Chapter 11
Database Performance Tuning and Query Optimization

Learning Objectives

- In this chapter, you will learn:
  - Basic database performance-tuning concepts
  - How a DBMS processes SQL queries
  - About the importance of indexes in query processing
  - About the types of decisions the query optimizer has to make
  - Some common practices used to write efficient SQL code
  - How to formulate queries and tune the DBMS for optimal performance

Database Performance-Tuning Concepts

- Goal of database performance is to execute queries as fast as possible
- **Database performance tuning**: Set of activities and procedures that reduce response time of database system
- Fine-tuning the performance of a system requires that all factors must operate at optimum level with minimal bottlenecks

Table 11.1 - General Guidelines for Better System Performance

TABLE 11.1

GENERAL GUIDELINES FOR BETTER SYSTEM PERFORMANCE

	SYSTEM RESOURCES	CLIENT	SERVER
Hardware	CPU	The fastest possible Dual-core CPU or higher	The fastest possible Multiple processors (quad-core technology) Cluster of networked computers
	RAM	The maximum possible to avoid OS memory to disk swapping	The maximum possible to avoid OS memory to disk swapping
	Hard disk	Fast SATA/EIDE hard disk with sufficient free hard disk space Solid State Drives (SSD) for faster speed	Multiple high-speed, high-capacity disks Fast disk interface (SAS / SCSI / Firewire / Fibre Channel RAID configuration optimized for throughput Solid State Drives (SSD) for faster speed Separate disks for OS, DBMS, and data spaces
	Network	High-speed connection	High-speed connection
Software	Operating System (OS)	64-bit OS for larger address spaces Fine-tuned for best client application performance	64-bit OS for larger address spaces Fine-tuned for best server application performance
	Network	Fine-tuned for best throughput	Fine-tuned for best throughput
	Application	Optimize SQL in client application	Optimize DBMS server for best performance

Performance Tuning: Client and Server

- Client side
  - **SQL performance tuning**: Generates SQL query that returns correct answer in least amount of time
    - Using minimum amount of resources at server
- Server side
  - **DBMS performance tuning:** DBMS environment configured to respond to clients' requests as fast as possible
    - Optimum use of existing resources

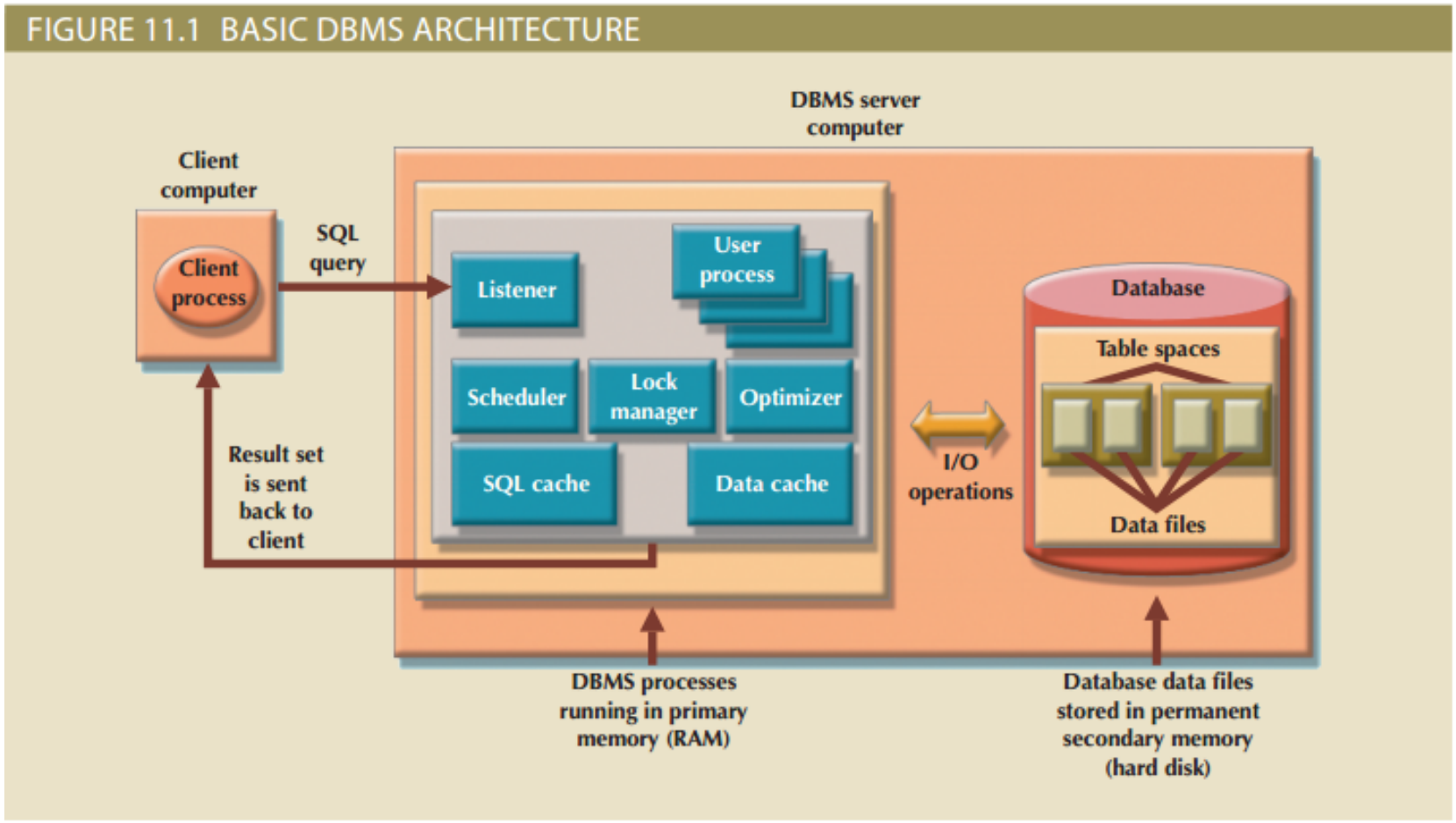
DBMS Architecture

- All data in a database are stored in **data files**
  - Data files automatically expand in predefined increments known as **extends**
- Data files are grouped in file groups or table spaces
  - **Table space** or **file group**: Logical grouping of several data files that store data with similar characteristics
- **Data cache** or **buffer cache**: Shared, reserved memory area
  - Stores most recently accessed data blocks in RAM

DBMS Architecture

- **SQL cache** or **procedure cache**: Stores most recently executed SQL statements or PL/SQL procedures
- DBMS retrieves data from permanent storage and places them in RAM
- **Input/output request**: Low-level data access operation that reads or writes data to and from computer devices
- Data cache is faster than working with data files
- Majority of performance-tuning activities focus on minimizing I/O operations

Figure 11.1 - Basic DBMS Architecture



Database Query Optimization Modes

- Algorithms proposed for query optimization are based on:
  - Selection of the optimum order to achieve the fastest execution time
  - Selection of sites to be accessed to minimize communication costs
- Evaluated on the basis of:
  - Operation mode
  - Timing of its optimization

