שרה עמיהוד, מירי אלעני**

עיצוב: **גלית גרבר-קטן**

עיצוב עטיפה: **שרון רז**

שמות מסחריים

שמות המוצרים והשירותים המוזכרים בספר הינם שמות מסחריים רשומים של החברות שלהם. הוצאת הוד-עמי ו- Microsoft Press עשו כמיטב יכולתם 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 ו-Microsoft Press 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, 1-700-7000-44**
- **פקס: 09-9571582**
- **דואר אלקטרוני: info@hod-ami.co.il**
- **אתר באינטרנט: www.hod-ami.co.il**

Microsoft SQL Server 2005

שאילתות T-SQL

Itzik Ben-Gan

Lubor Kollar, Dejan Sarka

Microsoft

הוצאת הוד-עמי

Inside Microsoft SQL Server 2005:

T-SQL Querying (Solid Quality Learning)

By Itzik Ben-Gan, Lubor Kollar, Dejan Sarka

ISBN 978-0-7356-2313-2

© 2006 by Microsoft Corporation. All rights reserved. Original English language edition © 2006 by Itzik Ben-Gan and Lubor Kollar. All rights reserved. Published by arrangement with the original publisher, Microsoft Corporation, Redmond, Washington, U.S.A.

Hebrew language edition published by Hod-Ami Ltd. Copyright © 2006.

© כל הזכויות שמורות

הוצאת הוד-עמי בע"מ

ת.ד. 6108 הרצליה 46160

טלפון: 09-9564716   פקס: 09-9571582

www.hod-ami.co.il

info@hod-ami.co.il

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.

מהדורה ראשונה 2006

All Rights Reserved

HOD-AMI Ltd.

P.O.B. 6108, Herzliya

ISRAEL, 2006

מסת"ב ISBN 965-361-381-2

BP -ל

$e\Pi i + 1 = 0$

תוכן עניינים מקוצר

1. עיבוד לוגי של שאילתות..27

2. עיבוד פיסי של שאילתות..63

3. כוונון שאילתות..103

4. תת–שאילתות, ביטויי טבלה ופונקציות דירוג..243

5. Joins ופעולות סטים..331

6. פונקציות צבירה של נתונים וסיבוב על ציר..391

7. TOP ו–APPLY..471

8. שינוי נתונים..511

9. גרפים, עצים, היררכיות ושאילתות רקורסיביות..559

נספח א: חידות היגיון..671

אינדקס..697

תוכן העניינים

הקדמה 17

1. עיבוד לוגי של שאילתות 27

המקור להגייה של SQL 27
שלבי עיבוד לוגי של שאילתה 29
תיאור קצר של שלבי העיבוד הלוגי של שאילתה 30
שאילתה לדוגמה מבוססת על לקוחות/הזמנות 31
פירוט שלבי העיבוד הלוגי של שאילתה 33
שלב 1: ביצוע מכפלה קרטזית (Cross Join) 33
שלב 2: יישום המסנן ON (Join Condition) 35
שלב 3: הוספת שורות חיצוניות 38
שלב 4: יישום המסנן WHERE 39
שלב 5: קיבוץ 40
שלב 6: יישום האפשרות CUBE או ROLLUP 42
שלב 7: יישום המסנן HAVING 42
שלב 8: עיבוד רשימת ה–SELECT 43
שלב 9: יישום הפסוקית DISTINCT 44
שלב 10: יישום הפסוקית ORDER BY 44
שלב 11: יישום האפשרות TOP 47
שלבים חדשים בעיבוד לוגי של שאילתה ב– SQL Server 2005 48
אופרטורים טבלאיים 49
APPLY 50
PIVOT 52
UNPIVOT 55
הפסוקית OVER 58
פעולות על קבוצות (Set Operations) 60
סיכום 62

2. עיבוד פיסי של שאילתות 63

זרימת נתונים במהלך עיבוד שאילתה 64
קומפילציה 68

Algebrizer 71
שיטוח אופרטורים 71
פענוח שמות 72
גזירת טיפוס נתונים (Type Derivation) 72
כריכת צבירות (aggregate binding) 73
כריכת קיבוץ (Grouping Binding) 73
אופטימיזציה 74
עבודה עם תוכנית שאילתה 82
SET SHOWPLAN_TEXT ו–SHOWPLAN_ALL 83
צורת XML של ה–Showplan 85
Showplan גרפי 87
מידע זמן ריצה ב–Showplan 89
SET STATISTICS XML ON|OFF 89
SET STATISTICS PROFILE 90
לכידת Showplan עם SQL Trace 91
חילוץ ה–Showplan מ–Cache הפרוצדורות 94
תוכניות עדכון 96
סיכום 101
תודות 101

3. כוונון שאילתות 103
נתוני דוגמה עבור פרק זה 103
מתודולוגיית כוונון 108
ניתוח המתנות ברמת המופע (instance) 111
קישור בין המתנות לתורים 120
קביעת דרך פעולה 122
ממשיכים מטה לרמת מסד הנתונים/הקובץ 122
ממשיכים מטה לרמת התהליך 125
Trace ביצועי פעילות 127
ניתוח נתוני Trace 132
כוונון אינדקסים/שאילתות 148
כלים לכוונון שאילתות 149
Syscacheobjects 149
ניקוי ה–Cache 150

אובייקטי ניהול דינמיים (Dynamic Management Objects) 150
STATISTICS IO 150
מדידת זמן הריצה של שאילתות 151
ניתוח תוכניות עבודה (Execution Plans) 152
תוכניות עבודה גרפיות 153
Showplan טקסטואלי 161
XML Showplans 163
Hints 165
Traces/Profiler 167
Database Engine Tuning Advisor 167
כוונון אינדקסים 168
מבני טבלאות ואינדקסים 168
דפים ו–Extents 168
Heap 169
אינדקס–Clustered 171
אינדקס–Nonclustered על Heap 175
אינדקס–Nonclustered על טבלה–Clustered 177
שיטות גישה לאינדקסים 178
Table Scan/Unordered Clustered Index Scan 179
Unordered Covering Nonclustered Index Scan 181
Ordered Clustered Index Scan 183
Ordered Covering Nonclustered Index Scan 185
Nonclustered Index Seek
+ Ordered Partial Scan + Lookups 188
Unordered Nonclustered Index Scan + Lookups 192
Clustered Index Seek + Ordered Partial Scan 196
Covering Nonclustered Index Seek
+ Ordered Partial Scan 198
הצלבת אינדקסים (Index Intersection) 201
Indexed Views 202
סרגל לאופטימיזציה של אינדקסים 204
Table Scan (Unordered Clustered Index Scan) 205
Unordered Covering Nonclustered Index Scan 206
Unordered Nonclustered Index Scan + lookups 207

Nonclustered Index Seek
\+ Ordered Partial Scan + Lookups 208
קביעת נקודת הסלקטיביות 208
Clustered Index Seek + Ordered Partial Scan 211
Covering Nonclustered Index Seek
\+ Ordered Partial Scan 212
סיכום וניתוח סרגל אופטימיזציה של אינדקסים 213
פרגמנטציה 218
חציצה (Partitioning) 221
הכנת נתונים לדוגמה 221
הכנת נתונים 221
TABLESAMPLE 230
בחינה של גישות מבוססות–סטים לעומת
גישות איטרטיביות/פרוצדורליות, ותרגיל כוונון 233
מקורות נוספים 241
סיכום 242

4. תת–שאילתות, ביטויי טבלה ופונקציות דירוג 243
תת–שאילתות 243
תת–שאילתות עצמאיות 244
תת–שאילתות תלויות 248
שובר–שוויון 249
EXISTS 253
EXISTS לעומת IN 254
NOT EXISTS לעומת NOT IN 255
ערך חסר מינימלי 258
הפעלת לוגיקה הפוכה על בעיות של חלוקה רלציונית 261
שאילתות המתנהגות בצורה לא צפויה 263
אופרטורים לא שגרתיים 265
ביטויי טבלה 267
טבלאות נגזרות 267
כינויים לטורי תוצאה 268
שימוש בארגומנטים 269
קינון 269
התייחסויות מרובות 270

ביטויי טבלה שגורים (CTE) 271
כינויים של טורי תוצאה 271
שימוש בארגומנטים 272
CTEs מרובים 273
התייחסויות מרובות 273
שינוי נתונים 274
אובייקטים מכילים 275
CTEs רקורסיביים 277
פונקציות דירוג אנליטיות 280
מספר שורה 282
הפונקציה ROW_NUMBER ב– SQL Server 2005 283
דטרמיניזם 284
חציצה (partitioning) 286
שיטה מבוססת–סטים לפני SQL Server 2005 286
טור מיון ייחודי 287
טור מיון לא ייחודי ושובר–שוויון 289
טור מיון לא ייחודי ללא שובר–שוויון 290
חציצה 293
פתרון מבוסס–סמן 294
פתרון מבוסס–IDENTITY 296
ללא–מחיצות 296
עם–מחיצות 297
בוחן ביצועים 298
דפדוף (Paging) 305
דפדוף אד–הוק 305
גישה לדפים מרובים 307
RANK ו–DENSE RANK 308
פונקציות RANK ו-DENSE_RANK ב– SQL Server 2005 308
פתרונות מבוססי–סטים טרום SQL Server 2005 310
NTILE 310
הפונקציה NTILE ב– SQL Server 2005 310
פתרונות מבוססי–סטים אחרים ל–NTILE 313
טבלת עזר של מספרים 317
טווחים קיימים וחסרים (Islands ו–Gaps) 321

טווחים חסרים (ידועים גם כפערים – Gaps) 323
טווחים קיימים (ידועים גם כאיים) 326
סיכום 329

5. Joins ופעולות סטים 331

Joins 331
סגנון ישן לעומת סגנון חדש 331
סוגי Join בסיסיים 332
Cross Join 333
Inner Join 339
Outer Join 341
סוגי Joins שאינם נתמכים 346
דוגמאות נוספות של Joins 346
Self Join 346
Nonequijoin 347
Joins מרובים 349
שליטה בסדר הערכה (evaluation order) פיסי של Joins 350
Hints 352
שליטה בסדר הערכה (evaluation order) לוגי של Joins 353
Semi Join 357
כמות כוללת נעה של שנה קודמת 359
אלגוריתמים של Join 364
Loop Join 364
Merge Join 365
Hash Join 367
אכיפת אסטרטגיית Join 368
הפרדת אלמנטים (פירוק מערכים) 369
פעולות סטים 378
UNION 379
UNION DISTINCT 379
UNION ALL 379
EXCEPT 380
EXCEPT DISTINCT 380

EXCEPT ALL 381
INTERSECT 383
INTERSECT DISTICNT 383
INTERSECT ALL 384
קדימות של פעולות סטים 385
שימוש ב–INTO עם פעולות סטים 386
עקיפת שלבים לוגיים לא נתמכים 386
סיכום 390

6. פונקציות צבירה של נתונים וסיבוב על ציר 391

הפסוקית OVER 391
שוברי–שוויון 395
פונקציות צבירה רצות 398
פונקציות צבירה מצטברות 401
פונקציות צבירה נעות 406
מתחילת–תקופה (Year-To-Date – YTD) 409
סיבוב על ציר – משורות לטורים (Pivoting) 410
סיבוב מאפיינים 410
חלוקה רלציונית 415
פונקציות צבירה של נתונים 418
סיבוב על ציר – מטורים לשורות (Unpivoting) 423
פונקציות צבירה מוגדרות–משתמש 426
פונקציות צבירה מוגדרות–משתמש המשתמשות ב–Pivoting 427
שרשור מחרוזות על ידי שימוש בסיבוב על ציר 428
פונקציית הצבירה הכפלה על ידי שימוש בסיבוב על ציר 429
פונקציות צבירה מוגדרות משתמש (UDA) 429
קוד CLR בבסיס נתונים 430
יצירת assembly ב– Visual Studio 2005 437
בדיקת פונקציות צבירה מוגדרות משתמש (UDA) 442
פתרונות ייחודיים 442
פתרון ייחודי לפונקציית הצבירה שרשור מחרוזות 443
פתרון ייחודי לפונקציית הצבירה הכפלה 443
פתרון ייחודי לפונקציית הצבירה פעולות מבוססות–סיבית 445
פונקציית הצבירה של פעולת סיבית OR 447

פונקציית הצבירה של פעולת סיבית AND 449
פונקציית הצבירה של פעולת הסיבית XOR 450
חציון 451
היסטוגרמות 454
גורם הקבצה 459
CUBE ו–ROLLUP 463
CUBE 463
ROLLUP 468
סיכום 470

7. TOP ו–APPLY 471

SELECT TOP 471
TOP ודטרמיניזם 473
TOP וביטויי קלט 475
TOP ושינויי נתונים 476
APPLY 480
פתרונות לבעיות נפוצות המשתמשים ב–TOP ו–APPLY 483
TOP n לכל קבוצה 483
התאמת מופעים נוכחיים וקודמים 490
דפדוף (Paging) 495
דף ראשון 496
הדף הבא 497
דף קודם 504
שורות רנדומאליות 505
חציון 507
סיכום 510

8. שינוי נתונים 511

הוספת נתונים 511
SELECT INTO 511
INSERT EXEC 513
הוספת שורות חדשות 518
INSERT עם OUTPUT 522
מנגנוני רצף 524

טורי Identity 524
רצפים מוגדרי–משתמש 525
יצירת רצף סינכרוני 525
יצירת רצף בלתי–סינכרוני 529
Globally Unique Identifiers 531
מחיקת נתונים 531
TRUNCATE לעומת DELETE 531
הסרת שורות עם נתונים כפולים 532
DELETE על ידי שימוש ב–Joins 535
DELETE עם OUTPUT 539
עדכון נתונים 541
UPDATE על ידי שימוש ב–joins 541
UPDATE עם OUTPUT 546
משפטי SELECT ו–UPDATE של הצבה 549
SELECT של הצבה 549
UPDATE של הצבה 551
שיקולי ביצועים אחרים 554
סיכום 557

9. גרפים, עצים, הירכיות ושאילתות רקורסיביות 559

טרמינולוגיה 560
גרפים 560
עצים 561
הירכיות 561
תרחישים 562
מבנה ארגוני 562
עץ מוצר (BOM) 564
מערכת כבישים 569
איטרציה/רקורסיה 574
כפופים 575
אבות (Ancestors) 589
פתרונות תת–גרף/תת–עץ עם מסלול אבות (Path Enumeration).... 594
מיון 598
מחזוריות 611

מסלול אבות ממומש.. 615
תחזוקת נתונים.. 616
הוספת עובדים שאינם מנהלים (עלים)........................ 617
הזזת תת-עץ.. 619
הסרת תת-עץ.. 623
ביצוע שאילתות.. 624
סטים מקוננים (Nested Sets).. 630
הקצאת ערכי שמאל וימין.. 631
ביצוע שאילתות.. 643
Transitive Closure.. 646
גרף מכוון לא-מעגלי (Directed Acyclic Graph)........................ 647
גרף בלתי-מכוון מעגלי (Undirected Cyclic Graph)............ 655
סיכום.. 668

נספח א: חידות היגיון.. 671
חידות.. 671
פתרונות לחידות.. 679

אינדקס.. 697

הקדמה...

כאשר חשבתי על כתיבת ההקדמה לכרך זה, המילה "מומחה" ("master") עלתה במוחי עוד טרם פתחתי את הטיוטה. כפועל – "להתמחות" משמעותו לרכוש ידע מעמיק במשהו או להפוך להיות מיומן בשימושיו. כשם–עצם – "מומחה" היא מילה עשירה עם פרשנויות עדינות רבות. בהקשר של ספר זה עולות שתיים. מומחה כמישהו בעל כישורים מושלמים במיזם כלשהו ומומחה כמישהו מוסמך ללמד חניכים. כאשר פתחתי טיוטה מוקדמת הופתעתי לגלות, במעין סינכרוניות, את הפרק הראשון שפותח בדיון על "התמחות" בעקרונות היסוד של SQL.

המאפיין המדהים של שפת ה–SQL הוא הנגישות. תוכל ללמד אדם את הרעיונות הבסיסיים בתוך דקות ולהביא אותו מהר מאוד למצב בו הוא יכול לחקור את מסד הנתונים לבדו. SQL ניתן ללמוד בקלות אך לא בקלות מתמחים בה. התמחות בעושר של SQL דורשת זמן, אך הגמול רב. קיימות דרכים רבות לבצע דבר כלשהו בשפת SQL. במקרים רבים, כאשר אנשים מתחילים ללמוד SQL, הם מעצבים פתרונות לשאילתות שלהם בדרכים כואבות, עקומות ובלתי יעילות. מומחי SQL, מצד שני, לעיתים קרובות יוצרים פתרונות פשוטים ואלגנטיים שמהנה להסתכל עליהם, להעריך אותם וללמוד מהם. ספר זה מלא בדוגמאות לפתרונות פשוטים ואלגנטיים אלו, שיסייעו לך לשלוט ב–SQL.

בחיים אנו פוגשים בשני סוגים של מומחים; אלו החולקים את מומחיותם אך ורק דרך העבודה שלהם; ואלו הלוקחים חניכים לקדם את האמנות שלהם – להעביר אותה הלאה ולחלוק אותה עם אחרים. לקוראים של ספר זה יש את ההזדמנות להפוך לחניכים של SQL מפי מומחים של השפה ושל היישום הייחודי שלה ב– Microsoft SQL Server. איציק בן–גן, הסופר הראשי של ספר זה, הוא מומחה מוכר בתחום ה–SQL. איציק הוא אחד מהאנשים המגדירים מחדש את המדיום. לעיתים קרובות, כאשר אני מתבונן בפתרון שאיציק עיצב, אני נתקף בתחושה הזו של "לא ידעתי שאפשר לעשות את זה!". לובור קולר (Lubor Kollar), מנהל צוות בצוות פיתוח המוצר של Microsoft SQL Server, הוא מומחה בפתרון בעיות של לקוחות. לעיתים אני הלקוח. פעמים רבות נכנסתי למשרד של לובור בשאלה על דרך מסוימת לפתרון בעיה כלשהי ב–SQL, ובתוך דקות, לובור שלח לי פתרון שבסופו של דבר מצא את דרכו למחשב הנייד שלי לשימוש עתידי. דיין סרקה (Dejan Sarka), SQL Server MVP ידוע, הוא בעל ידע נרחב ביותר בתחום הרלציוני הטהור, אך הוא גם מומחה BI ובשילוב XML ו–CLR ב– SQL Server 2005. תרומתו העיקרית של דיין לספר זה היא בהדרכת הקוראים מתי רצוי לשלב XML או CLR בפתרונות שלהם.