Classification of Operation Modes

- **Automatic query optimization**: DBMS finds the most cost-effective access path without user intervention
- **Manual query optimization**: Requires that the optimization be selected and scheduled by the end user or programmer

Classification Based on Timing of Optimization

- **Static query optimization**: best optimization strategy is selected when the query is compiled by the DBMS
  - Takes place at compilation time
- **Dynamic query optimization**: Access strategy is dynamically determined by the DBMS at run time, using the most up-to-date information about the database
  - Takes place at execution time

Classification Based on Type of Information Used to Optimize the Query

- **Statistically based query optimization algorithm**: Statistics are used by the DBMS to determine the best access strategy
- Statistical information is generated by DBMS through:
  - **Dynamic statistical generation mode**
  - **Manual statistical generation mode**
- **Rule-based query optimization algorithm**: based on a set of user-defined rules to determine the best query access strategy

Table 11.2 - Sample Database Statistics Measurements

TABLE 11.2

SAMPLE DATABASE STATISTICS MEASUREMENTS

DATABASE OBJECT	SAMPLE MEASUREMENTS
Tables	Number of rows, number of disk blocks used, row length, number of columns in each row, number of distinct values in each column, maximum value in each column, minimum value in each column, and columns that have indexes
Indexes	Number and name of columns in the index key, number of key values in the index, number of distinct key values in the index key, histogram of key values in an index, and number of disk pages used by the index
Environment Resources	Logical and physical disk block size, location and size of data files, and number of extends per data file

Query Processing

Parsing

- DBMS parses the SQL query and chooses the most efficient access/execution plan

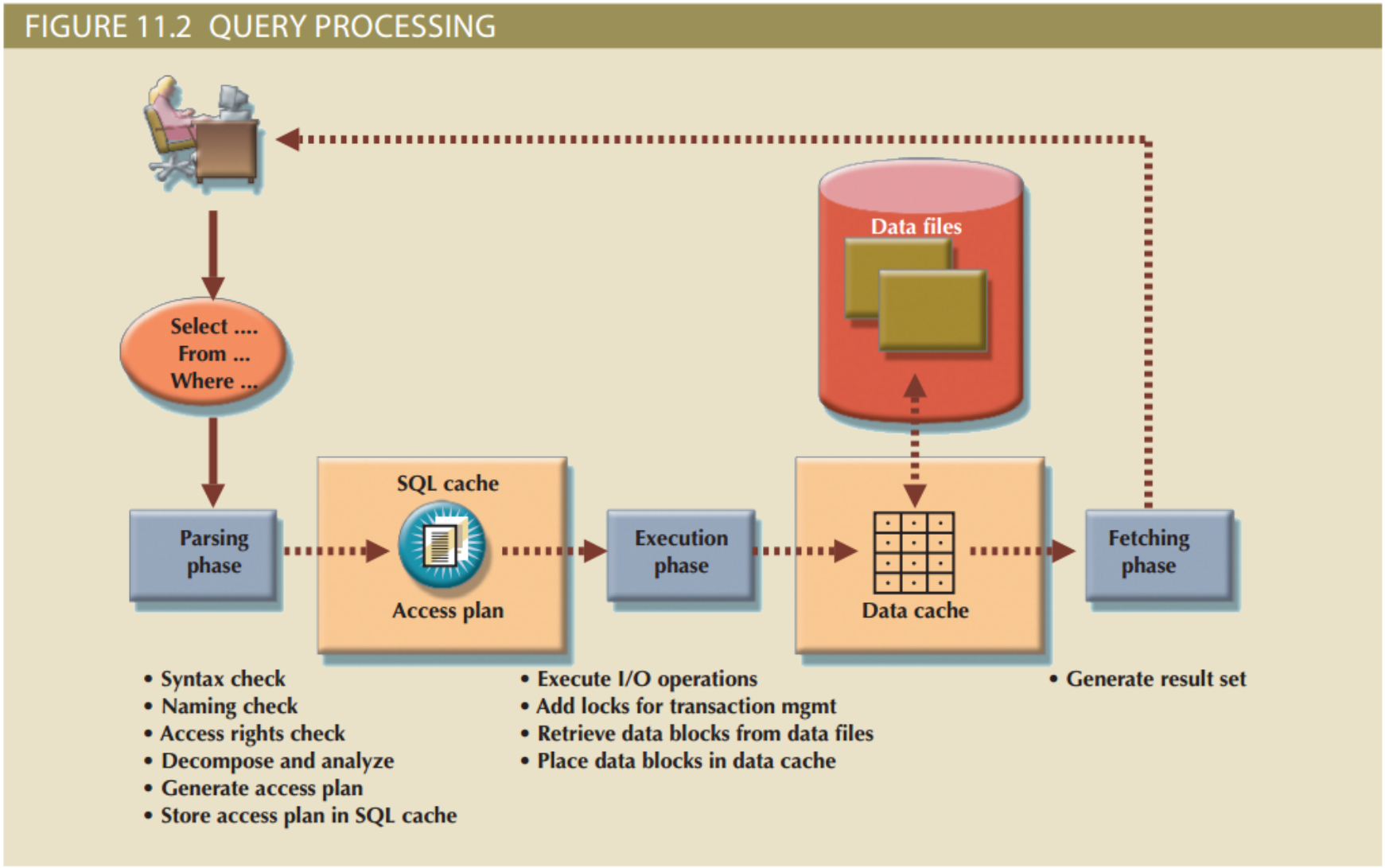
Execution

- DBMS executes the SQL query using the chosen execution plan

Fetching

- DBMS fetches the data and sends the result set back to the client

Figure 11.2 – Query Processing



SQL Parsing Phase

- Query is broken down into smaller units
- Original SQL query transformed into slightly different version of original SQL code which is fully equivalent and more efficient
- **Query optimizer**: Analyzes SQL query and finds most efficient way to access data
- **Access plans**: DBMS-specific and translate client's SQL query into a series of complex I/O operations
- If access plan already exists for query in SQL cache, DBMS reuses it
  - If not, optimizer evaluates various plans and chooses one to be placed in SQL cache for use

SQL Execution Phase

- All I/O operations indicated in the access plan are executed
  - Locks are acquired
  - Data are retrieved and placed in data cache
  - Transaction management commands are processed

SQL Fetching Phase

- Rows of resulting query result set are returned to client
  - DBMS may use temporary table space to store temporary data
  - Database server coordinates the movement of the result set rows from the server cache to the client cache

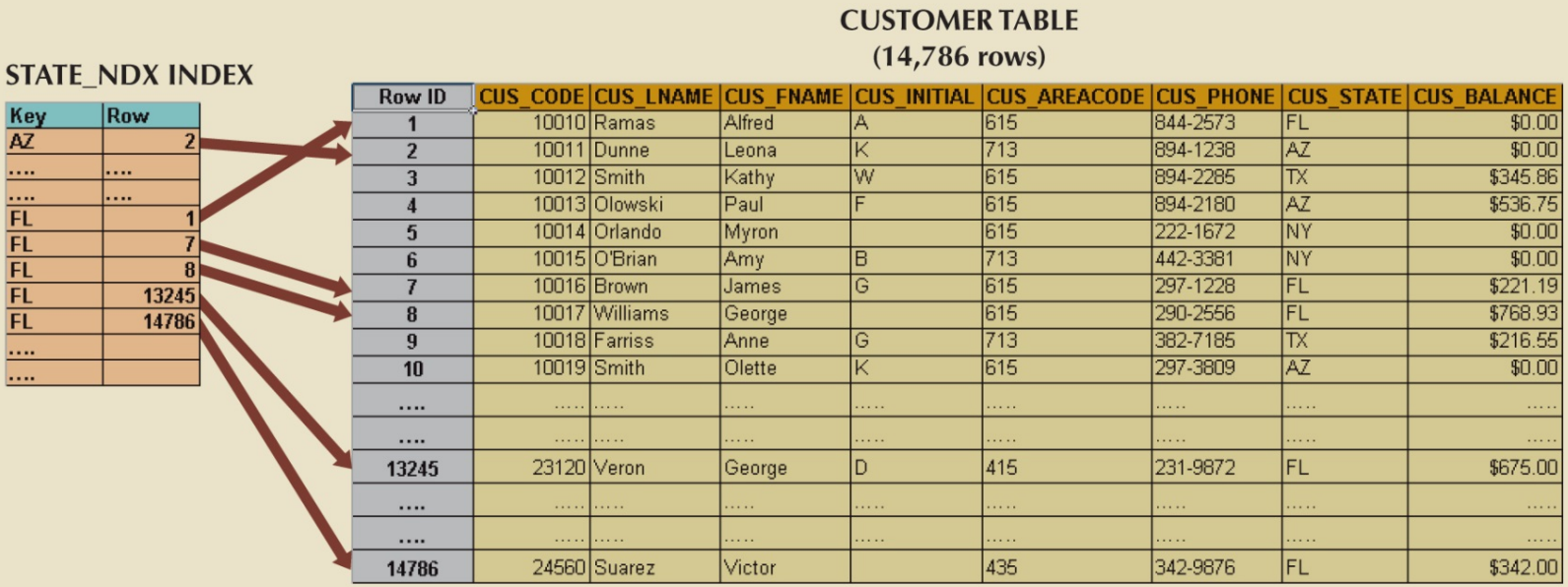
Query Processing Bottlenecks

- Delay introduced in the processing of an I/O operation that slows the system
- Caused by the:
  - CPU
  - RAM
  - Hard disk
  - Network
  - Application code

Indexes and Query Optimization

- Indexes
  - Help speed up data access
  - Facilitate searching, sorting, using aggregate functions, and join operations
  - Ordered set of values that contain the index key and pointers
  - More efficient than a full table scan

FIGURE 11.3 INDEX REPRESENTATION FOR THE CUSTOMER TABLE



STATE_NDX INDEX

Key	Row
AZ	2
....	....
....	....
FL	1
FL	7
FL	8
FL	13245
FL	14786
....	
....	

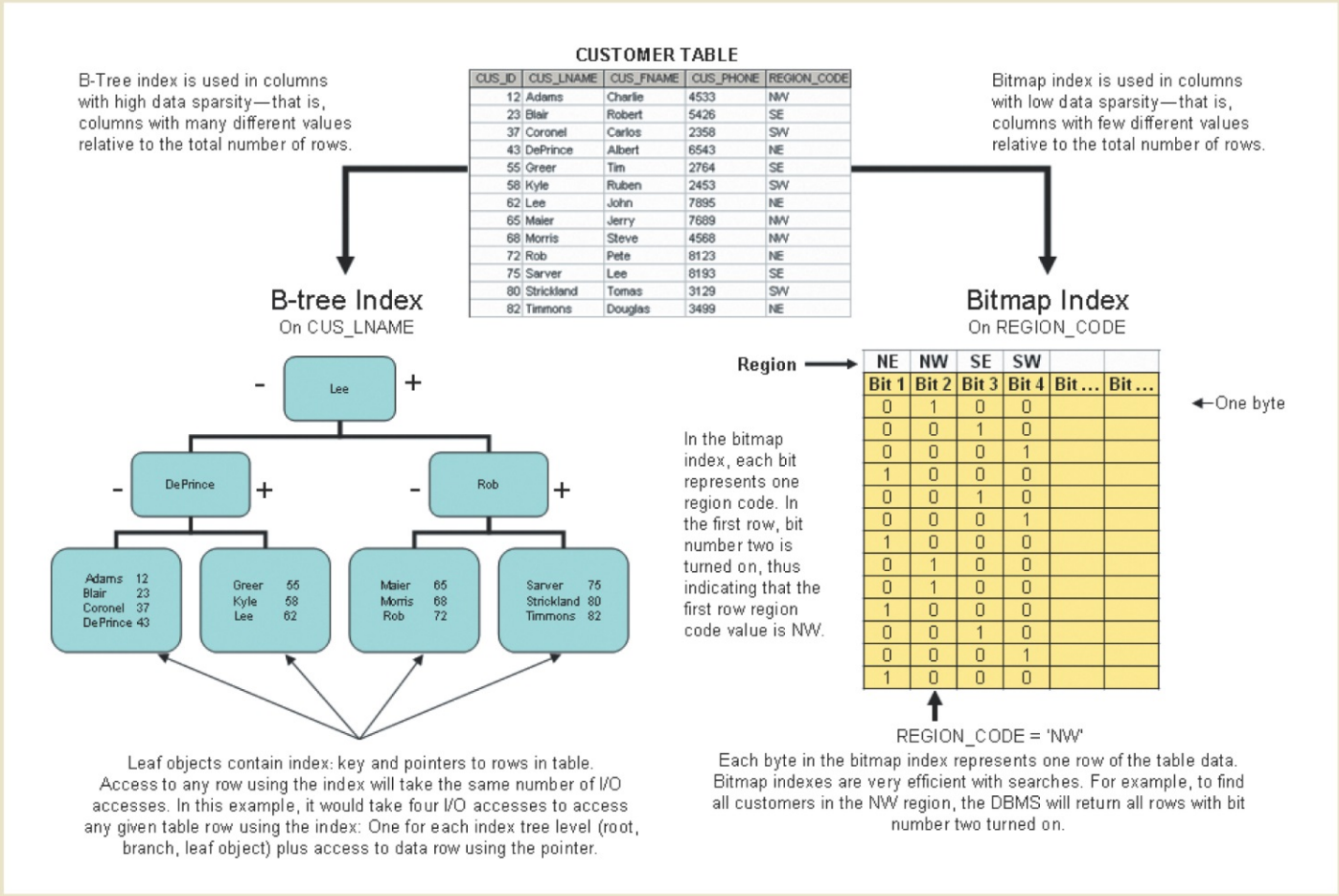
CUSTOMER TABLE (14,786 rows)