הספר **שאילתות T-SQL** מתחיל היכן שצריך, בהפרדת המבנים הלוגיים של שפת SQL מהיישומים הפיסיים שלהם ב– Microsoft SQL Server. כמו במתמטיקה, נדרשת הבנה מעמיקה של המבנים הלוגיים עליהם SQL נוצרה, כדי לבנות בסיס יציב להמשך חקירה. פרקים 2 ו–3 נכנסים להיבטים של העיבוד הפיסי של שאילתות ומראים כיצד המוצר, Microsoft SQL Server, הופך שאילתה לוגית לתוכנית עבודה ניתנת לביצוע,

המחזירה את תוצאות השאילתה בצורה יעילה. רצוי להחזיק במושגים הללו של לוגי ופיסי בנפרד לחלוטין. עם זאת, ישנן מספר דרכים לתאר שאילתה בצורה לוגית, וחלק מהן מתורגמות ביתר קלות מאשר אחרות על ידי ה– SQL Server query optimizer לתוכניות יעילות. פרקים 4 עד 8 יוצרים את הבסיס לשליטה בשאילתות SQL. בפרקים אלה תלמד לבצע במספר שורות מלל של שאילתת SQL דברים שאנשים רבים (המשתמשים בשפת SQL אך עדיין לא שולטים בה) טורחים עליהם עם סמנים, טבלאות זמניות, או גרוע מכך, עם היגיון תכנות קשיח. כאן תלמד לשאול את השאלה כשאילתת SQL ותקבל את התשובה מ– SQL Server במקום לתכנת את התשובה בעצמך. תמצא שהפרקים האלה עשירים בדוגמאות רבות מהעולם האמיתי עם מספר יישומים חלופיים והסברים מתי להעדיף אחד על פני השני. לבסוף, פרק 9: "גרפים, עצים, היררכיות ושאילתות רקורסיביות" הוא ממלכתם של המומחים. אנשים רבים, שאינם מנוסים מספיק ב–SQL או במודל הרלציוני, מאמינים שמבני גרף אינם יכולים להיות מיוצגים בעולם הרלציוני. בעוד שברור כי דבר זה אינו נכון, ישנם כמה טיפולים בנושא בצורה מקיפה בטקסטים קיימים של SQL. פרק סיום זה מתאר כיצד לעצב גרפים ועצים דרך המהות שלהם – יחסים. הוא מציג מודל נתונים נפוץ לייצוג מבנים כגרפיים ואז מתאר מספר שיטות, כולל שימוש במאפיינים חדשים של SQL Server 2005, לנווט בין המבנים הללו.

הספר **שאילתות T-SQL** אינו רק ספר מתכונים. זה אינו המקום לחפש לגזור ולהדביק 20 שורות של קוד T-SQL לתוך התוכנית שלך. בעוד שהוא מכיל מתכונים ודוגמאות מפורשים, הספר **שאילתות T-SQL** הוא ספר על לימוד האמנות של SQL. על ידי קריאה בו תלמד לעשות דברים אותם חשבת לבלתי אפשריים לביצוע עם שפת SQL. כאשר תפגוש בעתיד אתגרים של SQL הדורשים שיטות שהוצגו בספר זה, תמצא את עצמך מושיט יד לספר זה שעל המדף שלך. בין אם אתה חדש בשפת SQL ו– Microsoft SQL Server, או שהינך עוסק בה, או שאתה כבר בדרכך להיות מומחה SQL, אני משוכנע שהספר הזה יסייע לך לקדם את האמנות שלך.

Microsoft SQL Server הוא כלי או פלטפורמה. כל אחד בצוות הפיתוח של המוצר מרגיש גאווה גדולה במסירת כלי נהדר. אך לבסוף – הערך האמיתי של SQL Server מובן רק כאשר הוא נמסר לידיו של מישהו שיוצר איתו משהו; בדיוק כפי שנגר משתמש בכלי ליצירת רהיט מובחר. כדי להיות נגר מומחה נדרש ידע של עקרונות היסוד של המדיום ושיטות לעבודה בו; חוש עיצוב מצוין; וידע מעמיק של כלי העבודה. אמנים מכירים את הכלים שלהם – כיצד ומתי להשתמש בהם; כמו גם כיצד לתחזק אותם. הספר **שאילתות T-SQL** יציג לך את עקרונות היסוד של שפת SQL; מקריאה בו, תבין בצורה ברורה כיצד לעצב פתרונות SQL אלגנטיים ממומחים של האומנות; לבסוף, תקבל הערכה להיבטים הייחודיים של Microsoft SQL Server ככלי לשימושך, כאשר אתה מתמחה בשפת SQL ליצירת פתרונות באמצעים פשוטים ואלגנטיים.

דיויד קמפבל
מנהל כללי, Microsoft SQL Server: אסטרטגיה, תשתית וארכיטקטורה.

מבוא

ספר זה וספר ההמשך – **Inside Microsoft SQL Server 2005: T-SQL Programming** (**MS SQL Server 2005: תכנות T-SQL**) – עוסקים בנושאים מתקדמים של ביצוע שאילתות, כוונון שאילתות ותכנות ב- SQL Server 2005. הם נכתבו עבור מפתחים ומנהלי מסדי נתונים הנדרשים לכתוב ולשפר קוד ב- SQL Server, הן בגרסת 2000 והן בגרסת 2005. בספר זה אתייחס לשני הספרים, ולשם הקיצור אתייחס לספרים בשמות **שאילתות T-SQL** ו**תכנות T-SQL**.

הספרים מתמקדים בבעיות מעשיות נפוצות, ודנים במספר גישות לפתרון כל אחת מהן. במהלך הספר תפגוש שיטות מלוטשות רבות שיעשירו את הכלים שברשותך ואת אוצר המילים התכנותי שלך, ויאפשרו לך לספק פתרונות יעילים בצורה טבעית.

הספרים חושפים את הכוח של שאילתות מבוססות-סטים, ומסבירים מדוע הן לרוב עדיפות על פני תכנות פרוצדורלי עם סמנים וכדומה. בו בזמן, הם ילמדו אותך כיצד לזהות את המקרים הבודדים בהם פתרונות מבוססי-סמן טובים יותר מאלו מבוססי-הסטים.

הספרים גם מציגים מבנים אחרים השנויים במחלוקת - כגון טבלאות זמניות, הפעלה דינמית, שילוב XML ו-NET. בכולם טמונים יכולות חזקות, אך בו בזמן גם סיכון רב. מבנים אלו דורשים בגרות תכנותית. ספרים אלו ילמדו אותך מתי יש להשתמש במבנים אלה, וכיצד לעשות זאת בצורה חכמה, בטוחה ויעילה.

ספר זה - **שאילתות T-SQL** - מתמקד בשאילתות מבוססות-סטים, ואני ממליץ לקרוא אותו ראשון. הספר השני - **תכנות T-SQL** (שבשלב זה לא ראה אור במהדורה עברית) - מתמקד בתכנות פרוצדורלי ומניח שקראת את הספר הראשון או שיש לך רקע מספק בתחום השאילתות.

הספר מתחיל בשלושה פרקים המניחים את הבסיס לעיבוד לוגי ופיסי של שאילתות, הנדרש לצורך הפקת המרב מיתר הפרקים.

הפרק הראשון עוסק בעיבוד לוגי של שאילתות. הוא מתאר בפירוט את השלבים הלוגיים המעורבים בעיבוד שאילתות, את ההיבטים הייחודיים של שאילתות SQL ואת הלך המחשבה המיוחד הדרוש כדי לתכנת בסביבה רלציונית (יחסית, Relational), מוכוונת סטים.

הפרק השני עוסק בעיבוד פיסי של שאילתות. הוא מתאר בפירוט את הדרך בה המנוע של SQL Server מעבד שאילתות, ומשווה מנגד עיבוד פיסי של שאילתות עם עיבוד לוגי שלהן. פרק זה נכתב על ידי לובור קולר (Lubor Kollar). לובור היה ראש צוות פיתוח בתקופת הפיתוח של SQL Server 2005 והצוות שלו היה אחראי לחלק "התחתון" של המנוע הרלציוני - מקומפילציה ואופטימיזציה של שאילתות לביצוע שאילתות, אחידות טרנזקציות, גיבוי/שחזור וזמינות גבוהה. המאפיינים הבולטים ב- SQL Server 2005 עליהם עבד הצוות שלו היו מחיצות של טבלאות ואינדקסים, שיקוף (Mirroring) של מסד נתונים, Snapshot של מסד נתונים, Snapshot Isolation, שאילתות רקורסיביות ושיפורים

נוספים ב- T-SQL, Database Tuning Advisor ויצירה ותחזוקה מקוונות של אינדקסים. בכל העולם סביר להניח שרק מספר קטן של אנשים מכיר את סוגיית האופטימיזציה של שאילתות טוב כמו לובור. אני רואה בכך פריבילגיה שאחד מהמעצבים של ה-optimizer מסביר אותו במילותיו הוא.

הפרק השלישי עוסק במתודולוגיה של כוונון שאילתות שפיתחנו בחברתנו (Solid Quality Learning) ויישמנו במערכות תפעוליות. הפרק עוסק גם בעבודה עם אינדקסים ובניתוח תוכניות עבודה. פרק זה מספק רקע חשוב הנדרש לפרקים שאחריו, העוסקים בעבודה עם אינדקסים וניתוח תוכניות עבודה. אלו היבטים חשובים של ביצוע וכוונון שאילתות.

הפרקים הבאים מתעמקים בהיבטים מתקדמים של ביצוע וכוונון שאילתות, בהם משתלבים הן ההיבטים הלוגיים והן ההיבטים הפיסיים של הקוד שלך. פרקים אלו כוללים: תת-שאילתות, ביטויי טבלה ופונקציות דירוג; Joins ופעולות סטים; סיכומי נתונים וסיבובם על ציר (כולל סעיף על סיכומים מוגדרי-משתמש ב-CLR, שנכתב על ידי דיין סרקה (Dejan Sarka)); TOP ו-APPLY; שינויי נתונים; והיררכיות ושאילתות רקורסיביות.

נספח א' עוסק בחידות היגיון. כאן יש לך הזדמנות לתרגל חידות היגיון כדי לשפר את הכישורים הלוגיים שלך. שאילתות SQL ביסודן עוסקות בלוגיקה. אני מוצא שחשוב לתרגל היגיון טהור כדי לשפר את יכולותיך לפתרון בעיות של שאילתות. אני גם מוצא בחידות אלו הנאה ואתגר, וניתן לתרגל אותן עם כל המשפחה. חידות אלו הן אוסף של חידות ההיגיון שפרסמתי בטור T-SQL שלי בירחון SQL Server Magazine. אני רוצה להודות ל- SQL Server Magazine שהרשו לי לחלוק את החידות הללו עם קוראי הספר.

הספר השני - **תכנות T-SQL** - מתמקד במבנים תכנותיים של T-SQL ומרחיב על שילוב של XML ו-NET. הנושאים אותם הוא כולל: בעיות הקשורות לטיפוסי נתונים, כולל טיפוסים מוגדרי-משתמש (UDT) ב-XML ו-CLR; טבלאות זמניות; סמנים; הפעלה דינמית; views; פונקציות מוגדרות-משתמש (UDF), כולל CLR UDFs; פרוצדורות מאוחסנות, כולל פרוצדורות CLR; triggers, כולל DDL ו- CLR triggers; טרנזקציות, כולל כיסוי של רמות הבידוד החדשות מבוססות-snapshot; טיפול בשגיאות ו- service broker.

הסעיפים בספר שעוסקים בשילוב של XML ו-NET. ("טיפוסים מוגדרי-לקוח", "פונקציות מוגדרות-לקוח", "פרוצדורות מאוחסנות" ו"Triggers") נכתבו על ידי דיין סרקה. דיין הוא מומחה SQL Server בעל ידע רב במודל הרלציוני (יחסי). הוא בעל השקפות מרתקות על הדרך בה מבנים חדשים אלו יכולים להתאים למודל הרלציוני כאשר משתמשים בהם בחוכמה. לדעתי חשוב שתחומים אלו של המוצר, השנויים במחלוקת, יוסברו על ידי מישהו עם תפיסה רחבה של המודל הרלציוני. כל דוגמאות הקוד של CLR מובאות הן ב-C# והן ב- Visual Basic .NET.

הפרק האחרון הכולל את Service Broker נכתב על ידי רוג'ר וולטר (Roger Wolter). רוג'ר הוא ראש צוות המפתחים של SQL Server שאחראי על ה- Service Broker. שוב, אין כמו מעצב הרכיב עצמו שיסביר עליו במילותיו הוא.

אחרון אך לא פחות חשוב, סטיב קאס (Steve Kass) הוא העורך הטכני של הספרים. סטיב הוא בחור חריף ביותר. הוא SQL Server MVP, והוא מלמד מתמטיקה באוניברסיטת דרו (Drew). יש לו ידע נרחב ב– SQL Server ולוגיקה, ותרומתו לספרים יקרת ערך.

ולך, הקורא של ספרים אלו, הייתי רוצה לומר שעבורי SQL היא מדע, לוגיקה ואומנות. בישלתי ספרים אלו זמן רב, ושפכתי לתוכם את כל הלהט שלי ושנים רבות של ניסיון. אני מקווה שתמצא את הספרים שימושיים ומעניינים, ושתמצא ב–SQL מקור השראה כפי שהיא לי. אם יש לך הערות או תיקונים שתרצה לחלוק, אשמח לשמוע. ניתן ליצור איתי קשר דרך **http://www.insidetsql.com**.

בהוקרה,

איציק בן–גן

תודות

מרבית הקוראים בדרך כלל מדלגים על פרק התודות, וסופרים רבים בדרך כלל כותבים אותו קצר מאוד. באמת שאיני רוצה לשפוט אף אחד; אני יכול רק לנחש שסופרים חושבים שחשיפת רגשותיהם עלולה להיראות קיטשית, או שהם נבוכים לעשות כך. ובכן, אנשים רבים תרמו את לבם ונשמתם לספרים אלו, ולא אכפת לי איך פרק זה עלול להיראות. אני רוצה שהם יוכרו בתרומתם!

הכרת תודה עמוקה ביותר שלי לכל אלו שלקחו חלק או תרמו בכל דרך לספרים. חלקם השקיעו שעות רבות במעורבות ישירה בפרויקט, ולחלקם הייתה השפעה עליי ועל עבודתי, שהשפיעה על הספרים באופן עקיף.

לסופרים האורחים לובור קולר, דיין סרקה ורוג'ר וולטר: תודה שלקחתם חלק בפרויקט זה והוספתם את התובנות יקרות הערך שלכם. זה היה כבוד ועונג לעבוד איתכם. לובור, עומק הידע והלהט שלך הם מקור השראה. דייקו, חברי הטוב, אני תמיד לומד ממך דברים חדשים. אני מוצא שההשקפות שלך על עולם מסדי הנתונים פשוט מרתקות. רוג'ר, אני מעריך מאוד את העובדה שהסכמת לתרום לספרים. Service Broker – התינוק שלך – מביא ממד חשוב ל– SQL Server שלא היה שם קודם לכן. אני בטוח שחברות ימצאו בו ערך רב, ואני להוט לראות כיצד טכנולוגיית התורים תיושם - יש לה פוטנציאל כה רב.

לסטיב קאס, העורך הטכני של הספרים: סטיב, אין לי מילים לתאר עד כמה אני מעריך את התרומה שלך. אתה פשוט גאוני ומדהים, ואדם יכול רק לקוות שיהיה לו מחצית מהתבונה וההיגיון בהם בורכת. תזכיר לי לעולם לא להגיע לקרב מוחות איתך. השקעת כל–כך הרבה זמן בפרויקט וסיפקת כל–כך הרבה הצעות בעלות תובנה עד כי אני מרגיש שלמעשה סייעת לכתוב את הספרים. אני מקווה שבמהדורות עתידיות תמלא תפקיד כתיבה רשמי.

לדיויד קמפבל (David Campbell) ולובור קולר שכתבו את ההקדמות: העבודה וההישגים שלכם הם אור מנחה לרבים מאיתנו. SQL Server גדל להיות מוצר בוגר ומרתק - כזה ששווה בהחלט להקדיש לו את הקריירות המקצועיות שלנו ולמקד את הלהט שלנו בו. תודה לשניכם על שהסכמתם לכתוב את ההקדמות. זהו כבוד אמיתי! לכל התורמים, אני מצפה לעבודה משותפת על פרויקטים בעתיד. אני בכל אופן, כבר התחלתי לבשל רעיונות למהדורות עתידיות של הספרים.

הרבה תודות לצוות בהוצאת הספרים של מיקרוסופט (Microsoft Press): בן ריאן (Ben Ryan), קריסטין האגסט (Kristine Haugset), רוג'ר לבלאנק (Roger LeBlanc), וכפי הנראה רבים אחרים שלקחו חלק בהכנת הספרים. בן, אני מצטער שהייתי כל–כך מעיק, ושרציתי להיות מעורב בכל פרט קטן שיכולתי. מושלמות וכנות הם שני הקווים שמנחים אותי, על אף שאפשר רק לשאוף לראשון. אני מאמין שעל ידי הליכה בדרך זו, התוצאה הסופית יכולה רק להשתפר, ובלי קשר, אני מאמין שאין דרך אחרת. תודה שהיית כה קשוב. קריסטין, את פשוט נהדרת! מסורה, מקצועית, אכפתית ומנווטת את הפרויקט בצורה כה אלגנטית. ברמה אישית יותר, אני מרגיש שהרווחתי חברה חדשה. רוג'ר, אני

לא מקנא בך על השעות הרבות מספור שהשקעת בעריכת הספרים. תודה שעזרת לשפר את איכותם. ואני בטוח שהיו רבים נוספים בהוצאה שעבדו שעות ארוכות מאחורי הקלעים כדי לאפשר לספרים לראות אור יום.

אני רוצה להודות לקיילן דלייני (Kalen Delaney). ספרי **Inside SQL Server** הקודמים של קיילן היו תנ"ך עבורי בכל הקשור ל- internals של SQL Server, ואני להוט לקרוא את הסקירה שלה על internals ב- SQL Server 2005 בכרכים החדשים. קיילן הייתה גם זו שביקשה ממני לראשונה לכתוב את הכרכים של T-SQL כעת כשהמוצר גדל כל-כך.

אנשים רבים סיפקו משוב יקר ערך ביותר על ידי ביקורת לא רשמית של הספרים. ביניהם חברים בצוות הפיתוח של SQL Server, מנטורים מ- Solid Quality Learning ו-MVPs. לא הייתם חייבים לעשות זאת, ועשיתם זאת בהתנדבות - נתונה לכם הכרת תודתי הכנה.

לצוות ב- SQL Server Magazine: עבורי אתם משפחה. אנחנו עובדים ביחד שנים וגדלנו והתפתחנו ביחד. אני בטוח שאיני היחיד שמאמין ש- SQL Server Magazine הוא המגזין הטוב בעולם למקצועני SQL Server. ספרים אלו בחלקם הגדול הם בזכות מה שספגתי ולמדתי בעבודה אתכם.

לחברי, שותפי ועמיתי ב- Solid Quality Learning: החברה הזו היא ללא ספק הדבר הטוב ביותר שקרה לי בקריירה המקצועית שלי, ומבחינות רבות בחיי האישיים. זה פשוט חלום שהתגשם. בהקשר לספרים, רבים מכם תרמו להם רבות דרך הביקורת שלכם ודרך העבודה שביצענו יחד, שכמובן משתקפת בספרים. אך הרבה מעבר לכך, אתם משפחה, חברים, ויותר מכל מה שאפשר לבקש. אני עדיין צריך לצבוט את עצמי כדי להיות בטוח שזו מציאות ולא חלום. ולפרננדו: כולנו שותפים לדעה שאתה הסיבה לכך ש- Solid Quality Learning קמה לתחייה, ושבלעדיך היא לא הייתה קיימת. החזון שלך, הסבלנות, תשומת הלב, הלהט והלב הם השראה לכולנו. אין מילים שאוכל להגיד או לכתוב שיבטאו את הכרת התודה שיש בלבי לך ועד כמה אני מעריך את החברות בינינו.