Row ID	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_STATE	CUS_BALANCE
1	10010	Ramas	Alfred	A	615	844-2573	FL	$0.00
2	10011	Dunne	Leona	K	713	894-1238	AZ	$0.00
3	10012	Smith	Kathy	W	615	894-2285	TX	$345.86
4	10013	Olowski	Paul	F	615	894-2180	AZ	$536.75
5	10014	Orlando	Myron		615	222-1672	NY	$0.00
6	10015	O'Brian	Amy	B	713	442-3381	NY	$0.00
7	10016	Brown	James	G	615	297-1228	FL	$221.19
8	10017	Williams	George		615	290-2556	FL	$768.93
9	10018	Farriss	Anne	G	713	382-7185	TX	$216.55
10	10019	Smith	Olette	K	615	297-3809	AZ	$0.00
....	.....	.....	.....	.....	.....	.....	.....	.....
....	.....	.....	.....	.....	.....	.....	.....	.....
13245	23120	Veron	George	D	415	231-9872	FL	$675.00
....	.....	.....	.....	.....	.....	.....	.....	.....
....	.....	.....	.....	.....	.....	.....	.....	.....
14786	24560	Suarez	Victor		435	342-9876	FL	$342.00

Indexes and Query Optimization

- **Data sparsity**: Number of different values a column could have
- Data structures used to implement indexes:
  - **Hash indexes**
  - **B-tree indexes**
  - **Bitmap indexes**
- DBMSs determine best type of index to use

FIGURE 11.4 B-TREE AND BITMAP INDEX REPRESENTATION



Optimizer Choices

- **Rule-based optimizer**: Uses preset rules and points to determine the best approach to execute a query
- **Cost-based optimizer**: Uses algorithms based on statistics about objects being accessed to determine the best approach to execute a query

TABLE 11.4

COMPARING ACCESS PLANS AND I/O COSTS

PLAN	STEP	OPERATION	I/O OPERATIONS	I/O COST	RESULTING SET ROWS	TOTAL I/O COST
A	A1	Cartesian product (PRODUCT, VENDOR)	7,000 + 300	7,300	2,100,000	7,300
	A2	Select rows in A1 with matching vendor codes	2,100,000	2,100,000	7,000	2,107,300
	A3	Select rows in A2 with V_STATE = 'FL'	7,000	7,000	1,000	**2,114,300**
B	B1	Select rows in VENDOR with V_STATE = 'FL'	300	300	10	300
	B2	Cartesian Product (PRODUCT, B1)	7,000 + 10	7,010	70,000	7,310
	B3	Select rows in B2 with matching vendor codes	70,000	70,000	1,000	**77,310**

Using Hints to Affect Optimizer Choices

- Optimizer might not choose the best execution plan
  - Makes decisions based on existing statistics, which might be old
  - Might choose less-efficient decisions
- **Optimizer hints**: Special instructions for the optimizer, embedded in the SQL command text

Table 11.5 - Optimizer Hints

TABLE 11.5

OPTIMIZER HINTS

HINT	USAGE
ALL_ROWS	Instructs the optimizer to minimize the overall execution time—that is, to minimize the time needed to return all rows in the query result set. This hint is generally used for batch mode processes. For example: SELECT /*+ ALL_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;
FIRST_ROWS	Instructs the optimizer to minimize the time needed to process the first set of rows—that is, to minimize the time needed to return only the first set of rows in the query result set. This hint is generally used for interactive mode processes. For example: SELECT /*+ FIRST_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;
INDEX(name)	Forces the optimizer to use the P_QOH_NDX index to process this query. For example: SELECT /*+ INDEX(P_QOH_NDX) */ * FROM PRODUCT WHERE P_QOH < 10

SQL Performance Tuning

- Evaluated from client perspective
  - Most current relational DBMSs perform automatic query optimization at the server end
  - Most SQL performance optimization techniques are DBMS-specific and thus rarely portable
- Majority of performance problems are related to poorly written SQL code

Index Selectivity

- Measure of the likelihood that an index will be used in query processing
- Indexes are used when a subset of rows from a large table is to be selected based on a given condition
- Index cannot always be used to improve performance
- **Function-based index**: Based on a specific SQL function or expression

Conditional Expressions

- Expressed within WHERE or HAVING clauses of a SQL statement
  - Restricts the output of a query to only rows matching conditional criteria
- Guidelines to write efficient conditional expressions in SQL code
  - Use simple columns or literals as operands
  - Numeric field comparisons are faster than character, date, and NULL comparisons

Conditional Expressions

- Equality comparisons are faster than inequality comparisons
- Transform conditional expressions to use literals
- Write equality conditions first when using multiple conditional expressions
- When using multiple AND conditions, write the condition most likely to be false first
- When using multiple OR conditions, put the condition most likely to be true first
- Avoid the use of NOT logical operator

Table 11.6 – Conditional Criteria

TABLE 11.6

CONDITIONAL CRITERIA

OPERAND1	CONDITIONAL OPERATOR	OPERAND2
P_PRICE	>	10.00
V_STATE	=	FL
V_CONTACT	LIKE	Smith%
P_QOH	>	P_MIN * 1.10

Query Formulation

- Identify what columns and computations are required
- Identify source tables
- Determine how to join tables
- Determine what selection criteria are needed
- Determine the order in which to display the output

DBMS Performance Tuning

- Managing DBMS processes in primary memory and the structures in physical storage
- DBMS performance tuning at server end focuses on setting parameters used for:
  - Data cache
  - SQL cache
  - Sort cache
  - Optimizer mode
- **In-memory database**: Store large portions of the database in primary storage

DBMS Performance Tuning

- Recommendations for physical storage of databases:
  - Use **RAID** (Redundant Array of Independent Disks) to provide a balance between performance improvement and fault tolerance
  - Minimize disk contention
  - Put high-usage tables in their own table spaces
  - Assign separate data files in separate storage volumes for indexes, system, and high-usage tables

DBMS Performance Tuning

- Take advantage of the various table storage organizations in the database
  - **Index-organized table** or **clustered index table**: Stores the end-user data and the index data in consecutive locations in permanent storage
- Partition tables based on usage
- Use denormalized tables where appropriate
- Store computed and aggregate attributes in tables