אני מוצא שישנם שלושה מרכיבי מפתח המעצבים את הידע והיכולת של אדם: מורים, תלמידים ולהט. להט הוא הזרע שחייב לבוא מתוך האדם, ויש בי להט רב ל-SQL. בורכתי במורים נפלאים - לובור קולר וסנסאי יהודה פנטנוביץ'. לובור, הלהט והידע הנרחבים שלך של SQL והאופטימיזציה שלו, החיפוש שלך אחר ידע חדש, הצניעות שלך, הניצוץ בעיניך והחיבה שלך להיגיון, הם כולם מקור השראה בשבילי. אני אסיר תודה על שהכרנו, ואני מוקיר את החברות שלנו. סנסאי יהודה, על אף שייתכן שנראה לך כי לעולם שלך אין כל קשר ל-SQL, עבורי יש לו קשר חזק. העצה שלך, ההוראה, ההכוונה והחברות עזרה לי בעולם ה-SQL בדרכים רבות מכפי שתדע. אתה וסנסאי היגאונה (Higaonna) הנכם מקורות השראה אדירים בשבילי; אתם הוכחה חיה שאין דבר שהוא בלתי אפשרי - שאם נעבוד בשקידה ונשאף תמיד לשלמות, נוכל להשתפר ולהשיג דברים עצומים בחיים. ההתמקדות בפרטים קטנים, לעולם לא להיכנע, שליטה בהתרגשות על ידי המחשבה ש"זהו רק עוד יום במשרד", כנות, התמודדות עם הקטעים הרעים בחיים... אלו רק מספר דוגמאות של עצות שעזרו לי בדרכים רבות. כתיבה במשך שעות ארוכות

הייתה קשה ביותר, אך אני מייחס את העובדה שהצלחתי לעשות זאת בעיקר לך. תמיד חשבתי על הניסיון כאימון גוג'ו ארוך.

לסטודנטים שלי: למדתי, ואני ממשיך ללמוד המון דרך הוראה. למעשה, הוראה היא התשוקה האמיתית שלי, ואתם הסיבה העיקרית לכך שכתבתי ספרים אלו. בשנים האחרונות, טיילתי מסביב לעולם כדי ללמד. ביליתי זמן מועט מאוד בבית, והקרבתי הרבה לטובת התשוקה הזו. התקווה האמיתי שלי היא שלאחר שתקראו ספרים אלו, ניפגש בכיתת–הדוג'ו (חדר אימונים) המועדפת עלי ללימוד ותרגול של SQL.

להורי: החרטה היחידה שלי בכך שאני נוסע כל–כך הרבה, היא שזה בא על חשבון הזמן שלי אתכם. אני זוכר רגע סוריאליסטי שבאתם להרצאה שהעברתי על טבלאות מחולקות ואינדקסים ב– SQL Server 2005, כדי לראות אותי לאחר שלא התראינו שלושה חודשים. אני מתנצל ומקווה שאתם מבינים שאני צריך לרדוף אחר התשוקה שלי כדי להגיע להגשמה עצמית. אבא, תודה על כל העזרה במתמטיקה והיגיון. אמא, תפסיקי לצעוק על אבא כשהוא נותן לי חידה חדשה בשיחת טלפון טראנס–אטלאנטית; הוא לא יכול לעמוד בפיתוי וגם אני לא.

לילך, האהבה והעוגן שלי: הדבר היחידי ששומר על השפיות שלי כשאני רחוק מהבית הוא העובדה שאת איתי. אני חושב שאת היתר אני אגיד לך אישית; אנחנו לא רוצים להביך את עצמנו מול הקוראים.

אודות הספר

ספר זה מיועד לתוכניתני T-SQL ומנהלי מסדי נתונים מנוסים הנדרשים לכתוב שאילתות T-SQL. הספר עוסק בשאילתות T-SQL מתקדמות ומניח שיש לך כבר הבנה טובה של הרמה הבסיסית והבינונית של הנושאים (לפחות שנה של ניסיון בכתיבת קוד T-SQL) ושאתה מוכן להמשיך לרמה הבאה.

מבנה הספר

ספר זה הוא הראשון בשני כרכים, והוא מכסה שאילתות T-SQL מתקדמות. הכרך השני (שכרגע לא מתורגם לעברית) — Inside Microsoft SQL Server 2005: T-SQL Programming — מכסה תכנות מתקדם ב-T-SQL. הכרך השני מניח שקראת את הראשון או שיש לך רקע מקביל. לפרטים נוספים על המבנה של שני הכרכים, אנא פנה למבוא של ספר זה.

דרישות מערכת

כדי לבנות ולהריץ את דוגמאות הקוד בספר זה תידרש לרכיבי החומרה ולתוכנות הבאות:

- Microsoft Windows XP with Service Pack 2, Microsoft Windows Server 2003 with Service Pack 1, or Microsoft Windows 2000 with Service Pack 4
- Microsoft SQL Server 2005 Standard, Developer, or Enterprise Edition
- Microsoft Visual Studio 2005 Standard Edition or Microsoft Visual Studio 2005 Professional Edition
- 600-MHz Pentium or compatible processor (1-GHz Pentium recommended)
- 512 MB RAM (1 GB or more recommended)
- Video (800 by 600 or higher resolution) monitor with at least 256 colors (1024 by 768 High Color 16-bit recommended)
- CD−ROM or DVD−ROM drive
- Microsoft mouse or compatible pointing device

לפרטים נוספים על דרישות המערכת ועל התקנת SQL Server 2005, אנא פנה לפרק "Preparing to Install SQL Server 2005" ב−SQL Server Books Online (מותקן עם המוצר).

התקנת מסדי נתונים לדוגמה

ספר זה דורש ממך ליצור ולהשתמש במסדי נתונים, לדוגמה Northwind ו–pubs, אשר אינם מותקנים כברירת מחדל ב– SQL Server 2005. הקוד ליצירת מסדי נתונים אלו ניתן להורדה ממרכז ההורדות של מיקרוסופט בכתובת הבאה:

http://go.microsoft.com/fwlink/?LinkId=30196

לחלופין, תוכל להוריד את הקוד המייצר את מסדי נתונים אלו כחלק מדוגמאות הקוד עבור הספר. הוראות להורדה של דוגמאות הקוד עבור הספר יובאו מייד.

עדכונים

תוכל למצוא את העדכונים האחרונים של SQL Server בכתובת הבאה:

http://www.microsoft.com/sql/

תוכל למצוא מקורות הקשורים לספר בכתובת הבאה:

http://www.insidetsql.com/

דוגמאות קוד

כל דוגמאות הקוד הנידונות בספר זה ניתנות להורדה בכתובת הבאה:

http://www.insidetsql.com/

עברית

כן, אפשר ללמוד T-SQL בעזרת ספר הכתוב עברית. במהלך הקריאה והלימוד תראה שהדבר אפשרי.

מחבר הספר, איציק בן–גן, ישראלי יליד הארץ, המתגורר בה בדרך קבע, מעביר הרצאות וקורסים בכל העולם וגם בישראל, יודע כיצד לפנות לקהל מאזיניו דוברי העברית.

*** המונחים Attributes ו–Properties תורגמו שניהם למילה מאפיינים.**

תמיכה לספר זה

נעשו כל המאמצים כדי להבטיח את הדיוק של ספר זה ושל החומר המלווה אותו. עם זאת, שגיאות הן כנראה דבר בלתי נמנע. דף תיקונים ניתן למצוא בכתובת:

http://www.insidetsql.com/

התיקונים מתייחסים למהדורה האנגלית של הספר. תיקונים למהדורה העברית יצוינו בנפרד.

אם יש לך הערות, שאלות, או רעיונות בנוגע לספר זה, תוכל ליצור איתי קשר באתר הספר בכתובת לעיל.

1

עיבוד לוגי של שאילתות

מהתבוננות במומחים בתחומים שונים, ניתן להבחין כי יש נוהג המשותף לכולם – שליטה בעקרונות ובשיטות היסוד. בצורה זו או אחרת, כל המקצוענים מתמודדים עם פתרון בעיות. כל הפתרונות לבעיות, מורכבות ככל שיהיו, משלבים תערובת של שיטות מפתח. אם ברצונך להתמחות במקצוע, עליך לבנות את הידע שלך על יסודות מוצקים. השקע מאמצים רבים בשיפור שיטות העבודה שלך; שלוט בעקרונות היסוד, ותוכל לפתור כל בעיה.

ספר זה עוסק בשאילתות ב– **Transact-SQL (T-SQL)** – לימוד טכניקות מפתח ושימוש בהן כדי לפתור בעיות. אינני יכול לחשוב על דרך טובה יותר להתחיל את הספר מאשר בפרק על העקרונות של עיבוד לוגי של שאילתות. אני מוצא שפרק זה הוא החשוב ביותר בספר – לא רק מכיוון שהוא מכסה את היסודות של עיבוד שאילתות, אלא גם מכיוון שתכנות ב–SQL שונה מאוד, תפיסתית, מכל סוג תכנות אחר.

הניב של Microsoft SQL Server - Transact-SQL – תואם לתקן של ANSI. Microsoft SQL Server 2000 תואם לתקן ANSI SQL:1992 ברמת ה–Entry, ו– Microsoft SQL Server 2005 מיישם מספר מאפיינים חשובים של ANSI SQL:1999 ושל ANSI SQL:2003.

לאורך הספר אשתמש לעיתים במונח SQL ולעיתים במונח T-SQL. כאשר אדון בהיבטים של השפה אשר נובעים מ– ANSI SQL ורלוונטיים לרוב הניבים, אשתמש לרוב במונח SQL. כאשר אדון בהיבטים של השפה מתוך מחשבה על יישום של SQL Server, אשתמש לרוב במונח T-SQL. יש לציין כי השם הרשמי של השפה הוא Transact-SQL, על אף שלרוב היא נקראת T-SQL. מרבית התוכניתנים, ואני ביניהם, חשים נוח יותר לקרוא לשפה T-SQL, כך שעשיתי החלטה מודעת להשתמש במונח זה לאורך כל הספר.

המקור להגייה של SQL

מקצועני מסדי נתונים דוברי אנגלית רבים הוגים את המילה SQL כ–sequel, למרות שההגייה הנכונה של שם השפה היא S-Q-L ("אֶס קְיוּ אֶל").

ניתן לנסות להציע ניחושים מלומדים לסיבה להגייה השגויה. הניחוש שלי הוא שהסיבות הן גם היסטוריות וגם לשוניות.

באשר לסיבות היסטוריות, בשנות ה–70 של המאה העשרים, IBM פיתחה שפה שנקראה SEQUEL, ראשי התיבות של Structured English QUEry Language (שפת אנגלית מובנית לשאילתות). השפה עוצבה כדי לשלוט בנתונים שאוחסנו במערכת מסדי נתונים שנקראה מערכת R, אשר הייתה מבוססת על המודל של דר' אדגר פ. קוד למערכות ניהול מסדי נתונים רלציונים (RDBMS). מאוחר יותר, ראשי התיבות SEQUEL קוצרו ל–SQL בעקבות ויכוח על סימן מסחרי. ANSI אימצה את SQL כתקן בשנת 1986, ו–ISO עשה כמותה בשנת 1987. ANSI הכריזה שההגייה הרשמית של שם השפה היא "אֶס קְיוּ אֶל", אך נראה כי עובדה זו אינה ידועה לכל.

באשר לסיבות לשוניות, ההגייה sequel פשוט "זורמת" יותר, בעיקר לדוברי אנגלית. אני חייב לומר שאני משתמש בהגייה זו לעיתים קרובות בעצמי, בדיוק מאותה סיבה.

לעיתים ניתן לנחש באיזו הגייה אנשים משתמשים על ידי עיון בדברים שכתבו. מישהו שכותב: "an SQL Server" משתמש, סביר להניח, בצורת ההגייה הנכונה, בעוד מישהו שכותב "a SQL Server" משתמש, כפי הנראה, בצורת ההגייה השגויה.

מידע נוסף: אני ממליץ לקרוא על ההיסטוריה של SQL ועל ההגייה של שם השפה, נושא מרתק בעיני, בכתובת: http://www.wikimirror.com/SQL. סקירת ההיסטוריה של SQL באתר Wikimirror ובפרק זה מבוססים על מאמר מהאנציקלופדיה החופשית ויקפדיה.

ישנם היבטים ייחודיים רבים של תכנות SQL, כגון חשיבה בסטים, סדר העיבוד הלוגי של פסוקיות השאילתה ולוגיקת שלושת–הערכים. ניסיון לתכנת ב–SQL ללא הידע הזה הוא מרשם לקוד ארוך, בעל ביצועים גרועים וקשה לתחזוקה. המטרה של פרק זה היא לסייע לך להבין את שפת SQL כפי שהמעצבים שלה חזו אותה. עליך ליצור שורשים עמוקים עליהם ייבנה כל היתר. כאשר הדבר יהיה רלוונטי, אציין במפורש פסוקיות אשר הינן ספציפיות ל–T-SQL.

לאורך כל הספר אכסה בעיות מורכבות וטכניקות מתקדמות. אך בפרק זה, כפי שציינתי, אעסוק אך ורק בעקרונות היסוד של השאילתות. לאורך כל הספר אתמקד בביצועים. אך בפרק זה אעסוק אך ורק בהיבטים הלוגיים של עיבוד שאילתות. אבקש ממך לעשות מאמץ, כאשר אתה קורא פרק זה, ולא לחשוב בשלב זה על ביצועים. כמה מהשלבים של עיבוד לוגי של שאילתות שאציג בפרק זה עשויים להיראות מאוד לא יעילים. אך זכור שבפועל, העיבוד הפיסי של שאילתות עשוי להיות שונה מאוד מזה הלוגי.

הרכיב ב– SQL Server האחראי על הפקת תוכנית העבודה (execution plan) עבור שאילתה הוא ה– query optimizer. ה–optimizer קובע באיזה סדר לגשת לטבלאות, באילו שיטות גישה ואינדקסים להשתמש, איזה אלגוריתמים של join ליישם וכו'. ה–optimizer מייצר מספר תוכניות עבודה אפשריות ובוחר בזו בעלת העלויות הנמוכות ביותר. לשלבים בעיבוד הלוגי של שאילתה יש סדר מאוד מסוים. מצד שני, ה–optimizer יכול לעיתים קרובות לעשות קיצורי דרך בתוכנית העבודה הפיסית שהוא מייצר. כמובן שהוא יבצע קיצורי דרך אך ורק אם יוכל להבטיח שהתוצאה שתתקבל תהיה הנכונה – במילים אחרות, תוצאה זהה לזו שתקבל על ידי מעקב אחר השלבים של העיבוד הלוגי. לדוגמה, כדי להשתמש באינדקס, ה–optimizer עשוי להחליט להפעיל סינון מוקדם הרבה יותר מאשר כפי שמוכתב על ידי העיבוד הלוגי.

מהסיבות המוזכרות לעיל, חשוב לבצע הבחנה ברורה בין עיבוד לוגי לעיבוד פיסי של שאילתה.

ללא עיכובים נוספים, הבה נצלול לשלבי העיבוד הלוגי של שאילתה.

שלבי עיבוד לוגי של שאילתה

סעיף זה מציג את השלבים המעורבים בעיבוד הלוגי של שאילתה. ראשית אתאר בקצרה כל שלב. לאחר מכן, בסעיפים הבאים, אתאר את השלבים בפירוט רב הרבה יותר ואיישם אותם בשאילתה לדוגמה. תוכל להשתמש בחלק זה כמקור מידע בכל פעם שתצטרך להיזכר בסדר ובמשמעות של השלבים השונים.

קטע–קוד 1–1 מכיל צורה כללית של שאילתה, כאשר לכל שלב מצורף מספר בהתאם לסדר בו כל פסוקית מעובדת לוגית.

קטע–קוד 1–1: מספור שלבי עיבוד לוגי של שאילתה

```
(8)  SELECT (9) DISTINCT (11) <TOP_specification> <select_list>
(1)  FROM <left_table>
(3)    <join_type> JOIN <right_table>
(2)      ON <join_condition>
(4)  WHERE <where_condition>
(5)  GROUP BY <group_by_list>
(6)  WITH {CUBE | ROLLUP}
(7)  HAVING <having_condition>
(10) ORDER BY <order_by_list>
```

הפן הבולט הראשון של SQL השונה משפות תכנות אחרות הוא הסדר בו מעובד הקוד. במרבית שפות התכנות, הקוד מעובד בסדר שבו הוא נכתב. ב–SQL, הפסוקית הראשונה

שעוברת עיבוד היא הפסוקית FROM, בעוד שהפסוקית SELECT, המופיעה ראשונה, מעובדת כמעט אחרונה.

כל שלב מייצר טבלה וירטואלית המשמשת כקלט לשלב הבא. טבלאות וירטואליות אלו אינן זמינות ללקוח (יישום לקוח או שאילתה חיצונית). אך ורק הטבלה המיוצרת על ידי השלב הסופי מוחזרת ללקוח. אם פסוקית מסוימת אינה נכללת בשאילתה, העיבוד מדלג על השלב הקשור בפסוקית. כעת אתאר בקצרה את השלבים הלוגיים השונים המיושמים הן ב- SQL Server 2000 והן ב- SQL Server 2005. בהמשך הפרק אדון בנפרד בשלבים שנוספו ב- SQL Server 2005.

תיאור קצר של שלבי העיבוד הלוגי של שאילתה

אל תדאג אם תיאור השלבים לא ייראה כל-כך הגיוני כעת. תיאור זה נכתב כמקור מידע. הסעיפים שיבואו לאחר השאילתה לדוגמה יכסו את השלבים הללו בצורה מעמיקה הרבה יותר.

1. **FROM:** מכפלה קרטזית (cross join) מבוצעת בין שתי הטבלאות הראשונות בפסוקית FROM, וכתוצאה מכך נוצרת טבלה וירטואלית VT1.
2. **ON:** מסנן ON מיושם על VT1. רק שורות אשר עבורן תנאי join (<join_condition>) הוא TRUE נכנסות לטבלה VT2.
3. **OUTER (join):** אם בשאילתה נכלל OUTER JOIN (בניגוד ל- CROSS JOIN או ל- INNER JOIN), שורות מהטבלה השמורה (או מהטבלאות השמורות) אשר עבורן לא נמצאה התאמה מתווספות לטבלה VT2 כשורות חיצוניות. טבלה VT3 נוצרת. אם בפסוקית FROM מופיעות יותר משתי טבלאות, שלבים 1 עד 3 מיושמים שוב ושוב בין התוצאה של ה-join האחרון לבין הטבלה הבאה בפסוקית FROM, עד שכל הטבלאות מעובדות.
4. **WHERE:** מסנן WHERE מיושם על VT3. רק שורות אשר עבורן תנאי where (<where_condition>) הוא TRUE נכנסות לטבלה VT4.
5. **GROUP BY:** השורות מטבלה VT4 מסודרות בקבוצות בהתאם לרשימת הטורים המצוינת בפסוקית GROUP BY. טבלה VT5 נוצרת.
6. **CUBE | ROLLUP:** קבוצות-על (קבוצות של קבוצות) מתווספות לשורות מטבלה VT5. טבלה VT6 נוצרת.
7. **HAVING:** מסנן HAVING מיושם על VT6. רק קבוצות אשר עבורן תנאי having (<having_condition>) הוא TRUE נכנסות לטבלה VT7.
8. **SELECT:** רשימת SELECT מעובדת וטבלה VT8 נוצרת.
9. **DISTINCT:** שורות כפולות מוסרות מטבלה VT8. טבלה VT9 נוצרת.