Query Optimization Example

FIGURE 11.5 INITIAL EXPLAIN PLAN

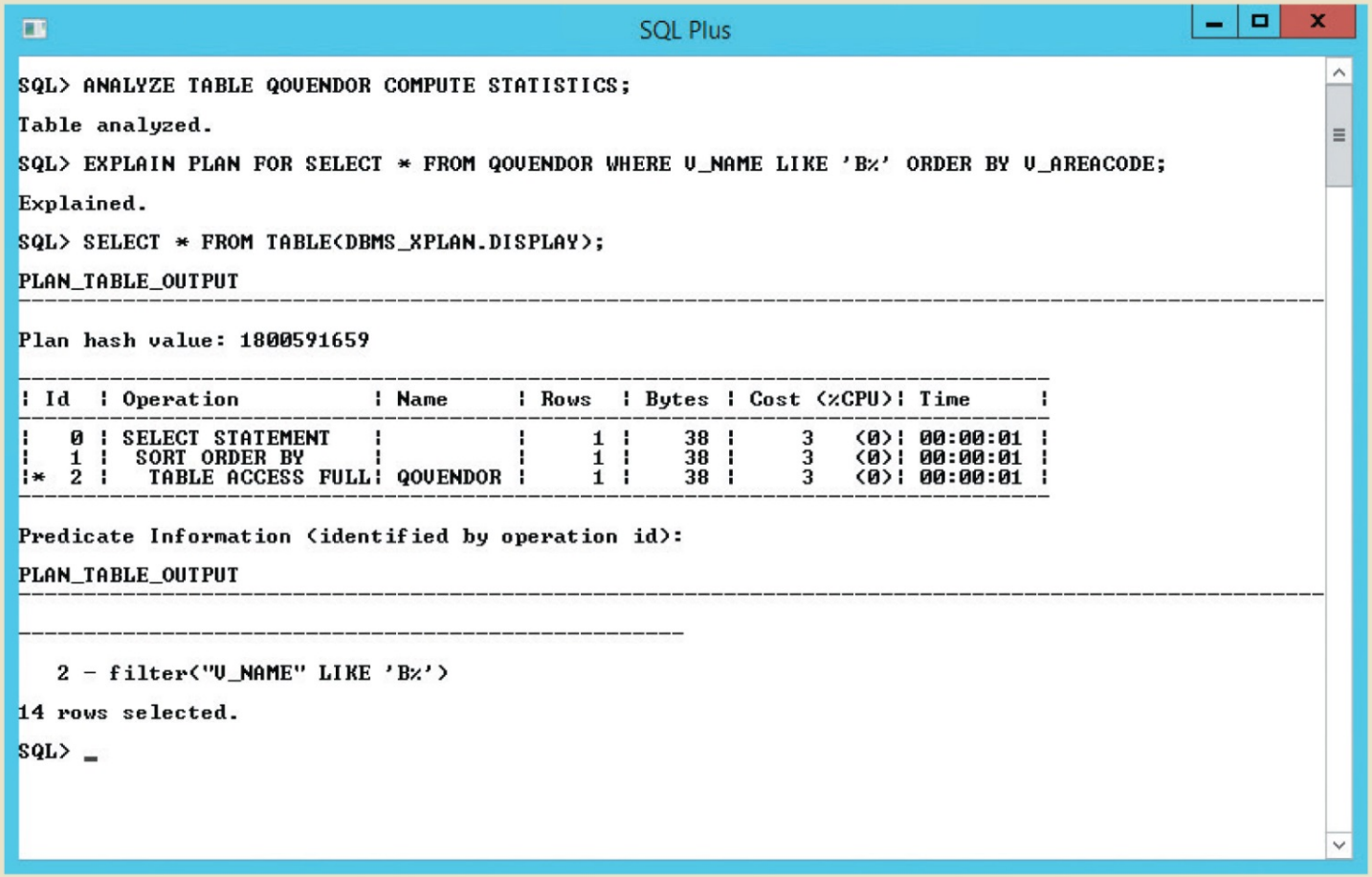


FIGURE 11.6 EXPLAIN PLAN AFTER INDEX ON V_AREACODE

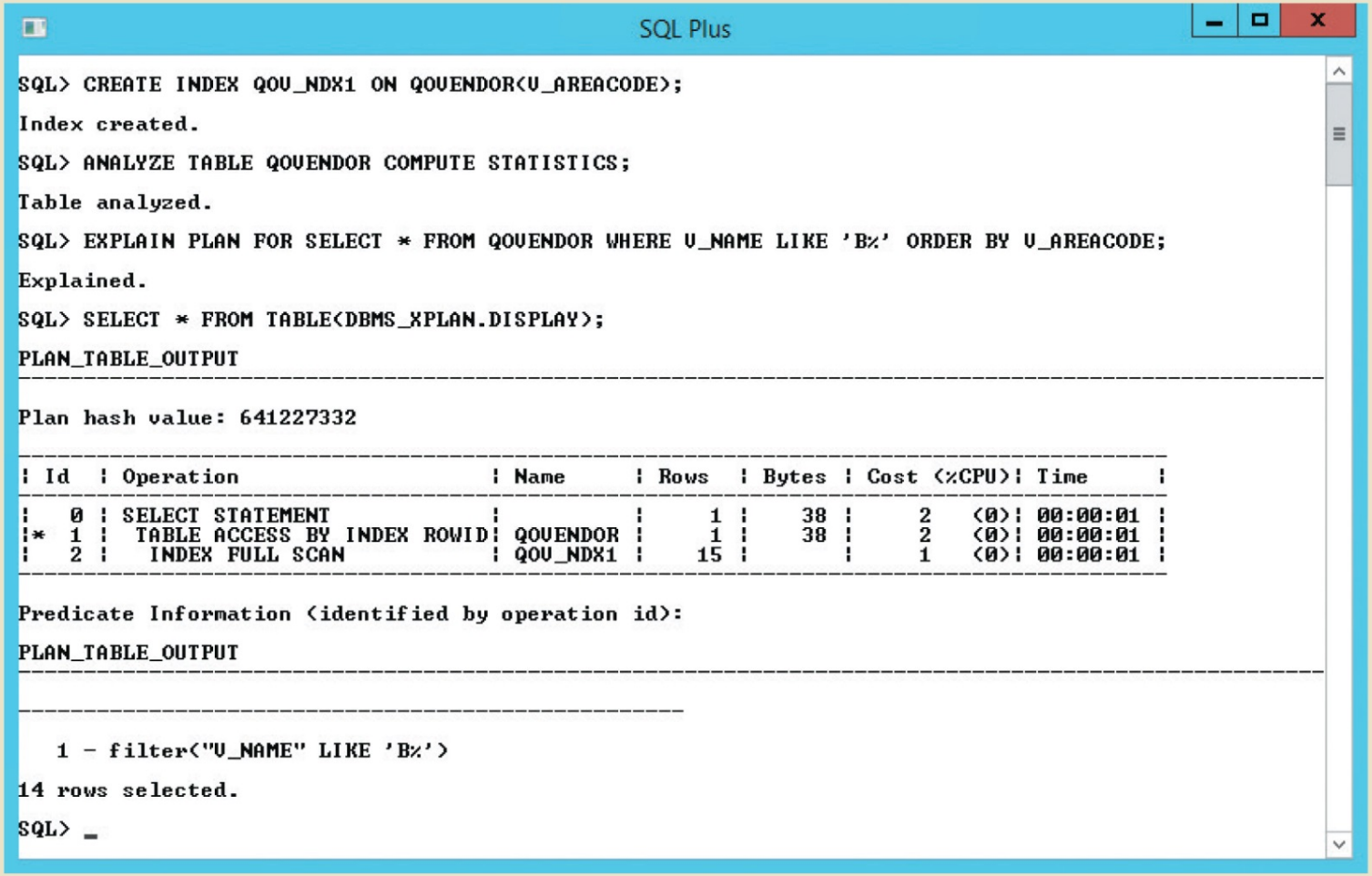


FIGURE 11.7 EXPLAIN PLAN AFTER INDEX ON V_NAME

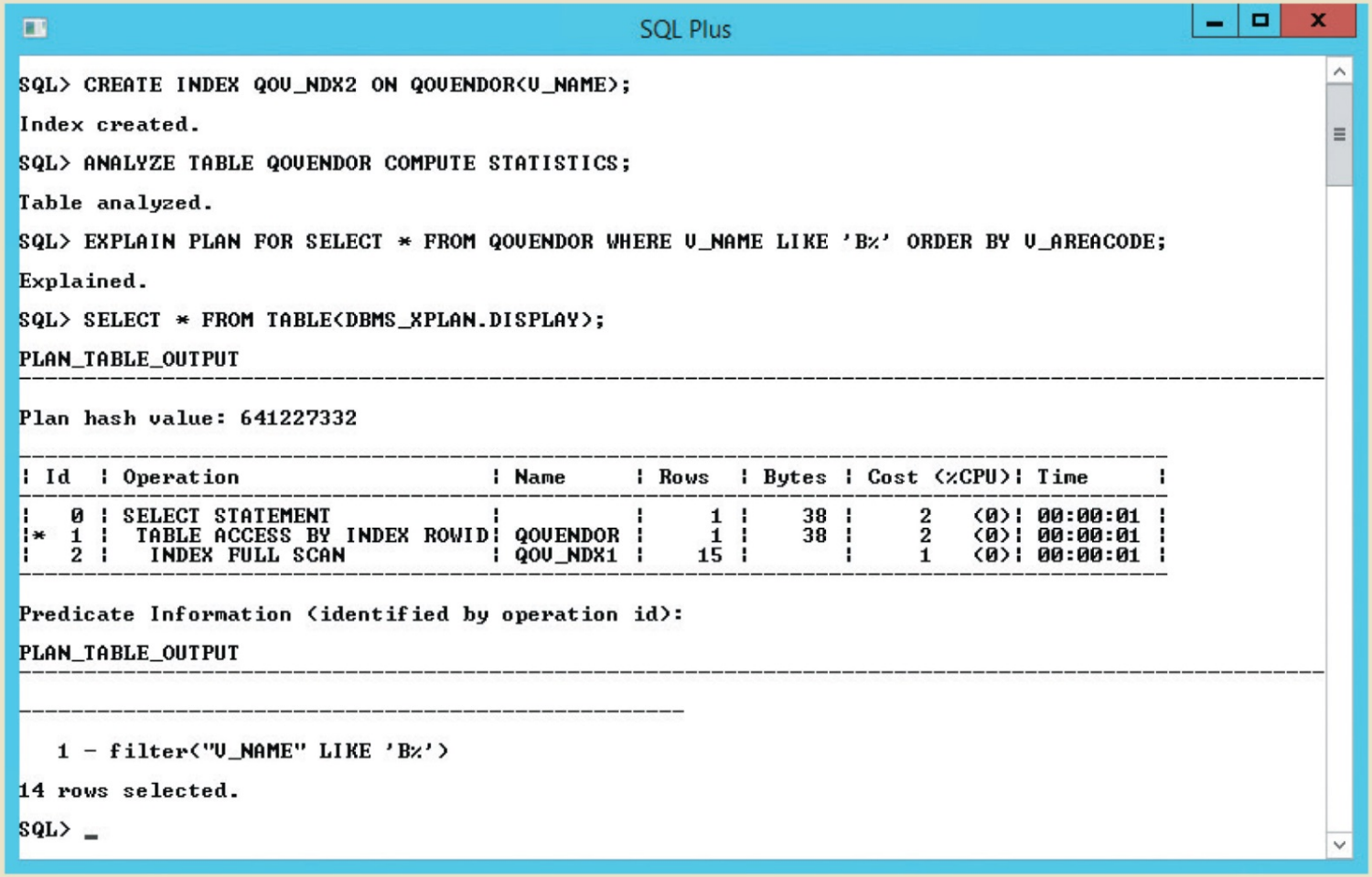


FIGURE 11.8 ACCESS PLAN USING INDEX ON V_NAME

SQL Plus

```
SQL> EXPLAIN PLAN FOR SELECT V_NAME, P_CODE FROM QOVENDOR V, QOPRODUCT P
  2                  WHERE  V.V_CODE = P.V_CODE AND V_NAME = 'ORDVA, Inc.';

Explained.

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

Plan hash value: 4146133223

-------------------------------------------------------------------------------
| Id  | Operation                           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |            |     2 |    62 |     5   (0)| 00:00:01 |
|*  1 |  HASH JOIN                          |            |     2 |    62 |     5   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| QOVENDOR   |     1 |    17 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                 | QOV_NDX2   |     1 |       |     1   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL                 | QOPRODUCT  |    14 |   196 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------


Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("V"."V_CODE"="P"."V_CODE")
   3 - access("V_NAME"='ORDVA, Inc.')
   4 - filter("P"."V_CODE" IS NOT NULL)

18 rows selected.

SQL> _
```