10. **ORDER BY:** השורות מטבלה VT9 ממוינות לפי רשימת הטורים שמוגדרת בפסוקית ORDER BY. נוצר סמן (VC10).

11. **TOP:** מספר או אחוז השורות שהוגדר נבחר מתחילת הסמן VC10. טבלה VT11 נוצרת ומוחזרת ללקוח.

שאילתה לדוגמה מבוססת על לקוחות/הזמנות

כדי לתאר את שלבי העיבוד הלוגי בפירוט, אעקוב יחד אתך אחר מהלך של שאילתה לדוגמה. ראשית הרץ את הקוד המוצג בקטע-קוד 2-1 כדי ליצור את הטבלאות Customers ו-Orders ולהכניס בהן נתונים לדוגמה. טבלאות 1-1 ו-2-1 מציגות את התוכן של Customers ושל Orders.

קטע-קוד 2-1: שפה להגדרת נתונים (DDL) ונתונים לדוגמה עבור Customers ו-Orders

```
SET NOCOUNT ON;
USE tempdb;
GO
IF OBJECT_ID('dbo.Orders') IS NOT NULL
  DROP TABLE dbo.Orders;
GO
IF OBJECT_ID('dbo.Customers') IS NOT NULL
  DROP TABLE dbo.Customers;
GO
CREATE TABLE dbo.Customers
(
  customerid  CHAR(5)     NOT NULL PRIMARY KEY,
  city        VARCHAR(10) NOT NULL
);

INSERT INTO dbo.Customers(customerid, city) VALUES('FISSA', 'Madrid');
INSERT INTO dbo.Customers(customerid, city) VALUES('FRNDO', 'Madrid');
INSERT INTO dbo.Customers(customerid, city) VALUES('KRLOS', 'Madrid');
INSERT INTO dbo.Customers(customerid, city) VALUES('MRPHS', 'Zion');

CREATE TABLE dbo.Orders
(
  orderid    INT         NOT NULL PRIMARY KEY,
  customerid CHAR(5)     NULL     REFERENCES Customers(customerid)
);
```

```
INSERT INTO dbo.Orders(orderid, customerid) VALUES(1, 'FRNDO');
INSERT INTO dbo.Orders(orderid, customerid) VALUES(2, 'FRNDO');
INSERT INTO dbo.Orders(orderid, customerid) VALUES(3, 'KRLOS');
INSERT INTO dbo.Orders(orderid, customerid) VALUES(4, 'KRLOS');
INSERT INTO dbo.Orders(orderid, customerid) VALUES(5, 'KRLOS');
INSERT INTO dbo.Orders(orderid, customerid) VALUES(6, 'MRPHS');
INSERT INTO dbo.Orders(orderid, customerid) VALUES(7, NULL);
```

טבלה 1–1: תוכן של טבלת Customers

customerid	*city*
FISSA	Madrid
FRNDO	Madrid
KRLOS	Madrid
MRPHS	Zion

טבלה 1–2: תוכן של טבלת Orders

orderid	*customerid*
1	FRNDO
2	FRNDO
3	KRLOS
4	KRLOS
5	KRLOS
6	MRPHS
7	NULL

לצורך הדגמה אשתמש בשאילתה המוצגת בקטע–קוד 1–3. השאילתה מחזירה לקוחות מהעיר מדריד אשר ביצעו פחות משלוש הזמנות (כולל אפס הזמנות), ואת מספר ההזמנות שביצעו. התוצאה ממוינת לפי מספר הזמנות, מהנמוך לגבוה. הפלט של שאילתה זו מוצג בטבלה 1–3.

קטע–קוד 3–1: שאילתה – לקוחות מדריד שלהם פחות משלוש הזמנות

```
SELECT C.customerid, COUNT(O.orderid) AS numorders
FROM dbo.Customers AS C
  LEFT OUTER JOIN dbo.Orders AS O
    ON C.customerid = O.customerid
WHERE C.city = 'Madrid'
GROUP BY C.customerid
HAVING COUNT(O.orderid) < 3
ORDER BY numorders;
```

טבלה 3–1: פלט – לקוחות מדריד שלהם פחות משלוש הזמנות

customerid	*numorders*
FISSA	0
FRNDO	2

הן FISSA והן FRNDO הם לקוחות מהעיר מדריד אשר ביצעו פחות משלוש הזמנות. בחן את השאילתה ונסה לקרוא אותה תוך כדי מעקב אחר השלבים המתוארים בקטע–קוד 1–1 והסעיף "תיאור קצר של שלבי העיבוד הלוגי של שאילתה". אם זו הפעם הראשונה שהנך חושב על שאילתה במונחים אלו, ייתכן שהדבר יהיה מבלבל. הסעיף הבא יסייע לך להיכנס לעובי הקורה.

פירוט שלבי העיבוד הלוגי של שאילתה

סעיף זה מתאר את שלבי העיבוד הלוגי של שאילתה בפירוט על ידי יישומם בשאילתה לדוגמה.

שלב 1: ביצוע מכפלה קרטזית (Cross Join)

מכפלה קרטזית (cross join או join בלתי מוגבל) מבוצעת בין שתי הטבלאות הראשונות המופיעות בפסוקית FROM, וכתוצאה מכך נוצרת טבלה VT1. VT1 מכילה שורה לכל שילוב אפשרי של שורה מהטבלה השמאלית ושורה מהטבלה הימנית. אם הטבלה השמאלית מכילה n שורות והטבלה הימנית מכילה m שורות, טבלה VT1 תכיל n x m שורות. הטורים בטבלה VT1 מקבלים תחילית של שמות טבלאות המקור שלהם (או כינויי טבלה, אם הגדרת כאלה בשאילתה). בשלבים הבאים (שלב 2 ואילך), התייחסות לשם טור רב–משמעי (מופיע ביותר מטבלת קלט אחת) חייב לכלול תחילית של שם טבלה (לדוגמה,

C.customerid). אין הכרח להכליל תחילית של שם טבלה עבור טורים המופיעים בקלט אחד בלבד (למשל, O.orderid או פשוט orderid).

יישם את שלב 1 על השאילתה לדוגמה (מוצג בקטע–קוד 3–1):

```
FROM Customers AS C ... JOIN Orders AS O
```

כתוצאה תקבל את הטבלה הווירטואלית VT1 המוצגת בטבלה 4–1 עם 28 שורות (7x4).

טבלה 4–1: טבלה וירטואלית VT1 המוחזרת משלב 1

C.customerid	*C.city*	*O.orderid*	*O.customerid*
FISSA	Madrid	1	FRNDO
FISSA	Madrid	2	FRNDO
FISSA	Madrid	3	KRLOS
FISSA	Madrid	4	KRLOS
FISSA	Madrid	5	KRLOS
FISSA	Madrid	6	MRPHS
FISSA	Madrid	7	NULL
FRNDO	Madrid	1	FRNDO
FRNDO	Madrid	2	FRNDO
FRNDO	Madrid	3	KRLOS
FRNDO	Madrid	4	KRLOS
FRNDO	Madrid	5	KRLOS
FRNDO	Madrid	6	MRPHS
FRNDO	Madrid	7	NULL
KRLOS	Madrid	1	FRNDO
KRLOS	Madrid	2	FRNDO
KRLOS	Madrid	3	KRLOS
KRLOS	Madrid	4	KRLOS
KRLOS	Madrid	5	KRLOS
KRLOS	Madrid	6	MRPHS
KRLOS	Madrid	7	NULL
MRPHS	Zion	1	FRNDO

C.customerid	C.city	O.orderid	O.customerid
MRPHS	Zion	2	FRNDO
MRPHS	Zion	3	KRLOS
MRPHS	Zion	4	KRLOS
MRPHS	Zion	5	KRLOS
MRPHS	Zion	6	MRPHS
MRPHS	Zion	7	NULL

שלב 2: יישום המסנן ON (Join Condition)

מסנן ON הוא הראשון משלושה מסננים אפשריים (ON, WHERE ו–HAVING) אשר ניתן להגדיר בשאילתה. הביטוי הלוגי במסנן ON מיושם על כל השורות בטבלה הווירטואלית המוחזרת על ידי השלב הקודם (VT1). רק שורות שעבורן תנאי join הוא TRUE הופכות לחלק מהטבלה הווירטואלית המוחזרת על ידי השלב הנוכחי (VT2).

לוגיקת שלושת הערכים

אבקש לסטות מעט מהנושא כדי לכסות היבטים חשובים של SQL הקשורים לשלב זה. הערכים האפשריים של ביטוי לוגי ב–SQL הם TRUE, FALSE ו–UNKNOWN ומכאן השם לוגיקת שלושת–הערכים. לוגיקת שלושת–הערכים היא ייחודית ל–SQL. ביטויים לוגיים במרבית שפות התכנות יכולים להיות רק TRUE או FALSE. הערך הלוגי UNKNOWN ב–SQL מופיע בדרך–כלל בביטוי לוגי שבו מעורב NULL (למשל, הערך הלוגי של כל אחד משלושת הביטויים הבאים הוא UNKNOWN: NULL > 42; NULL = NULL; X + NULL > Y). הערך הייחודי NULL מייצג בדרך–כלל ערך חסר או לא רלוונטי. כאשר משווים ערך חסר לערך אחר (אפילו ל–NULL אחר), התוצאה הלוגית היא UNKNOWN.

הטיפול בתוצאות לוגיות שהן UNKNOWN יכול להיות מאוד מבלבל. בעוד ש– NOT TRUE הוא FALSE, ו– NOT FALSE הוא TRUE, ההופכי של UNKNOWN (NOT UNKNOWN) הוא עדיין UNKNOWN.

תוצאות לוגיות שהן UNKNOWN וכן NULLs מטופלים בצורה לא עקבית באלמנטים שונים של השפה. למשל, כל מסנני השאילתה (ON, WHERE ו–HAVING) מתייחסים ל–UNKNOWN בצורה דומה ל–FALSE. שורה אשר עבורה מסנן הוא UNKNOWN נמחקת מהתוצאה. מצד שני, ערך UNKNOWN באילוץ CHECK מטופל למעשה

כ–TRUE. נניח שיש לך בטבלה אילוץ CHECK הדורש שטור השכר יהיה גדול מאפס. שורה אשר מוכנסת לטבלה עם שכר NULL מתקבלת, מכיוון ש– (NULL > 0) הוא UNKNOWN, והוא מטופל כ–TRUE באילוץ CHECK.

השוואה בין שני NULL במסננים מניבה UNKNOWN, אשר כפי שהזכרתי קודם מטופל כ–FALSE – כאילו NULL אחד שונה מהשני.

מצד שני, אילוץ UNIQUE, מיון וקיבוץ מתייחסים ל–NULLs כשווים:

- אינך יכול להכניס לטבלה שתי שורות עם NULL בטור שמוגדר לו אילוץ UNIQUE.
- פסוקית GROUP BY מקבצת את כל ה–NULLs לקבוצה אחת.
- פסוקית ORDER BY ממיינת את כל ה–NULLs יחד.

בקיצור, כדי לחסוך לעצמך עוגמת נפש, כדאי שתהיה מודע לדרך בה אלמנטים שונים של השפה מטפלים בתוצאות לוגיות של UNKNOWN וב–NULLs.

יישם את שלב 2 על השאילתה לדוגמה:

```
ON C.customerid = O.customerid
```

טבלה 5–1 מציגה את הערך של הביטוי הלוגי במסנן ON עבור השורות מטבלה VT1.

טבלה 5–1: תוצאות לוגיות של מסנן ON מיושמות על שורות מטבלה VT1

Match?	*C.customerid*	*C.city*	*O.orderid*	*O.customerid*
FALSE	FISSA	Madrid	1	FRNDO
FALSE	FISSA	Madrid	2	FRNDO
FALSE	FISSA	Madrid	3	KRLOS
FALSE	FISSA	Madrid	4	KRLOS
FALSE	FISSA	Madrid	5	KRLOS
FALSE	FISSA	Madrid	6	MRPHS
UNKNOWN	FISSA	Madrid	7	NULL
TRUE	FRNDO	Madrid	1	FRNDO
TRUE	FRNDO	Madrid	2	FRNDO
FALSE	FRNDO	Madrid	3	KRLOS

Match?	C.customerid	C.city	O.orderid	O.customerid
FALSE	FRNDO	Madrid	4	KRLOS
FALSE	FRNDO	Madrid	5	KRLOS
FALSE	FRNDO	Madrid	6	MRPHS
UNKNOWN	FRNDO	Madrid	7	NULL
FALSE	KRLOS	Madrid	1	FRNDO
FALSE	KRLOS	Madrid	2	FRNDO
TRUE	KRLOS	Madrid	3	KRLOS
TRUE	KRLOS	Madrid	4	KRLOS
TRUE	KRLOS	Madrid	5	KRLOS
FALSE	KRLOS	Madrid	6	MRPHS
UNKNOWN	KRLOS	Madrid	7	NULL
FALSE	MRPHS	Zion	1	FRNDO
FALSE	MRPHS	Zion	2	FRNDO
FALSE	MRPHS	Zion	3	KRLOS
FALSE	MRPHS	Zion	4	KRLOS
FALSE	MRPHS	Zion	5	KRLOS
TRUE	MRPHS	Zion	6	MRPHS
UNKNOWN	MRPHS	Zion	7	NULL

רק שורות אשר עבורן תנאי join הוא TRUE נכנסות ל–VT2 – טבלת הקלט הווירטואלית של השלב הבא, המוצגת בטבלה 6–1.

טבלה 6–1: טבלה וירטואלית VT2 המוחזרת משלב 2

Match?	C.customerid	C.city	O.orderid	O.customerid
TRUE	FRNDO	Madrid	1	FRNDO
TRUE	FRNDO	Madrid	2	FRNDO
TRUE	KRLOS	Madrid	3	KRLOS
TRUE	KRLOS	Madrid	4	KRLOS
TRUE	KRLOS	Madrid	5	KRLOS
TRUE	MRPHS	Zion	6	MRPHS

שלב 3: הוספת שורות חיצוניות

שלב זה רלוונטי רק עבור outer join. עבור outer join, אתה מסמן את אחת מטבלאות הקלט או את שתיהן **כטבלה שמורה** על ידי ציון סוג ה– outer join (LEFT, RIGHT, או FULL). המשמעות של סימון טבלה כשמורה היא שאתה מעוניין שכל שורותיה יוחזרו, אפילו אם הן סוננו החוצה על ידי תנאי ה–join. Left outer join מסמן את הטבלה השמאלית כשמורה, right outer join מסמן את הימנית, ו– full outer join מסמן את שתיהן. שלב 3 מחזיר את השורות מטבלה VT2 וכן שורות מהטבלה השמורה אשר עבורן לא נמצאה התאמה בשלב 2. שורות נוספות אלו נקראות **שורות חיצוניות**. במאפיינים (ערכי טור) של השורות החיצוניות השייכים לטבלאות הלא שמורות יופיעו ערכי NULL. כתוצאה נוצרת טבלה וירטואלית VT3.

בדוגמה שלנו, הטבלה השמורה היא Customers:

```
Customers AS C LEFT OUTER JOIN Orders AS O
```

רק הלקוח FISSA לא מצא הזמנה תואמת (לא היה חלק מטבלה VT2). לפיכך, FISSA מתווסף לשורות מהשלב הקודם עם NULLs במאפייני Orders וכתוצאה מכך נוצרת טבלה וירטואלית VT3 (מוצגת בטבלה 7–1).

טבלה 7–1: טבלה וירטואלית VT3 מוחזרת משלב 3

C.customerid	*C.city*	*O.orderid*	*O.customerid*
FRNDO	Madrid	1	FRNDO
FRNDO	Madrid	2	FRNDO
KRLOS	Madrid	3	KRLOS
KRLOS	Madrid	4	KRLOS
KRLOS	Madrid	5	KRLOS
MRPHS	Zion	6	MRPHS
FISSA	Madrid	NULL	NULL

שים לב: אם יותר משתי טבלאות מאוחדות, שלבים 1 עד 3 ייושמו בין VT3 לבין הטבלה השלישית מהפסוקית FROM. תהליך זה יחזור על עצמו אם טבלאות נוספות מופיעות בפסוקית FROM, והטבלה הווירטואלית הסופית תשמש כקלט לשלב הבא.

שלב 4: יישום המסנן WHERE

מסנן WHERE מיושם על כל השורות בטבלה הווירטואלית שהוחזרה מהשלב הקודם. רק שורות שעבורן תנאי where (<where condition>) הוא TRUE הופכות לחלק מהטבלה הווירטואלית המוחזרת משלב זה (VT4).

אזהרה: מכיוון שהנתונים אינם מקובצים עדיין, אינך יכול להשתמש עדיין במסנני צבירה (אגרגציה) – למשל, אינך יכול לכתוב: WHERE orderdate = MAX(orderdate). כמו כן, אינך יכול להתייחס לכינויי טור הנוצרים על ידי רשימת ה–SELECT שכן רשימת ה–SELECT עדיין לא עברה עיבוד – למשל, אינך יכול לכתוב: SELECT YEAR (orderdate) AS orderyear … WHERE orderyear > 2000.

היבט מבלבל של שאילתות המכילות פסוקית OUTER JOIN הוא האם לציין ביטוי לוגי במסנן ON או במסנן WHERE. ההבדל העיקרי בין השניים הוא ש–ON מיושם לפני הוספת שורות חיצוניות (שלב 3), בעוד ש–WHERE מיושם אחרי שלב 3. הסרת שורה מטבלה שמורה על ידי מסנן ON אינה סופית מכיוון ששלב 3 יוסיף אותה בחזרה, בעוד שהסרת שורה על ידי מסנן WHERE היא סופית. זכור עובדה זו, שתסייע לך לעשות את ההחלטה הנכונה.

למשל, נניח שאתה מעוניין להחזיר לקוחות מסוימים ואת ההזמנות שלהם מטבלאות Customers ו–Orders. הלקוחות שאתה מעוניין להחזיר הם רק לקוחות מהעיר מדריד, הן אלה שביצעו הזמנות והן אלה שלא. Outer join מיועד בדיוק לבקשות מסוג זה. אתה מבצע left outer join בין Customers ו–Orders ועל ידי כך מסמן את Customers כטבלה שמורה. כדי שתוכל להחזיר לקוחות שלא ביצעו הזמנות, עליך לציין את המתאם (קורלציה) בין לקוחות להזמנות בפסוקית ON (ON C.customerid = O.customerid). לקוחות ללא הזמנות מוסרים בשלב 2 אך מתווספים חזרה בשלב 3 כשורות חיצוניות. לעומת זאת, מכיוון שהנך מעוניין להשאיר רק שורות עבור לקוחות מהעיר מדריד, בין אם ביצעו הזמנות ובין אם לאו, עליך לציין את מסנן העיר בפסוקית WHERE (WHERE C.city='Madrid'). ציון מסנן העיר בפסוקית ON יגרום לכך שלקוחות שאינם ממדריד יתווספו לתוצאה על ידי שלב 3.

טיפ: יש הבדל לוגי בין הפסוקיות ON ו–WHERE רק בעת השימוש ב– outer join. כאשר משתמשים ב– inner join, אין זה משנה היכן אתה מציין את הביטויים הלוגיים שלך מכיוון שהעיבוד מדלג על שלב 3. המסננים מיושמים אחד אחר השני ללא שלב ביניים ביניהם. ישנו יוצא דופן אחד הרלוונטי רק כאשר משתמשים באפשרות של GROUP BY ALL. אדון באפשרות זו בסעיף הבא, המכסה את שלב GROUP BY.

יישם את המסנן על השאילתה לדוגמה:

```
WHERE C.city = 'Madrid'
```

השורה עבור הלקוח MRPHS מטבלה VT3 מוסרת מכיוון שאינו ממדריד, ונוצרת הטבלה הווירטואלית VT4, המוצגת בטבלה 8–1.

טבלה 8–1: טבלה וירטואלית VT4 המוחזרת משלב 4

C.customerid	*C.city*	*O.orderid*	*O.customerid*
FRNDO	Madrid	1	FRNDO
FRNDO	Madrid	2	FRNDO
KRLOS	Madrid	3	KRLOS
KRLOS	Madrid	4	KRLOS
KRLOS	Madrid	5	KRLOS
FISSA	Madrid	NULL	NULL

שלב 5: קיבוץ

השורות מהטבלה המוחזרת על ידי השלב הקודם מסודרות בקבוצות. כל צירוף ייחודי של ערכים ברשימת הטורים המופיעה בפסוקית GROUP BY יוצר קבוצה. כל שורת בסיס מהשלב הקודם משובצת לקבוצה אחת ויחידה. טבלה וירטואלית VT5 נוצרת. VT5 מורכבת משתי יחידות: יחידת **הקבוצות** המורכבת מהקבוצות עצמן, ויחידת **המידע הגולמי** המורכבת משורות הבסיס המצורפות מהשלב הקודם.

יישם את שלב 5 על השאילתה לדוגמה:

```
GROUP BY C.customerid
```

מתקבלת הטבלה הווירטואלית VT5 המוצגת בטבלה 9–1.

טבלה 9–1: טבלה וירטואלית VT5 המוחזרת משלב 5

Groups	*Raw*			
C.customerid	*C.customerid*	*C.city*	*O.orderid*	*O.customerid*
FRNDO	FRNDO	Madrid	1	FRNDO
	FRNDO	Madrid	2	FRNDO

Groups	*Raw*			
C.customerid	*C.customerid*	*C.city*	*O.orderid*	*O.customerid*
KRLOS	KRLOS	Madrid	3	KRLOS
	KRLOS	Madrid	4	KRLOS
	KRLOS	Madrid	5	KRLOS
FISSA	FISSA	Madrid	NULL	NULL

אם פסוקית GROUP BY מצוינת בשאילתה, כל השלבים הבאים (HAVING, SELECT וכו') יכולים לציין רק ביטויים שתוצאתם היא ערך סקלרי (יחידני) עבור קבוצה. במילים אחרות, התוצאות יכולות להיות או טור/ביטוי המשתתף ברשימת ה– GROUP BY, למשל, C.customerid, או פונקציית צבירה (אגרגציה) כגון COUNT(O.orderid). ההיגיון מאחורי מגבלה זו הוא שבתוצאה הסופית תיווצר שורה יחידה לכל קבוצה (אלא אם כן היא סוננה החוצה). בחן את VT5 בטבלה 1–9, ונסה לחשוב מה אמורה השאילתה להחזיר עבור הלקוח FRNDO אם רשימת ה–SELECT שציינת הייתה SELECT C.customerid, O.orderid. ישנם שני ערכי orderid שונים בקבוצה; לפיכך, התשובה איננה דטרמיניסטית. SQL אינו מאפשר בקשה כזו. מצד שני, אם תציין: SELECT C.customerid, COUNT(O.orderid) AS numorders, התשובה עבור FRNDO תהיה דטרמיניסטית: 2.

שים לב: מותר לך גם לקבץ לפי התוצאה של ביטוי – למשל, GROUP BY YEAR(orderdate). אם תעשה כך, כשאתה עובד ב– SQL Server 2000, כל השלבים הבאים לא יוכלו לבצע כל מניפולציה נוספת לביטוי ה– GROUP BY, אלא אם כן זהו טור בסיס. למשל, הבקשה הבאה אינה מותרת ב– SQL Server 2000:

```
SELECT YEAR(orderdate) + 1 AS nextyear … GROUP BY YEAR(orderdate)
```

ב– SQL Server 2005 מגבלה זו הוסרה.

שלב זה מתייחס ל–NULLs כשווים. כלומר, כל ה–NULLs מקובצים לקבוצה אחת ממש כמו ערך ידוע.

כפי שציינתי קודם, הקלט לשלב GROUP BY הוא הטבלה הווירטואלית המוחזרת על ידי השלב הקודם (VT4). אם ציינת GROUP BY ALL, קבוצות אשר הוסרו על ידי השלב הרביעי (מסנן WHERE) מתווספות לטבלת התוצאה הווירטואלית של שלב זה (VT5) עם סט ריק ביחידת המידע הגולמי. זהו המקרה היחיד בו יש הבדל בין ציון ביטוי לוגי בפסוקית ON או בפסוקית WHERE כאשר משתמשים ב– inner join. אם תשנה את הדוגמה שלנו כך שתשתמש ב– GROUP BY ALL C.customerid במקום ב– GROUP BY C.customerid, תמצא שהלקוח MRPHS, שהוסר על ידי מסנן WHERE,

יתווסף ליחידת הקבוצות של VT5, בצירוף סט ריק ביחידת המידע הגולמי. פונקציית הצבירה COUNT באחד מהשלבים הבאים תהיה אפס עבור קבוצה כזו, בעוד שכל פונקציות הצבירה האחרות (SUM, AVG, MIN, MAX) יהיו NULL.

שים לב: האפשרות GROUP BY ALL היא מאפיין לא סטנדרטי, שנשאר במוצר מסיבות היסטוריות. הוא מציג סוגיות סמנטיות רבות כאשר מיקרוסופט מוסיפה מאפייני T-SQL חדשים. על אף שמאפיין זה נתמך לחלוטין ב- SQL Server 2005, ייתכן שתעדיף להימנע משימוש בו מכיוון שמיקרוסופט עשויה להפסיק לתמוך בפסוקית זו בעתיד.

שלב 6: יישום האפשרות CUBE או ROLLUP

אם מצוין CUBE או ROLLUP, נוצרות קבוצות-על ומתווספות לקבוצות בטבלה הווירטואלית המוחזרת מהשלב הקודם. טבלה וירטואלית VT6 נוצרת.

בדוגמה שלנו נדלג על שלב 6 מכיוון ש-CUBE ו-ROLLUP אינם מופיעים בשאילתה לדוגמה. CUBE ו-ROLLUP יכוסו בשלב מאוחר יותר בספר בפרק 6.

שלב 7: יישום המסנן HAVING

מסנן HAVING מיושם על הקבוצות בטבלה המוחזרת מהשלב הקודם. רק קבוצות שעבורן תנאי having (<having_condition>) הוא TRUE הופכות לחלק מהטבלה הווירטואלית המוחזרת על ידי שלב זה (VT7). מסנן HAVING הוא המסנן הראשון והיחיד המתייחס לנתונים המקובצים.

יישם שלב זה על השאילתה לדוגמה:

```
HAVING COUNT(O.orderid) < 3
```

הקבוצה של לקוח KRLOS מוסרת מכיוון שהיא מכילה שלוש הזמנות. נוצרת טבלה וירטואלית VT7, המוצגת בטבלה 10-1.

טבלה 10-1: טבלה וירטואלית VT7 המוחזרת על ידי שלב 7

C.customerid	*C.customerid*	*C.city*	*O.orderid*	*O.customerid*
FRNDO	FRNDO	Madrid	1	FRNDO
	FRNDO	Madrid	2	FRNDO
FISSA	FISSA	Madrid	NULL	NULL

שים לב: חשוב לציין כאן COUNT(O.orderid) ולא COUNT(*). מכיוון שה–join הוא outer join, שורות חיצוניות התווספו עבור לקוחות ללא הזמנות. COUNT(*) היה מצרף שורות חיצוניות לספירה תוך כדי הוספה שגויה של הזמנה אחת ללקוח FISSA, לו אין הזמנות. COUNT(O.orderid) סופר בצורה נכונה את מספר ההזמנות לכל לקוח, ומחשב 0 הזמנות ללקוח FISSA. זכור ש– COUNT(<expression>) מתעלם מ–NULLs כמו כל פונקציית צבירה אחרת.

פונקציית צבירה אינה מקבלת תת–שאילתה כקלט - למשל,

HAVING SUM((SELECT …)) > 10

שלב 8: עיבוד רשימת ה–SELECT

על אף שרשימת ה–SELECT מצוינת ראשונה בשאילתה, היא מעובדת רק בשלב השמיני. שלב ה–SELECT בונה את הטבלה אשר בסופו של דבר תוחזר ללקוח. הביטויים ברשימת ה–SELECT יכולים להחזיר טורי בסיס ומניפולציות של טורי בסיס מהטבלה הווירטואלית המוחזרת מהשלב הקודם. זכור שאם השאילתה היא שאילתת צבירה, לאחר שלב 5 תוכל להתייחס לטורי בסיס מהשלב הקודם אך ורק אם הם חלק מיחידת הקבוצות (רשימת GROUP BY). אם תרצה להתייחס לטורים מיחידת המידע הגולמי, אלו חייבים להופיע בתוך פונקציית הצבירה. טורי בסיס הנבחרים מהשלב הקודם שומרים על שמות הטורים שלהם אלא אם כן תיתן להם כינויים (למשל, col1 AS c1). ביטויים שאינם טורי בסיס חייבים לקבל כינוי כדי שיהיה להם שם טור בטבלת התוצאה - למשל, YEAR(orderdate) AS orderyear.

חשוב: לא ניתן להשתמש בכינויים הנוצרים על ידי רשימת ה–SELECT בשלבים קודמים. למעשה, בכינויי ביטויים לא ניתן אפילו להשתמש על ידי ביטויים אחרים בתוך אותה רשימת SELECT. ההיגיון מאחורי מגבלה זו הוא עוד פן ייחודי של SQL, היותו פעולה בו–זמנית. למשל, ברשימת ה–SELECT הבאה, הסדר הלוגי לפיו הביטויים מוערכים אינו חשוב ואינו מובטח: SELECT c1 + 1 AS e1, c2 + 1 AS e2. לפיכך, רשימת ה–SELECT הבאה אינה נתמכת: SELECT c1 + 1 AS e1, e1 + 1 AS e2. מותר לך להשתמש שוב בכינויי טורים רק בשלבים הבאים אחרי רשימת ה–SELECT, כמו שלב ה– ORDER BY – למשל, SELECT YEAR(orderdate) AS orderyear … ORDER BY orderyear.

יישם שלב זה על השאילתה לדוגמה:

```
SELECT C.customerid, COUNT(O.orderid) AS numorders
```

מתקבלת טבלה וירטואלית VT8, המוצגת בטבלה 1‏–11.

טבלה 1‏–11: טבלה וירטואלית VT8 המוחזרת משלב 8

C.customerid	*numorders*
FRNDO	2
FISSA	0

המושג פעולה בו–זמנית עשוי להיות קשה לתפיסה. למשל, במרבית סביבות הפיתוח, לצורך החלפת ערכים בין משתנים תשתמש במשתנה זמני. לעומת זאת, ב–SQL כדי להחליף ערכים בין טורים בטבלה תוכל להשתמש ב:

```
UPDATE dbo.T1 SET c1 = c2, c2 = c1;
```

לוגית, עליך להניח שכל הפעולה מבוצעת בו–זמנית. כאילו הטבלה אינה מעודכנת עד שכל הפעולה מסתיימת ואז התוצאה מחליפה את המקור. מסיבות דומות, UPDATE זה:

```
UPDATE dbo.T1 SET c1 = c1 + (SELECT MAX(c1) FROM dbo.T1);
```

יעדכן את כל השורות של T1 ויוסיף ל–c1 את הערך המקסימלי מ–T1 כאשר העדכון מתחיל. אינך צריך לדאוג שמא הערך המקסימלי של c1 ימשיך להשתנות כאשר הפעולה תמשיך, מכיוון שהפעולה כאילו מתרחשת בו–זמנית.

שלב 9: יישום הפסוקית DISTINCT

אם בשאילתה מצוינת פסוקית DISTINCT, שורות כפולות מוסרות מהטבלה הווירטואלית המוחזרת על ידי השלב הקודם, וטבלה וירטואלית VT9 נוצרת.

בדוגמה שלנו נדלג על שלב 9 מכיוון שבשאילתה לדוגמה לא מופיע DISTINCT. למעשה, DISTINCT היא פסוקית מיותרת כאשר משתמשים ב– GROUP BY והיא לא תסיר אף שורה.

שלב 10: יישום הפסוקית ORDER BY

השורות מהשלב הקודם ממוינות לפי רשימת הטורים המוגדרת בפסוקית ORDER BY ומוחזרות בסמן (cursor) VC10. שלב זה הוא הראשון והיחיד בו ניתן לשוב ולהשתמש בכינויי טורים שנוצרו ברשימת ה–SELECT.

הן לפי ANSI SQL:1992 והן לפי ANSI SQL:1999, אם מצוין DISTINCT, לביטויים בפסוקית ORDER BY יש גישה רק לטבלה הווירטואלית המוחזרת על ידי השלב הקודם

(VT9). כלומר, ניתן למיין רק לפי מה שבחרת. ANSI SQL:1992 מציב את אותה מגבלה אפילו כאשר DISTINCT אינו מצוין. לעומת זאת, ANSI SQL:1999 מרחיב את התמיכה ב– ORDER BY על ידי כך שהוא מאפשר גישה הן לטבלת הקלט הווירטואלית של שלב ה–SELECT והן לטבלת הפלט הווירטואלית של השלב הנוכחי. כלומר, אם DISTINCT אינו מופיע, ניתן לציין בפסוקית ORDER BY כל ביטוי שהיה מותר בפסוקית SELECT. בפרט, ניתן למיין לפי ביטויים שאינם מוחזרים בתוצאה הסופית.

ישנו היגיון באיסור גישה לביטויים שאינך מחזיר כאשר DISTINCT מצוין. כאשר מוסיפים ביטויים לרשימת ה–SELECT, קיימת אפשרות ש–DISTINCT ישנה את מספר השורות המוחזרות. ללא DISTINCT, כמובן ששינויים ברשימת ה–SELECT לא משפיעים על מספר השורות המוחזרות. T-SQL תמיד יישם את הגישה של ANSI SQL:1999.

בדוגמה שלנו, מכיוון ש–DISTINCT אינו מצוין, לפסוקית ORDER BY יש גישה הן לטבלה VT7, המוצגת בטבלה 1–10, והן לטבלה VT8, המוצגת בטבלה 1–11.

בפסוקית ORDER BY ניתן לציין גם מיקום סידורי של טורי תוצאה מרשימת ה–SELECT. לדוגמה, השאילתה הבאה ממיינת את ההזמנות ראשית לפי customerid ואז לפי orderid:

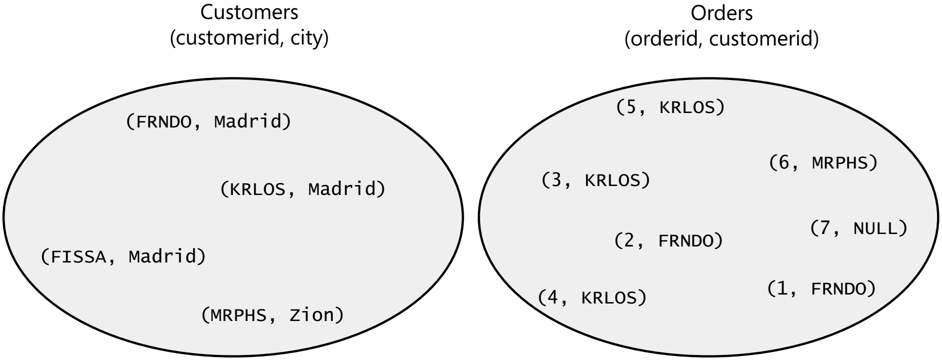
```
SELECT orderid, customerid FROM dbo.Orders ORDER BY 2, 1;
```

אף על פי כן, נוהג כזה אינו מומלץ מכיוון שאתה עלול לבצע שינויים ברשימת ה–SELECT ולשכוח לעדכן את רשימת ה– ORDER BY בהתאם. כמו כן, כאשר מחרוזת השאילתה ארוכה, קשה להבחין איזה פריט מרשימת ה– ORDER BY מתייחס לאיזה פריט מרשימת ה–SELECT.

חשוב: שלב זה שונה מכל השלבים האחרים במובן זה שאינו מחזיר טבלה; במקום זאת, הוא מחזיר סמן. זכור ש–SQL מבוסס על תורת הקבוצות (set theory). לשורות בסט אין סדר קבוע מראש; זהו אוסף לוגי של איברים, ללא כל חשיבות לסדר ביניהם. שאילתה המיישמת מיון לשורות בטבלה מחזירה אובייקט המכיל שורות מסודרות בסדר פיסי מסוים. ANSI קוראת לאובייקט כזה **cursor (סמן)**. הבנת שלב זה הינה אחד הדברים העקרוניים ביותר בהבנה נכונה של SQL.

בדרך כלל, כאשר מתארים תוכן של טבלה, מרבית האנשים (ואני ביניהם) משרטטים מבלי משים את השורות בסדר מסוים. למשל, מוקדם יותר הצגתי את טבלאות 1–1 ו–1–2 כדי לתאר את התוכן של טבלאות Customers ו–Orders. על ידי הצגת השורות בזו אחר זו, גרמתי, מבלי להתכוון, לבלבול–מה, בכך שרמזתי על סדר מסוים. דרך נכונה יותר לשרטט את התוכן של טבלאות Customers ו–Orders תהיה זו המוצגת בתרשים 1–1.

תרשים 1-1: סטים Customers ו-Orders



שים לב: אף על פי ששפת SQL לא מניחה שום סדר נתון לשורות בטבלה, היא כן מחזיקה מיקום סידורי עבור טורים, המבוסס על סדר היצירה שלהם. ציון SELECT * (על אף שזהו נוהג שאינו מומלץ ממספר סיבות שאתאר בהמשך הספר) מבטיח שהטורים יוחזרו לפי סדר יצירתם.