FIGURE 11.9 ACCESS PLAN USING FUNCTIONS ON INDEXED COLUMNS

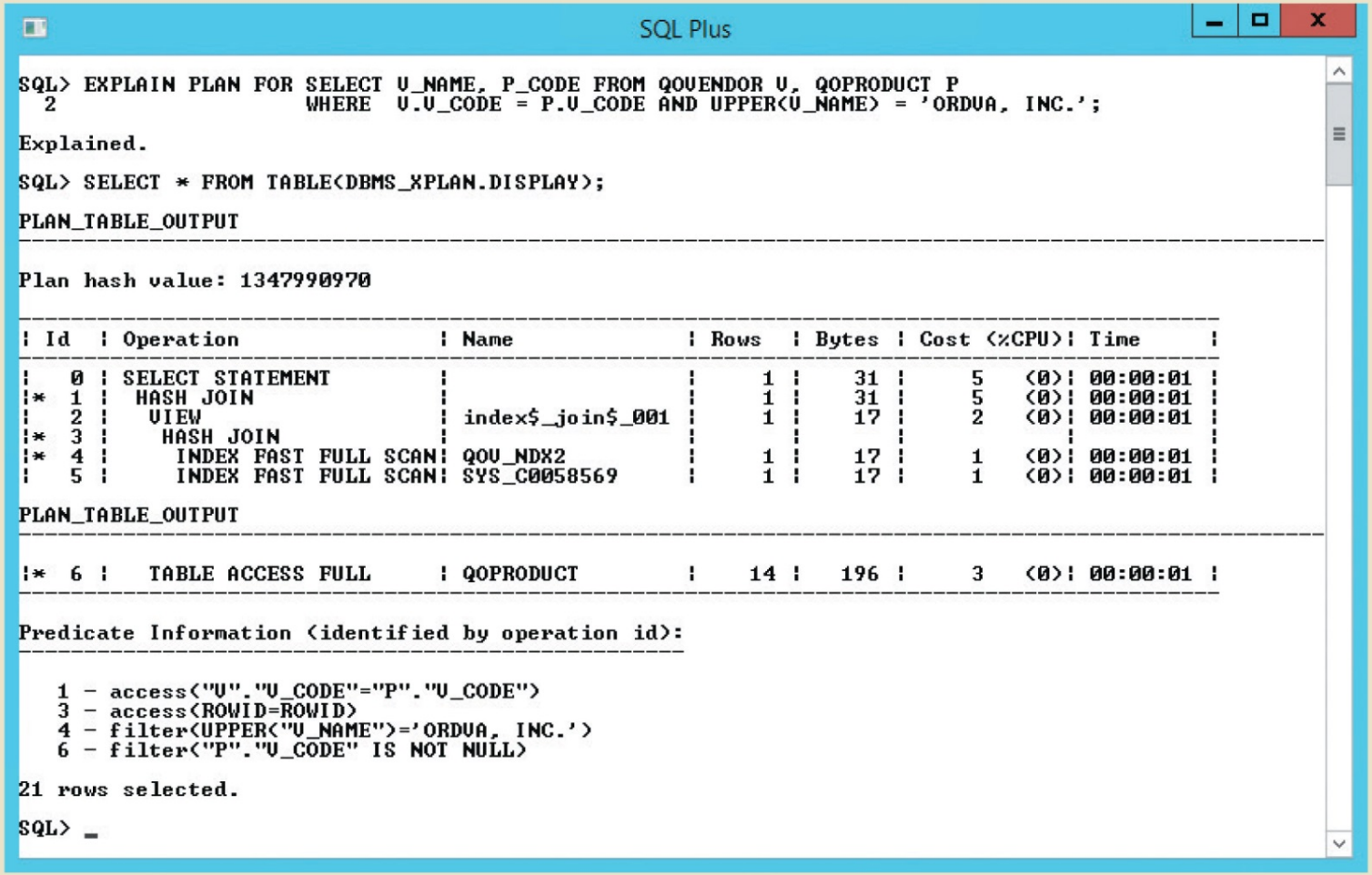


FIGURE 11.10 FIRST EXPLAIN PLAN: AGGREGATE FUNCTION ON A NON-INDEXED COLUMN

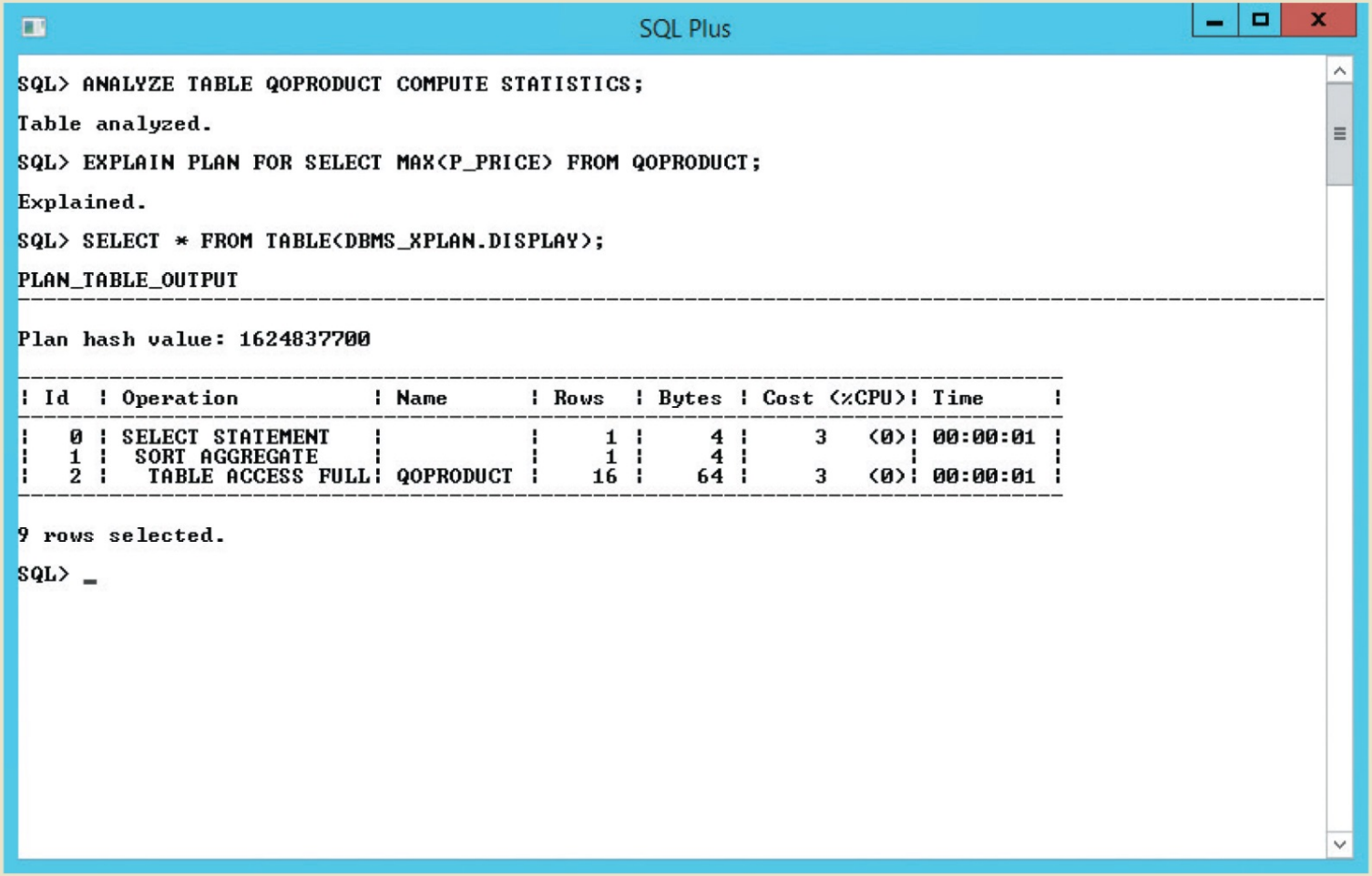