מכיוון ששלב זה אינו מחזיר טבלה (הוא מחזיר סמן), שאילתה עם פסוקית ORDER BY אינה יכולה לשמש כביטוי טבלה – כלומר, view, inline table-valued function, תת-שאילתה, טבלה נגזרת, או ביטוי טבלה שגור (CTE). התוצאה צריכה להיות מוחזרת ליישום הלקוח המצפה לקבל בחזרה קבוצת רשומות פיסית. למשל, שאילתת הטבלה הנגזרת הבאה אינה חוקית ומייצרת שגיאה:

```
SELECT *
FROM (SELECT orderid, customerid
      FROM dbo.Orders
      ORDER BY orderid) AS D;
```

בדומה, ה-view הבא אינו חוקי:

```
CREATE VIEW dbo.VSortedOrders
AS

SELECT orderid, customerid
FROM dbo.Orders
ORDER BY orderid
GO
```

ב-SQL שאילתה עם פסוקית ORDER BY אינה מותרת בביטוי טבלה. ב-T-SQL יש יוצא דופן לכלל זה המתואר בשלב הבא – יישום אפשרות TOP.

אם כך זכור, אל תניח שום סדר נתון לשורות בטבלה. והפוך, אל תציין פסוקית ORDER BY אלא אם כן אתה באמת צריך את השורות ממוינות. למיון יש מחיר – על SQL Server לבצע סריקה ממוינת של אינדקס או לחליפין ליישם אופרטור של מיון.

שלב ה– ORDER BY מתייחס ל–NULLs כשווים. כלומר, NULLs ממוינים יחד. ANSI משאירה את השאלה האם NULLs ממוינים לפני או אחרי ערכים ידועים, למוצר המממש, אשר חייב להיות עקבי. T-SQL ממיין NULLs לפני ערכים ידועים.

יישם שלב זה על השאילתה לדוגמה:

```
ORDER BY numorders
```

מתקבל הסמן VC10 המוצג בטבלה 12–1.

טבלה 12–1: סמן VC10 מוחזר משלב 10

C.customerid	*numorders*
FISSA	0
FRNDO	2

שלב 11: יישום האפשרות TOP

אפשרות TOP מאפשרת לך לציין מספר או אחוז של שורות (מעוגל כלפי מעלה) שיוחזרו. ב– SQL Server 2000 הקלט ל–TOP חייב להיות קבוע, בעוד שב– SQL Server 2005 הקלט יכול להיות כל ביטוי עצמאי. מספר השורות שהוגדר נבחר מתחילת הסמן שהוחזר על ידי השלב הקודם. טבלה VT11 נוצרת ומוחזרת ללקוח.

שים לב: האפשרות TOP היא ספציפית ל–T-SQL ואינה רלציונית.

שלב זה נשען על הסדר הפיסי של השורות כדי לקבוע אילו שורות נחשבות למספר השורות ה"ראשונות" המבוקש. אם בשאילתה מצוינת פסוקית ORDER BY עם רשימת ORDER BY ייחודית, התוצאה היא דטרמיניסטית. כלומר, קיימת רק תוצאה נכונה אחת, המכילה את מספר השורות הראשונות המבוקש בהתבסס על המיון שהוגדר. בדומה, כאשר מצוינת פסוקית ORDER BY עם רשימת ORDER BY שאינה ייחודית אך עם אפשרות TOP מצוין WITH TIES, התוצאה דטרמיניסטית גם כן. SQL Server בוחן את השורה האחרונה שהוחזרה פיסית ומחזיר את כל השורות האחרות מהטבלה שלהן ערך מיון זהה.

לעומת זאת, כאשר מצוינת רשימת ORDER BY שאינה ייחודית, ללא אפשרות WITH TIES, או ש– ORDER BY אינו מצוין כלל, שאילתת TOP אינה דטרמיניסטית. כלומר, השורות המוחזרות הן אלו שה– SQL Server במקרה ניגש אליהן ראשונות, ועשויות להיות תוצאות שונות שייחשבו נכונות.

אם ברצונך להבטיח דטרמיניסטיות, שאילתת TOP חייבת לכלול או רשימת ORDER BY ייחודית, או את האפשרות WITH TIES.

כפי שתוכל לשער, שאילתות TOP מכילות לרוב פסוקית ORDER BY הקובעת אילו שורות להחזיר. SQL Server מאפשר לך לציין שאילתות TOP בביטויי טבלה. אין הרבה היגיון באפשרות לציין שאילתות TOP בביטויי טבלה ללא מתן אפשרות לציין גם פסוקית ORDER BY (ראה מגבלה בשלב 10). לפיכך, שאילתות עם פסוקית ORDER BY מותרות למעשה בביטויי טבלה אך ורק אם מצוין גם TOP. במילים אחרות, שאילתה המכילה הן פסוקית TOP והן פסוקית ORDER BY מחזירה תוצאה רלציונית. האירוניה כאן היא שעל ידי שימוש באפשרות TOP שאינה תקנית ואינה רלציונית, שאילתה שאחרת הייתה מחזירה סמן, מחזירה כעת תוצאה רלציונית. תמיכה במאפיינים שאינם תקניים ואינם רלציונים (מעשיים ככל שיהיו) מאפשרת למתכנתים לנצל אותם לרעה, בדרכים אבסורדיות לעיתים, שלא היו נתמכות אחרת.

להלן דוגמה:

```
SELECT *
FROM (SELECT TOP 100 PERCENT orderid, customerid
      FROM dbo.Orders
      ORDER BY orderid) AS D;
```

או:

```
CREATE VIEW dbo.VSortedOrders
AS

SELECT TOP 100 PERCENT orderid, customerid
FROM dbo.Orders
ORDER BY orderid
GO
```

בדוגמה שלנו נדלג על שלב 11 שכן TOP אינו מופיע בשאילתה.

שלבים חדשים בעיבוד לוגי של שאילתה ב– SQL Server 2005

סעיף זה מתמקד בשלבי העיבוד המעורבים באלמנטים החדשים של שאילתות ב– SQL Server 2005. אלו כוללים אופרטורים טבלאיים (APPLY, PIVOT ו–UNPIVOT), פסוקית OVER החדשה, ופעולות סט חדשות (EXCEPT ו–INTERSECT).

שים לב: האופרטורים APPLY, PIVOT ו–UNPIVOT אינם תואמי ANSI אלא הם הרחבות ספציפיות ל–T-SQL.

אני מוצא שזה בעייתי מעט לכסות בפירוט בפרק הראשון את שלבי העיבוד הלוגי המעורבים בגרסת המוצר החדשה. אלמנטים אלו חדשים לגמרי, ויש כל–כך הרבה לומר על כל אחד מהם. במקום זאת, אביא כאן סקירה קצרה של כל אלמנט ואנהל דיונים מפורטים מאוחר יותר בספר בפרקים ממוקדים.

כפי שציינתי קודם לכן, המטרה שלי בפרק זה היא לתת לך תקציר אליו תוכל לחזור מאוחר יותר כאשר תתלבט בנוגע להיבטים הלוגיים של אלמנטים של שאילתה והדרך בה הם פועלים הדדית. עם מחשבה זו ברקע, המשמעות המלאה של שלבי העיבוד הלוגי של שאילתה, המטפלים באלמנטים החדשים עשויה להיות לא לגמרי ברורה לך כעת. אל דאגה, לאחר קריאת הפרקים הדנים בכל אלמנט בפירוט, סביר שתמצא את התקציר שאני מביא בפרק זה שימושי.

אופרטורים טבלאיים

SQL Server 2005 תומך בארבעה סוגים של אופרטורים טבלאיים בפסוקית FROM בשאילתה: JOIN, APPLY, PIVOT ו–UNPIVOT.

את שלבי העיבוד הלוגיים המעורבים ב–joins כיסיתי מוקדם יותר; כמו כן אמשיך ואדון ב–joins ביתר פירוט גם בפרק 5. כאן אתאר בקצרה את שלושת האופרטורים החדשים וכיצד הם פועלים הדדית.

אופרטורים טבלאיים מקבלים כקלטים טבלה אחת או שתיים. נקרא להם **קלט שמאל** ו**קלט ימין** בהתבסס על המיקום שלהן ביחס למילה השמורה של האופרטור הטבלאי (JOIN, APPLY, PIVOT, UNPIVOT). ממש כמו joins, כל האופרטורים הטבלאיים, מקבלים טבלה וירטואלית כקלט שמאל שלהם. האופרטור הטבלאי הראשון שמופיע בפסוקית FROM מקבל ביטוי טבלה כקלט שמאל ומחזיר טבלה וירטואלית כתוצאה. ביטוי טבלה יכול להחליף מספר דברים: טבלה אמיתית, טבלה זמנית, משתנה טבלה, טבלה נגזרת, CTE, view, או פונקציה טבלאית.

מידע נוסף: לפרטים אודות ביטויי טבלה, אנא עיין בפרק 4.

האופרטור הטבלאי השני המופיע בפסוקית FROM מקבל את הטבלה הווירטואלית המוחזרת מפעולת הטבלה הקודמת כקלט שמאל.

כל אופרטור טבלאי מאגד בתוכו סדרה שונה של צעדים. לצורך הנוחות וההירות אוסיף למספרי הצעדים תחילית עם ראשי התיבות של האופרטור הטבלאי (J עבור JOIN, A עבור APPLY, P עבור PIVOT ו–U עבור UNPIVOT).

להלן ארבעת האופרטורים הטבלאיים בצירוף האלמנטים שלהם:

```
(J) <left_table_expression>
      <join_type> JOIN <right_table_expression>
       ON <join_condition>

(A) <left_table_expression>
      {CROSS | OUTER} APPLY <right_table_expression>

(P) <left_table_expression>
      PIVOT (<aggregate_func(<expression>)> FOR
        <source_col> IN(<target_col_list>))
        AS <result_table_alias>

(U) <left_table_expression>
      UNPIVOT (<target_values_col> FOR
        <target_names_col> IN(<source_col_list>))
        AS <result_table_alias>
```

כתזכורת, join משלב תת-קבוצה (תלוי בסוג ה-join) של הצעדים הבאים:

1. J1: בצע מכפלה קרטזית בין קלטים שמאל וימין.
2. J2: יישם פסוקית ON.
3. J3: הוסף שורות חיצוניות.

APPLY

אופרטור APPLY משלב תת-קבוצה (תלוי בסוג ה-apply) של שני הצעדים הבאים:

1. A1: יישם ביטוי טבלה ימנית עבור כל שורה מהטבלה השמאלית.
2. A2: הוסף שורות חיצוניות.

אופרטור APPLY למעשה מחיל את ביטוי הטבלה הימנית על כל שורה מקלט שמאל. תוכל לחשוב על כך כדבר דומה ל-join, בהבדל אחד חשוב – ביטוי הטבלה הימנית יכול להתייחס לטורי קלט שמאל כמתאם (קורלציה). כאילו שב-join אין קדימות לאף אחד מהקלטים כאשר הם מוערכים. עבור APPLY, לוגית נוח לדמיין שקלט שמאל מוערך ראשון, ואז קלט ימין מוערך פעם אחת לכל שורה מקלט משמאל.

צעד A1 מיושם הן ב- CROSS APPLY והן ב- OUTER APPLY. צעד A2 מיושם אך ורק עבור OUTER APPLY. CROSS APPLY אינו מחזיר שורה חיצונית (שמאלית) אם ביטוי הטבלה הפנימי (ימני) מחזיר סט ריק עבורה. OUTER APPLY יחזיר שורה כזו, עם NULLs במאפייני ביטויי הטבלה הפנימיים.

למשל, השאילתה הבאה מחזירה את שתי ההזמנות האחרונות (בהנחה, לצורך דוגמה זו, ש–orderid מייצג סדר כרונולוגי) עבור כל לקוח, ומייצרת את הפלט המוצג בטבלה 13–1:

```
SELECT C.customerid, city, orderid
FROM dbo.Customers AS C
  CROSS APPLY
    (SELECT TOP(2) orderid, customerid
     FROM dbo.Orders AS O
     WHERE O.customerid = C.customerid
     ORDER BY orderid DESC) AS CA;
```

טבלה 13–1: שתי הזמנות אחרונות של כל לקוח

customerid	*city*	*orderid*
FRNDO	Madrid	2
FRNDO	Madrid	1
KRLOS	Madrid	5
KRLOS	Madrid	4
MRPHS	Zion	6

שים לב שלקוח FISSA אינו מופיע בפלט מכיוון שביטוי הטבלה CA החזיר קבוצה ריקה עבור שורת הלקוח. אם ברצונך להחזיר גם לקוחות שלא ביצעו הזמנות, עליך להשתמש ב– OUTER APPLY כלהלן, ותקבל את הפלט המוצג בטבלה 14–1:

```
SELECT C.customerid, city, orderid
FROM dbo.Customers AS C
  OUTER APPLY
    (SELECT TOP(2) orderid, customerid
     FROM dbo.Orders AS O
     WHERE O.customerid = C.customerid
     ORDER BY orderid DESC) AS OA;
```

טבלה 14–1: שתי הזמנות אחרונות לכל לקוח, כולל לקוחות שלא ביצעו הזמנות

customerid	*city*	*orderid*
FISSA	Madrid	NULL
FRNDO	Madrid	2

customerid	*city*	*orderid*
FRNDO	Madrid	1
KRLOS	Madrid	5
KRLOS	Madrid	4
MRPHS	Zion	6

מידע נוסף: לפרטים נוספים על אופרטור APPLY, אנא עיין בפרק 7.

PIVOT

האופרטור PIVOT מאפשר לך בעיקרון לסובב על ציר (pivot) נתונים, ממצב של קבוצות מרובות שורות, למצב של ריבוי טורים בשורה יחידה עבור כל קבוצה, תוך כדי ביצוע חישובי צבירה כחלק מהתהליך.

בטרם אסביר ואדגים את הצעדים הלוגיים המעורבים בשימוש באופרטור PIVOT, בחן את השאילתה הבאה, בה אשתמש מאוחר יותר כקלט שמאל עבור האופרטור PIVOT:

```
SELECT C.customerid, city,
  CASE
    WHEN COUNT(orderid)  = 0 THEN 'no_orders'
    WHEN COUNT(orderid) <= 2 THEN 'upto_two_orders'
    WHEN COUNT(orderid)  > 2 THEN 'more_than_two_orders'
  END AS category
FROM dbo.Customers AS C
  LEFT OUTER JOIN dbo.Orders AS O
    ON C.customerid = O.customerid
GROUP BY C.customerid, city;
```

שאילתה זו מחזירה קטגוריות של לקוח המבוססות על כמות ההזמנות (ללא הזמנות, עד שתי הזמנות, למעלה משתי הזמנות), ומפיקה את התוצאה המוצגת בטבלה 15–1.

טבלה 15–1: קטגוריות לקוח בהתאם לכמות הזמנות

customerid	*city*	*category*
FISSA	Madrid	no_orders
FRNDO	Madrid	upto_two_orders
KRLOS	Madrid	more_than_two_orders
MRPHS	Zion	upto_two_orders

נניח שברצונך לדעת את מספר הלקוחות בכל קטגוריה בכל עיר. שאילתת ה–PIVOT הבאה מאפשרת לך לקבל מידע זה, ומפיקה את הפלט המופיע בטבלה 1–16:

```
SELECT city, no_orders, upto_two_orders, more_than_two_orders
FROM (SELECT C.customerid, city,
        CASE
          WHEN COUNT(orderid)  = 0 THEN 'no_orders'
          WHEN COUNT(orderid) <= 2 THEN 'upto_two_orders'
          WHEN COUNT(orderid)  > 2 THEN 'more_than_two_orders'
        END AS category
      FROM dbo.Customers AS C
        LEFT OUTER JOIN dbo.Orders AS O
          ON C.customerid = O.customerid
      GROUP BY C.customerid, city) AS D
  PIVOT(COUNT(customerid) FOR
     category IN([no_orders],
                 [upto_two_orders],
                 [more_than_two_orders])) AS P;
```

טבלה 1–16: מספר לקוחות בכל קטגוריה בכל עיר

city	*no_orders*	*upto_two_orders*	*more_than_two_orders*
Madrid	1	1	1
Zion	0	1	0

אל תיתן לשאילתה המייצרת את הטבלה הנגזרת D לבלבל אותך. ככל הנוגע לך, האופרטור PIVOT מקבל כקלט שמאל שלו ביטוי טבלה שנקרא D, המכיל את קטגוריות הלקוח.

אופרטור PIVOT כולל את שלושת השלבים הלוגיים להלן:

1. P1: קיבוץ מרומז.
2. P2: בידוד ערכים.
3. P3: יישום פונקציית הצבירה.

השלב הראשון (P1) מאוד קשה לתפיסה. ניתן לראות בשאילתה שאופרטור PIVOT מתייחס לשניים מהטורים ב–D כקלטים (customerid ו–category). השלב הראשון מקבץ בעקיפין את השורות מ–D בהתבסס על כל הטורים שלא הוזכרו בקלטים של PIVOT, כאילו היה חבוי שם GROUP BY. במקרה שלנו, רק טור העיר לא הוזכר כאף אחד מהקלטים של PIVOT. כך מתקבלת קבוצה לכל עיר (Madrid ו–Zion במקרה שלנו).

שים לב: שלב הקיבוץ המרומז של PIVOT אינו מחליף פסוקית GROUP BY מפורשת, במידה שכזו מופיעה בשאילתה. PIVOT בסופו של דבר יפיק טבלת תוצאה וירטואלית, שבתורה תהווה קלט לשלב הלוגי הבא, בין אם תהיה זו פעולה טבלאית אחרת או שלב ה–WHERE. כפי שתיארתי מוקדם יותר בפרק, לאחר שלב WHERE עשוי לבוא שלב GROUP BY. כך שכאשר הן PIVOT והן GROUP BY מופיעים בשאילתה, מתקבלים שני שלבי קיבוץ נפרדים – אחד כשלב הראשון של PIVOT (P1) ואחד מאוחר יותר כשלב GROUP BY של השאילתה.

השלב השני של PIVOT (P2) מבודד ערכים מקבילים לטורי יעד. לוגית, הוא משתמש בביטוי CASE הבא עבור כל טור יעד המצוין בפסוקית IN:

```
CASE WHEN <source_col> = <target_col_element> THEN <expression> END
```

במצב זה, מיושמים לוגית שלושת הביטויים הבאים:

```
CASE WHEN category = 'no_orders'            THEN customerid END,
CASE WHEN category = 'upto_two_orders'      THEN customerid END,
CASE WHEN category = 'more_than_two_orders' THEN customerid END
```

שים לב: לביטוי CASE ללא פסוקית ELSE קיים ELSE NULL מרומז.

ביטוי CASE יחזיר את קוד הלקוח לכל טור יעד, רק אם לשורת המקור הייתה את הקטגוריה המקבילה; אחרת CASE יחזיר NULL.

השלב השלישי של PIVOT (P3) מיישם את פונקציית הצבירה המוגדרת על כל ביטוי CASE, ומייצר את טורי התוצאה. במקרה שלנו, הביטויים הופכים לוגית ל–

```
COUNT(CASE WHEN category = 'no_orders'
             THEN customerid END) AS [no_orders],
COUNT(CASE WHEN category = 'upto_two_orders'
             THEN customerid END) AS [upto_two_orders],
COUNT(CASE WHEN category = 'more_than_two_orders'
             THEN customerid END) AS [more_than_two_orders]
```

לסיכום, שאילתת ה–PIVOT הקודמת זהה לוגית לשאילתה הבאה:

```
SELECT city,
  COUNT(CASE WHEN category = 'no_orders'
             THEN customerid END) AS [no_orders],
  COUNT(CASE WHEN category = 'upto_two_orders'
             THEN customerid END) AS [upto_two_orders],
  COUNT(CASE WHEN category = 'more_than_two_orders'
             THEN customerid END) AS [more_than_two_orders]
FROM (SELECT C.customerid, city,
        CASE
          WHEN COUNT(orderid)  = 0 THEN 'no_orders'
          WHEN COUNT(orderid) <= 2 THEN 'upto_two_orders'
          WHEN COUNT(orderid)  > 2 THEN 'more_than_two_orders'
        END AS category
      FROM dbo.Customers AS C
        LEFT OUTER JOIN dbo.Orders AS O
          ON C.customerid = O.customerid
      GROUP BY C.customerid, city) AS D
GROUP BY city;
```

מידע נוסף: לפרטים נוספים על האופרטור PIVOT, אנא עיין בפרק 6.

UNPIVOT

UNPIVOT היא הפעולה ההפוכה ל–PIVOT, סיבוב נתונים על ציר ממצב של ערכי טבלה מרובים באותה שורה לשורות מרובות, כל אחת עם ערך טור מקור שונה.

לפני שאדגים את השלבים הלוגיים של UNPIVOT, ראשית הרץ את הקוד בקטע–קוד 4–1, ליצירה ומילוי של הטבלה PivotedCategories.

קטע–קוד 4–1: יצירה ומילוי של הטבלה PivotedCategories

```
SELECT city, no_orders, upto_two_orders, more_than_two_orders
INTO dbo.PivotedCategories
FROM (SELECT C.customerid, city,
        CASE
          WHEN COUNT(orderid)  =  0 THEN 'no_orders'
          WHEN COUNT(orderid) <=  2 THEN 'upto_two_orders'
          WHEN COUNT(orderid)  >  2 THEN 'more_than_two_orders'
        END AS category
      FROM dbo.Customers AS C
```

```
        LEFT OUTER JOIN dbo.Orders AS O
          ON C.customerid = O.customerid
      GROUP BY C.customerid, city) AS D
  PIVOT(COUNT(customerid) FOR
    category IN([no_orders],
                [upto_two_orders],
                [more_than_two_orders])) AS P;

UPDATE dbo.PivotedCategories
  SET no_orders = NULL, upto_two_orders = 3
WHERE city = 'Madrid';
```

לאחר שתריץ את הקוד בקטע-קוד 1-4, הטבלה PivotedCategories תכיל את הנתונים המוצגים בטבלה 1-17.

טבלה 1-17: תוכן של טבלת PivotedCategories

city	*no_orders*	*upto_two_orders*	*more_than_two_orders*
Madrid	NULL	3	1
Zion	0	1	0

אשתמש בשאילתה הבאה כדוגמה כדי לתאר את שלבי העיבוד הלוגי המעורבים באופרטור UNPIVOT:

```
SELECT city, category, num_custs
FROM dbo.PivotedCategories
  UNPIVOT(num_custs FOR
    category IN([no_orders],
                [upto_two_orders],
                [more_than_two_orders])) AS U
```

שאילתה זו מבצעת unpivot (או מחלקת) את קטגוריות הלקוח מכל שורת מקור לשורה נפרדת לכל קטגוריה, ומפיקה את הפלט המוצג בטבלה 1-18.

טבלה 1-18: קטגוריות לקוח לאחר unpivot

city	*category*	*num_custs*
Madrid	upto_two_orders	3

city	*category*	*num_custs*
Madrid	more_than_two_orders	1
Zion	no_orders	0
Zion	upto_two_orders	1
Zion	more_than_two_orders	0

שלושת שלבי העיבוד הלוגי להלן מעורבים בפעולת UNPIVOT:

1. **U1**: שכפול שורות.
2. **U2**: בידוד ערכי טורי יעד.
3. **U3**: סינון החוצה של שורות עם NULL.

הצעד הראשון (U1) משכפל שורות מביטוי הטבלה השמאלי המסופק ל–UNPIVOT כקלט (במקרה שלנו, PivotedCategories). כל שורה משוכפלת פעם אחת לכל טור מקור המופיע בפסוקית IN. מכיוון שישנם שלושה שמות טורים בפסוקית IN, כל שורת מקור תשוכפל שלוש פעמים. טבלת התוצאה הווירטואלית תכלול טור חדש המכיל את שמות טורי המקור כמחרוזות תווים. טור זה יקבל את השם המוגדר מייד לפני פסוקית IN (במקרה שלנו, קטגוריה). הטבלה הווירטואלית המוחזרת מהשלב הראשון בדוגמה שלנו מוצגת בטבלה 19–1.

טבלה 19–1: טבלה וירטואלית המוחזרת מהשלב הראשון של UNPIVOT

city	*no_orders*	*upto_two_orders*	*more_than_two_orders*	*category*
Madrid	NULL	3	1	no_orders
Madrid	NULL	3	1	upto_two_orders
Madrid	NULL	3	1	more_than_two_orders
Zion	0	1	0	no_orders
Zion	0	1	0	upto_two_orders
Zion	0	1	0	more_than_two_orders

השלב השני (U2) מבודד את ערכי טורי היעד. טור היעד שיכיל את הערכים, יקבל את השם המוגדר מייד לפני פסוקית FOR (במקרה שלנו, num_custs). שם טור היעד יכיל את הערך מהטור המקביל לקטגוריה של השורה הנוכחית מהטבלה הווירטואלית. הטבלה הווירטואלית המוחזרת מצעד זה בדוגמה שלנו מוצגת בטבלה 20–1.

טבלה 1-20: טבלה וירטואלית המוחזרת מהשלב השני של UNPIVOT

city	*category*	*num_custs*
Madrid	no_orders	NULL
Madrid	upto_two_orders	3
Madrid	more_than_two_orders	1
Zion	no_orders	0
Zion	upto_two_orders	1
Zion	more_than_two_orders	0

השלב השלישי של UNPIVOT (U3) מסנן החוצה שורות עם NULL בטור ערך התוצאה (במקרה שלנו, num_custs). הטבלה הווירטואלית המוחזרת מצעד זה בדוגמה שלנו מוצגת בטבלה 1–21.

טבלה 1–21: טבלה וירטואלית מוחזרת מהשלב השלישי של UNPIVOT

city	*category*	*num_custs*
Madrid	upto_two_orders	3
Madrid	more_than_two_orders	1
Zion	no_orders	0
Zion	upto_two_orders	1
Zion	more_than_two_orders	0

כשתסיים להתנסות עם אופרטור UNPIVOT, הסר את טבלה PivotedCategories:

```
DROP TABLE dbo.PivotedCategories;
```

מידע נוסף: לפרטים נוספים על האופרטור UNPIVOT, אנא עיין בפרק 6.

הפסוקית OVER

פסוקית OVER מאפשרת לך לבקש חישובים מבוססי–חלון. ב– SQL Server 2005, פסוקית זו היא אפשרות חדשה לפונקציות צבירה (הן פונקציות צבירה מובנות והן פונקציות צבירה מבוססות CLR [Common Language Runtime]) וזהו אלמנט חובה עבור ארבע פונקציות הדירוג החדשות (ROW_NUMBER, RANK, DENSE_RANK ו–NTILE). כאשר מוגדרת פסוקית OVER, הקלט שלה, במקום רשימת GROUP BY של השאילתה, מציין את חלון השורות עליהן מחושבת פונקציית הצבירה או הדירוג.

בשלב זה לא אדון ביישומים של חישובים מבוססי–חלון, כמו כן לא אתעמק בדרך המדויקת בה פונקציות אלו עובדות; אסביר רק את השלבים בהם פסוקית OVER ניתנת לשימוש. פסוקית OVER תכוסה בפירוט רב יותר בפרקים 4 ו–6.

פסוקית OVER ניתנת לשימוש רק באחד משני שלבים: שלב ה–SELECT (8) ושלב ה– ORDER BY (10). לפסוקית זו יש גישה לטבלה הווירטואלית המסופקת כקלט לשלב זה. קטע–קוד 1–5 מדגיש את שלבי העיבוד הלוגי בהם פסוקית OVER ניתנת לשימוש.

קטע–קוד 1–5: פסוקית OVER בעיבוד לוגי של שאילתה

```
(8)   SELECT (9)  DISTINCT (11)  TOP <select_list>
(1)   FROM <left_table>
(3)     <join_type> JOIN <right_table>
(2)        ON <join_condition>
(4)   WHERE <where_condition>
(5)   GROUP BY <group_by_list>
(6)   WITH {CUBE | ROLLUP}
(7)   HAVING <having_condition>
(10)  ORDER BY <order_by_list>
```

את פסוקית OVER הנך מגדיר לאחר הפונקציה עליה היא מיושמת, ב–select_list או ב–order_by_list.

על אף שלא הסברתי בפירוט כיצד פועלת פסוקית OVER, אדגים את השימוש בה בשני השלבים בהם היא ניתנת לשימוש. הדוגמה הבאה משתמשת בפסוקית OVER עם פונקציית הצבירה COUNT ברשימת ה–SELECT; הפלט של שאילתה זו מוצג בטבלה 1–22:

```
SELECT orderid, customerid,
  COUNT(*) OVER(PARTITION BY customerid) AS num_orders
FROM dbo.Orders
WHERE customerid IS NOT NULL
  AND orderid % 2 = 1;
```

טבלה 1–22: פסוקית OVER מיושם בשלב ה–SELECT

orderid	*customerid*	*num_orders*
1	FRNDO	1
3	KRLOS	2
5	KRLOS	2

פסוקית PARTITION BY מגדירה את החלון עבור החישוב. הפונקציה COUNT(*) סופרת את מספר השורות בטבלה הווירטואלית המסופקת כקלט לשלב ה–SELECT, בהן customerid שווה לזה בשורה הנוכחית. זכור שהטבלה הווירטואלית המסופקת כקלט לשלב ה–SELECT עברה כבר סינון WHERE – כלומר, קודי לקוח NULL ואפילו קודי הזמנה הוסרו כבר.

ניתן גם להשתמש בפסוקית OVER ברשימת ORDER BY. למשל, השאילתה הבאה ממיינת את השורות לפי המספר הסופי של שורות פלט ללקוח (בסדר יורד), ומייצרת את הפלט המופיע בטבלה 1–23:

```
SELECT orderid, customerid
FROM dbo.Orders
WHERE customerid IS NOT NULL
  AND orderid % 2 = 1
ORDER BY COUNT(*) OVER(PARTITION BY customerid) DESC;
```

טבלה 1–23: פסוקית OVER מיושמת בשלב ORDER BY

orderid	*customerid*
3	KRLOS
5	KRLOS
1	FRNDO

מידע נוסף: לפרטים על השימוש בפסוקית OVER עם פונקציות צבירה (אגרגציה), אנא עיין בפרק 6. לפרטים על השימוש בפסוקית OVER עם פונקציות דירוג, אנא עיין בפרק 4.

פעולות על קבוצות (Set Operations)

SQL Server 2005 תומך בשלוש פעולות על קבוצות: UNION, EXCEPT ו–INTERSECT. UNION היא היחידה הזמינה ב- SQL Server 2000. אופרטורים אלו של SQL מקבילים לאופרטורים המוגדרים בתיאוריה המתמטית של תורת הקבוצות. זהו התחביר עבור שאילתה המיישמת פעולת סט:

```
[(]left_query[)] {UNION [ALL] | EXCEPT | INTERSECT} [(]right_query[)]
[ORDER BY <order_by_list>]
```

פעולות סט משוות בין שורות שלמות משני קלטים. UNION מחזירה תוצאה אחת עם השורות משני הקלטים. אם אפשרות ALL אינה מצוינת, UNION מסירה מהתוצאה שורות כפולות. EXCEPT מחזירה שורות ייחודיות (distinct) המופיעות בקלט שמאל אך לא

בקלט ימין. INTERSECT מחזירה שורות ייחודיות המופיעות בשני הקלטים. ניתן לומר עוד דברים רבים בנוגע לפעולות סט אלו, אך כאן אבחר להתמקד בצעדי העיבוד הלוגי המעורבים בפעולת סט.

פסוקית ORDER BY אינה מותרת בשאילתות הקלט לפעולת הסט. מותר לך להגדיר פסוקית ORDER BY בסוף השאילתה, אך היא תיושם על התוצאה של פעולת הסט.

במושגים של עיבוד לוגי, כל שאילתת קלט מעובדת ראשית בנפרד עם כל השלבים הרלוונטיים לה. אז מיושמת פעולת הסט, ואם מצוינת פסוקית ORDER BY, היא מיושמת על התוצאה.

התבונן לדוגמה בשאילתה הבאה המייצרת את הפלט המופיע בטבלה 1–24:

```
SELECT 'O' AS letter, customerid, orderid FROM dbo.Orders
WHERE customerid LIKE '%O%'

UNION ALL

SELECT 'S' AS letter, customerid, orderid FROM dbo.Orders
WHERE customerid LIKE '%S%'

ORDER BY letter, customerid, orderid;
```

טבלה 1–24: תוצאה של פעולת הקבוצה UNION ALL

letter	*customerid*	*orderid*
O	FRNDO	1
O	FRNDO	2
O	KRLOS	3
O	KRLOS	4
O	KRLOS	5
S	KRLOS	3
S	KRLOS	4
S	KRLOS	5
S	MRPHS	6

ראשית, כל שאילתת קלט מעובדת בנפרד על כל שלבי העיבוד הלוגי הרלוונטיים. השאילתה הראשונה מחזירה טבלה עם הזמנות שבוצעו על ידי לקוחות המכילים את האות **O**. השאילתה השנייה מחזירה טבלה עם הזמנות שבוצעו על ידי לקוחות המכילים את האות **S**. פעולת הסט UNION ALL מצרפת את שני הסטים לאחד. לבסוף, פסוקית ORDER BY ממיינת את השורות לפי letter, customerid ו–orderid.

כדוגמה נוספת לשלבי עיבוד לוגי של פעולות סט, השאילתה הבאה מחזירה לקוחות שלא ביצעו הזמנות:

```
SELECT customerid FROM dbo.Customers
EXCEPT
SELECT customerid FROM dbo.Orders;
```

השאילתה הראשונה מחזירה את סט קודי הלקוח מטבלת Customers ({FISSA, FRNDO, KRLOS, MRPHS}), והשאילתה השנייה מחזירה את סט קודי הלקוח מטבלת Orders ({FRNDO, FRNDO, KRLOS, KRLOS, KRLOS, MRPHS, NULL}). פעולת הסט מחזירה ({FISSA}), שמייצג את השורות מהסט הראשון שלא מופיעות בסט השני. לבסוף, פעולת הסט מסירה שורות כפולות מהתוצאה. במקרה זה אין שורות כפולות להסרה.

שמות טורי התוצאה נקבעים על ידי קלט שמאל של פעולת הסט. טורים במיקומים מקבילים חייבים להיות תואמים בטיפוס הנתונים (datatype) שלהם או לעבור המרה בעקיפין. לסיום, היבט מעניין של פעולות סט הוא שהן מתייחסות ל-NULLs כשווים.

מידע נוסף: תוכל למצוא דיון מעמיק יותר בפעולות סט בפרק 5.

סיכום

הבנת שלבי עיבוד לוגי של שאילתה וההיבטים הייחודיים של SQL חיונית לפיתוח סגנון החשיבה המיוחד הנדרש לצורך תכנות בשפת SQL. על ידי היכרות עם היבטים אלו של השפה, תוכל לייצר פתרונות יעילים ולהסביר את הבחירות שאתה עושה. זכור, הרעיון הוא לשלוט בעקרונות ובטכניקות היסוד.

2

עיבוד פיסי של שאילתות

– מאת לובור קולר

בעוד שהפרק הקודם תיאר את **התוצאה** שהרצת שאילתה צריכה לייצר, פרק זה יסביר **כיצד** Microsoft SQL Server 2005 משיג תוצאה זו.

מרבית מומחי מסדי הנתונים משתמשים בשפת ה–SQL, וכל מוצרי מסדי הנתונים הרלציוניים מכילים ניב כלשהו של סטנדרט SQL. יחד עם זאת, לכל מוצר יש מנגנון עיבוד שאילתות ייחודי משל עצמו. הבנת הדרך בה מנוע מסד הנתונים מעבד שאילתות, מסייעת לארכיטקטים, מעצבים ותוכניתנים לבצע בחירות טובות כאשר הם מעצבים סכמות של מסדי נתונים וכותבים שאילתות.

כאשר שאילתה מגיעה למנוע מסד הנתונים, SQL Server מבצע שני צעדים עיקריים להפקת תוצאת השאילתה הרצויה. הצעד הראשון הוא קומפילציה של השאילתה, המייצרת **תוכנית שאילתה**, והצעד השני הוא הפעלה של תוכנית השאילתה.

קומפילציה של שאילתה ב- SQL Server 2005 מורכבת משלושה צעדים: פירוק (parsing), algebrization (מונח זה יוסבר בהמשך) ואופטימיזציה של שאילתה. לאחר ביצוע שלבים אלה, ה–compiler מאחסן את תוכנית השאילתה האופטימלית ב–cache הפרוצדורות, שם מנוע ההפעלה מעתיק את התוכנית למצב שניתן להפעלה ולאחר מכן מפעיל את הצעדים שבתוכנית השאילתה להפקת תוצאת השאילתה. אם אותה שאילתה או פרוצדורה מאוחסנת מופעלת שוב והתוכנית נמצאת ב–cache הפרוצדורות, יש לדלג על שלב הקומפילציה, והשאילתה או הפרוצדורה המאוחסנת ממשיכה ישירות להפעלה תוך שימוש חוזר באותה תוכנית מאוחסנת.

בפרק זה נבחן כיצד ה- query optimizer מייצר את תוכנית השאילתה וכיצד באפשרותך לשים את ידך הן על התוכנית המשוערת והן על התוכנית בפועל המשמשת לעיבוד השאילתה. נתחיל בדוגמה של התוצר הסופי, יחד עם תיאור המסביר כיצד המוצר מבצע את הפונקציה הרצויה – בדרך זו נבין טוב יותר את התהליך של בניית התוצר עצמו. לפיכך, אתחיל בדוגמה של הפעלת שאילתה דומה לזו איתה עבדנו בפרק 1. בהמשך, לאחר שהעקרונות יובנו, אביט מקרוב יותר לתוך תהליך הקומפילציה של השאילתה ואתאר את הצורות השונות של תוכניות השאילתה.

זרימת נתונים במהלך עיבוד שאילתה

אם ברצונך להפוך את הדוגמה הבאה לחוויה מעשית, התחל את SSMS (SQL Server Management Studio). הרץ את השאילתה המוצגת בקטע־קוד 2–1 על מסד הנתונים Northwind, לאחר שלחצת על הסמל Include Actual Execution Plan, כפי שמוצג בתרשים 2–1.

שים לב: מסד הנתונים Northwind אינו מסופק עם SQL Server 2005. תוכל להוריד את גרסת SQL Server 2000 של Northwind (שעובדת גם על SQL Server 2005) באתר Microsoft:
http://www.microsoft.com/technet/prodtechnol/sql/2000/downloads/default.mspx

קוד ההתקנה ייצור תיקייה בשם SQL Server 2000 Sample Databases בכונן C: שלך, ושם תמצא instnwnd.sql. הרץ קוד זה ב־SSMS ליצירת מסד הנתונים Northwind. לחלופין, תוכל להשתמש ב־ CREATE DATABASE Northwind FOR ATTACH… כדי להצמיד את הקבצים NORTHWND.LDF ו־NORTHWND.MDF למופע שלך של SQL Server 2005.

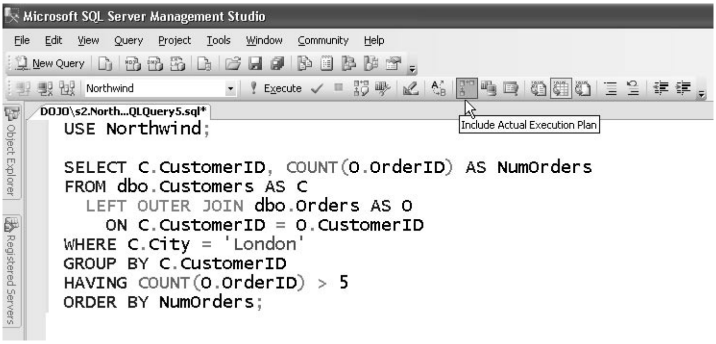
שים לב: זכור שבאפשרותך להוריד את קוד המקור לספר מאתר www://www.insidetsql.com.

קטע־קוד 2–1: שאילתה על Northwind להדגמת התוצאה של תהליך האופטימיזציה

```
USE Northwind;

SELECT C.CustomerID, COUNT(O.OrderID) AS NumOrders
FROM dbo.Customers AS C
  LEFT OUTER JOIN dbo.Orders AS O
    ON C.CustomerID = O.CustomerID
WHERE C.City = 'London'
GROUP BY C.CustomerID
HAVING COUNT(O.OrderID) > 5
ORDER BY NumOrders;
```

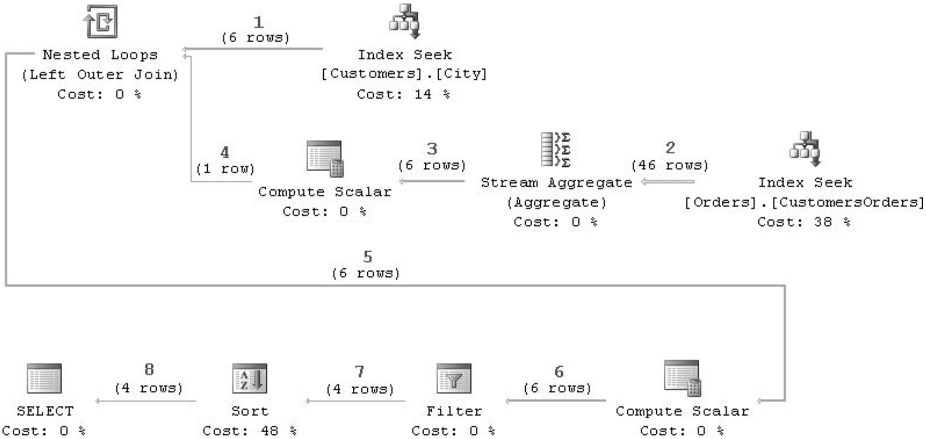
תרשים 2-1: Include Actual Execution Plan



תקבל תוכנית שאילתה גרפית דומה לזו שבתרשים 2-2 בחלונית Execution Plan של ה-SSMS.

שים לב: תוכנית העבודה בתרשים 2-2 מכילה כמה אלמנטים אשר נוספו למטרות הדגמה ולא יופיעו בתוכנית שלך. לכל חץ, הוספתי את מספר השורות המשוער בסוגריים ומספר מזהה בו אשתמש בסעיף הבא.

תרשים 2-2: תוכנית עבודה לשאילתה בקטע-קוד 2-1



שאילתה זו מחזירה את ה–ID (CustomerID) ואת מספר ההזמנות שבוצעו (NumOrders) לכל הלקוחות מלונדון שביצעו למעלה מחמש הזמנות. התוצאה מוצגת בטבלה 1–2.

טבלה 1–2: פלט השאילתה בקטע-קוד 1–2

CustomerID	*NumOrders*
EASTC	8
SEVES	9
BSBEV	10
AROUT	13

כיצד מפעיל SQL Server את התוכנית המוצגת בתרשים 2–2 להפקת התוצאה הרצויה?

ההפעלה של הענפים הבודדים אינה רלוונטית. אדגים את התהליך בדוגמה הבאה, בה SQL Server מעביר את הפעילות שלו בין שני הענפים של האופרטור Nested Loops. לשם התחלה, זכור שהחיצים האפורים בתרשים 2–2 מייצגים זרימת נתונים – שורות הנוצרות על ידי האופרטור, נצרכות על ידי האופרטור הבא בכיוון החיצים. עובי החיצים מתאים למספר היחסי של שורות שה–optimizer מעריך שיזרמו דרך הקישור.

המנוע מתחיל את ההפעלה בביצוע ה- Index Seek שבראש תרשים 2–2 על טבלת Customers – והוא יבחר את השורה הראשונה בה הלקוח הוא מלונדון. אם תרחף עם הסמן מעל הסמל Index Seek מול טבלת Customers, תוכל לראות את ביטוי החיפוש בחלון–צץ תחת Seek Predicates, Prefix: [Northwind].[dbo].[Customers].City = N'London' כפי שמוצג בתרשים 3–2. השורה הנבחרת מועברת לאופרטור Nested Loops על חץ 1, וברגע שהיא מגיעה ל- Nested Loops, מופעל הצד הפנימי של ה- Nested Loops. במקרה שלנו, בתרשים 2–2 הצד הפנימי של האופרטור Nested Loops מורכב מהאופרטורים Compute Scalar, Stream Aggregate ו– Index Seek המחוברים ל- Nested Loops על ידי חיצים 4, 3 ו–2 בהתאמה.

תרשים 3–2: חלון מידע צ'ץ עבור האופרטור Index Seek

Index Seek
Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001636
Estimated Operator Cost	0.0032886 (14%)
Estimated Subtree Cost	0.0032886
Estimated Number of Rows	6
Estimated Row Size	17 B
Ordered	True
Node ID	4

Object
[Northwind].[dbo].[Customers].[City] [C]
Output List
[Northwind].[dbo].[Customers].CustomerID
Seek Predicates
Prefix: [Northwind].[dbo].[Customers].City = N'London'

אם נחקור את האופרטור Index Seek בצד הפנימי של ה–Nested Loops בתרשים 2–2, נמצא שביטוי החיפוש שלו הוא:

Prefix: [Northwind].[dbo].[Orders].CustomerID = [Northwind].[dbo].[Customers].[CustomerID] as [C].[CustomerID]

ניתן לראות שהערך C.CustomerID משמש לחיפוש בתוך טבלת Orders להחזרת כל ההזמנות עבור ה–CustomerID. זוהי דוגמה בה הצד הפנימי של ה– Nested Loops מתייחס לערך שהושג בצד האחר, שנקרא הצד החיצוני של ה– Nested Loops.

לאחר שנמצאו כל ההזמנות ללקוח הראשון מלונדון, הן מועברות דרך החץ המסומן ב–2 לאופרטור Stream Aggregate, שם הן נספרות ותוצאת הספירה הנקראת Expr1004, מאוחסנת בשורה על ידי האופרטור Compute Scalar בין חיצים 3 ו–4. בהמשך, השורה המורכבת מה–CustomerID וספירת ההזמנות, מועברת דרך חיצים 5 ו–6 לאופרטור Filter עם הביטוי [Expr1004] > (5). Expr1004 מייצג את הביטוי COUNT(O.OrderID), וה–(5) הוא הקבוע בו השתמשנו בשאילתה להגבלת התוצאה רק ללקוחות עם למעלה מחמש הזמנות.

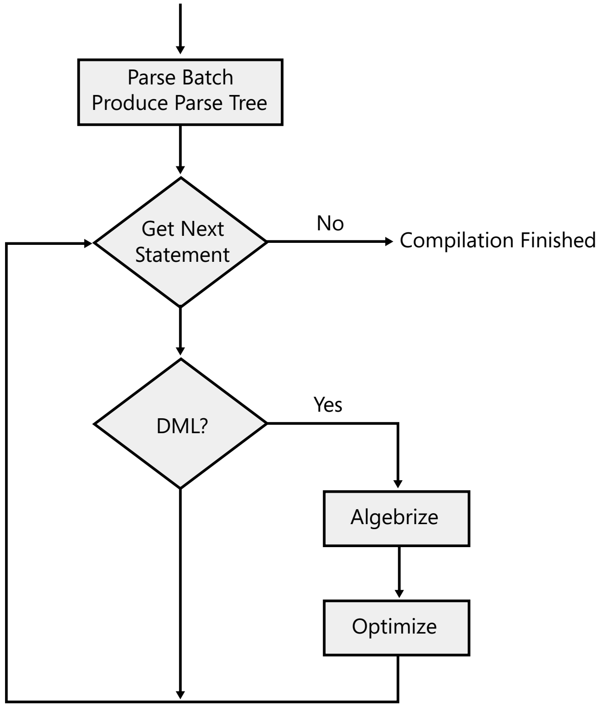
שוב ניתן לראות את הביטוי בחלון–צץ כאשר אתה מציב את הסמן על הסמל Filter. אם הביטוי מחזיר true (כלומר ללקוח יש יותר מחמש הזמנות), השורה מועברת לאופרטור Sort דרך חץ 7. שים לב ש– SQL Server אינו יכול לפלוט שום שורה מה–Sort עד שהוא אוסף את כל השורות שצריכות להיות ממוינות. זאת מכיוון שהשורה האחרונה המגיעה ל–Sort עשויה להיות זו שצריכה להיות "ראשונה" בסדר הנתון (הלקוח עם המספר הנמוך ביותר של הזמנות שהוא מעל חמש במקרה שלנו). לפיכך, השורות "מחכות" ב–Sort, והתהליך המתואר לעיל חוזר על עצמו עבור הלקוח הלונדוני הבא שיימצא על ידי אופרטור ה– Index Seek על טבלת Customers. ברגע שכל השורות שצריכות להיות מוחזרות מגיעות לאופרטור Sort, הוא יחזיר אותן בסדר הנכון (חץ 8).

קומפילציה

batch הוא מצבור של משפט או מספר משפטי Transact-SQL שעוברים קומפילציה כיחידה אחת. פרוצדורה מאוחסנת היא דוגמה ל–batch. דוגמה נוספת היא סט של משפטים בחלון SQL Query ב–SSMS. הפקודה GO מחלקת סטים של משפטים ל–batches נפרדים. שים לב ש–GO אינו משפט T-SQL. SQLCMD, OSQL ו–SSMS משתמשים במילה השמורה GO לסימון סוף ה–batch.

SQL Server עושה קומפילציה למשפטי batch לתוך יחידה אחת הניתנת להפעלה הנקראת **תוכנית עבודה** (**execution plan**). במהלך הקומפילציה, ה–compiler מרחיב את המשפטים על ידי הוספת האילוצים, הטריגרים ופעולות ה–cascade הרלוונטיים שצריכים להיות מופעלים בעת הרצת המשפט. אם ה–batch שעובר קומפילציה מכיל קריאות לפרוצדורות מאוחסנות או לפונקציות אחרות והתוכניות שלהן אינן ב–cache, הפרוצדורות המאוחסנות והפונקציות עוברות קומפילציה גם הן בצורה רקורסיבית. הצעדים העיקריים בקומפילציה של batch מוצגים בתרשים 4–2.

תרשים 4–2: קומפילציה



חשוב לזכור שקומפילציה והפעלה הם שלבים נפרדים בעיבוד שאילתה ושהפער בין הזמן בו SQL Server עושה קומפילציה לשאילתה לבין הזמן בו השאילתה מופעלת יכול להיות קצר כמיליוניות שנייה בודדות או ארוך כמספר ימים. לשאילתה אד–הוק לרוב אין תוכנית עבודה ב–cache כאשר היא מעובדת; לפיכך, היא עוברת קומפילציה והתוכנית שלה מופעלת מיידית. מצד שני, לאחר שהתכנית עוברת קומפילציה לפרוצדורה מאוחסנת המופעלת לעיתים קרובות, היא עשויה להיות מאוחסנת ב–cache הפרוצדורות למשך זמן ארוך מאוד. זאת מכיוון ש– SQL Server מסיר את התוכניות שאינן בשימוש תכוף מ–cache הפרוצדורות קודם לכן, במידה שהזיכרון נדרש למטרות אחרות, כמו אחסנת תוכניות שאילתה חדשות.

ה–optimizer לוקח בחשבון כמה מעבדים זמינים ל– SQL Server ואת כמות הזיכרון הזמינה להפעלת שאילתות. עם זאת, מספר המעבדים הזמינים להפעלת השאילתה והכמות של הזיכרון הזמין יכולים להשתנות משמעותית מרגע לרגע. חשוב להתייחס לקומפילציה ולהפעלה כשתי פעילויות נפרדות, גם כאשר אתה שולח משפט SQL אד–הוק דרך SSMS ומפעיל אותו מיידית.

כאשר SQL Server מוכן לעבד batch, תוכנית עבודה ל–batch עשויה כבר להיות זמינה ב–cache של SQL Server. אם לא, ה–compiler מבצע קומפילציה ל–batch ומייצר תוכנית שאילתה. תהליך הקומפילציה כולל מספר דברים. תחילה, SQL Server עובר דרך השלבים של פירוק (parsing) וכריכה (binding). פירוק הוא התהליך של בדיקת התחביר והפיכת ה– SQL batch שלך לעץ פירוק (parse tree). פירוק הוא פעולה כללית המשמשת קומפיילרים כמעט בכל שפות התכנות. הדבר היחיד הייחודי ל–parser של SQL Server הוא החוקים בהם הוא משתמש להגדרת תחביר T-SQL תקני.

פירוק בודק למשל, האם טבלה או שם טור מתחילים בספרה. אם נמצאת ספרה, ה–parser מרים דגל שגיאה. עם זאת, פירוק אינו בודק האם טור הנמצא בשימוש בפסוקית WHERE אכן קיים באחת הטבלאות המופיעות בפסוקית FROM; נושא זה מטופל במהלך הכריכה.

תהליך הכריכה קובע את מאפייני האובייקטים אליהם אתה מתייחס במשפטי ה–SQL שלך, והוא בודק האם הסמנטיקה בה אתה משתמש הגיונית. למשל, בעוד ששאילתה הכוללת FROM A JOIN B עשויה לעבור פירוק בהצלחה, כריכה תכשל אם A הוא טבלה ו–B הוא פרוצדורה מאוחסנת.

השלב האחרון בקומפילציה היא אופטימיזציה. על ה–optimizer לתרגם את הבקשה הלא–פרוצדורלית של משפט SQL מבוסס–סטים לפרוצדורה שיכולה להיות מופעלת בהצלחה ולהחזיר את התוצאה הרצויה. בדומה לכריכה, אופטימיזציה מתבצעת על כל משפט בנפרד עבור כל המשפטים ב–batch. לאחר שהקומפיילר מייצר את התוכנית עבור ה–batch ומאחסן את התוכנית ב–cache הפרוצדורות, מופעל עותק מיוחד של **הקשר ההפעלה (execution context)** של התוכנית. SQL Server מאחסן ב–cache את הקשר ההפעלה בדומה לדרך בה הוא מאחסן את תוכניות השאילתה, ואם אותו batch מתחיל

את ההפעלה השנייה בטרם הראשונה מסתיימת, SQL Server ייצר את הקשר ההפעלה השני מאותה תוכנית. תוכל ללמוד עוד על cache הפרוצדורות של SQL Server מהספר: Inside Microsoft SQL Server 2005: Query Processing and Optimization (Microsoft Press, 2006) מאת קיילן דלייני (Kalen Delaney) או מה–whitepaper: Batch Compilation, Recompilation, and Plan Caching Issues in SQL Server 2005 אותו ניתן למצוא ב–: http://www.microsoft.com/technet/prodtechnol/sql/2005/recomp.mspx#.

SQL Server אינו מבצע אופטימיזציה לכל משפט ב–batch. הוא מבצע אופטימיזציה רק לסוגי משפטים מסוימים: אלו הניגשים לטבלאות מסד הנתונים ואשר עשויים להיות להם מספר אפשרויות הפעלה. SQL Server מבצע אופטימיזציה לכל משפטי ה–DML (שפה למניפולציית נתונים) – אלו הם משפטי SELECT, INSERT, DELETE ו–UPDATE. בנוסף ל–DML, מבוצעת אופטימיזציה גם לכמה משפטי T-SQL אחרים, CREATE INDEX הוא אחד מהם. רק המשפטים שעברו אופטימיזציה יפיקו תוכניות שאילתה. הדוגמה הבאה מראה שה–optimizer יוצר תוכנית עבור CREATE INDEX:

```
CREATE TABLE dbo.T(a INT, b INT, c INT, d INT);
INSERT INTO dbo.T VALUES(1, 1, 1, 1);
SET STATISTICS PROFILE ON; -- forces producing showplan from execution
CREATE INDEX i ON dbo.T(a, b) INCLUDE(c, d);
SET STATISTICS PROFILE OFF; -- reverse showplan setting
DROP TABLE dbo.T; -- remove the table
```

הוא יפיק את תוכנית השאילתה הבאה שעברה אופטימיזציה:

```
insert [dbo].[T] select *, %%bmk%% from [dbo].[T]
 |--Index Insert(OBJECT:([db].[dbo].[T].[i]))
  |--Sort(ORDER BY:([db].[dbo].[T].[a] ASC, [db].[dbo].[T].[b] ASC, [Bmk1000] ASC))
    |--Table Scan(OBJECT:([db].[dbo].[T]))
```

בדומה למשפט CREATE INDEX, גם המשפטים CREATE STATISTICS, UPDATE STATISTICS, וכמה צורות של ALTER INDEX עוברים אופטימיזציה. מספר משפטים המופעלים פנימית כדי לבצע בדיקת מסד נתונים (DBCC CHECKDB) עוברים אופטימיזציה גם הם. עם זאת, יש לזכור שמתוך המשפטים שעברו אופטימיזציה שאינם משפטי DML, רק CREATE INDEX מייצר showplan על ידי אפשרות ה– STATISTICS PROFILE ואף אחד מהם אינו מייצר תוכנית שאילתה ישירות ב–SSMS. (showplan יוסבר מאוחר יותר בסעיף "עבודה עם תוכנית השאילתה").