FIGURE 11.11 SECOND EXPLAIN PLAN: AGGREGATE FUNCTION ON AN INDEXED COLUMN

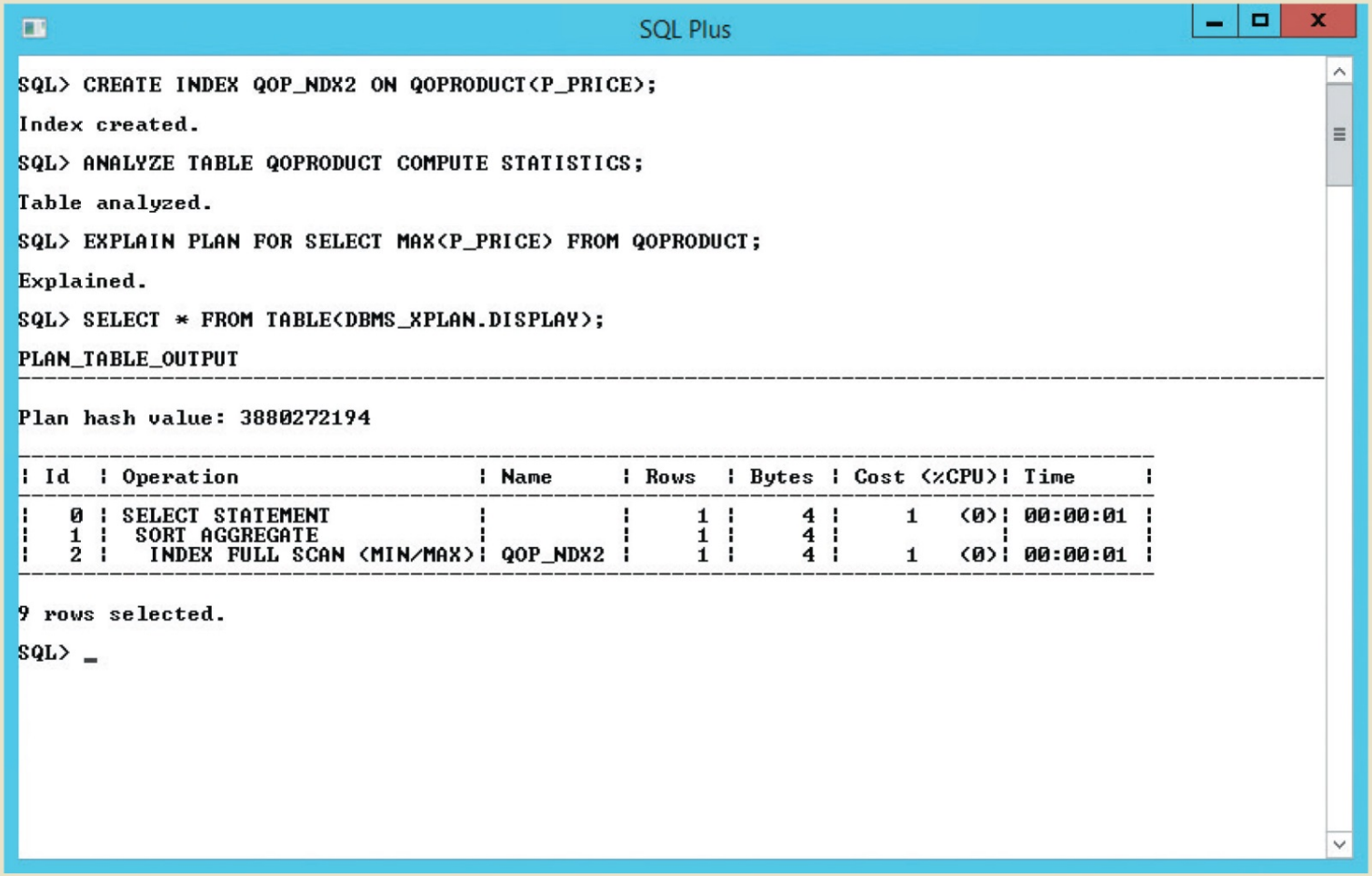


FIGURE 11.12 ORACLE TOOLS FOR QUERY OPTIMIZATION

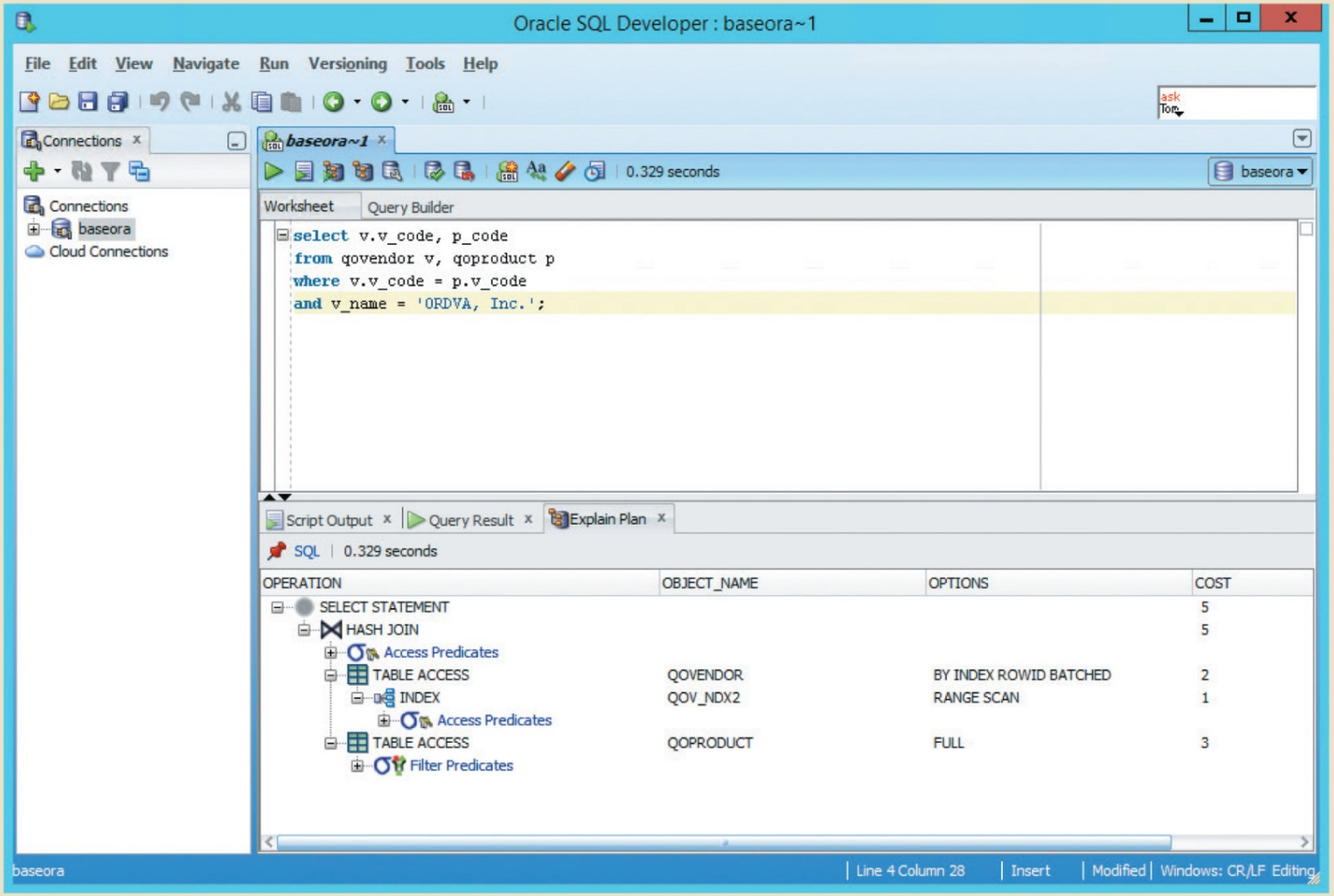


FIGURE 11.13 MYSQL TOOLS FOR QUERY OPTIMIZATION

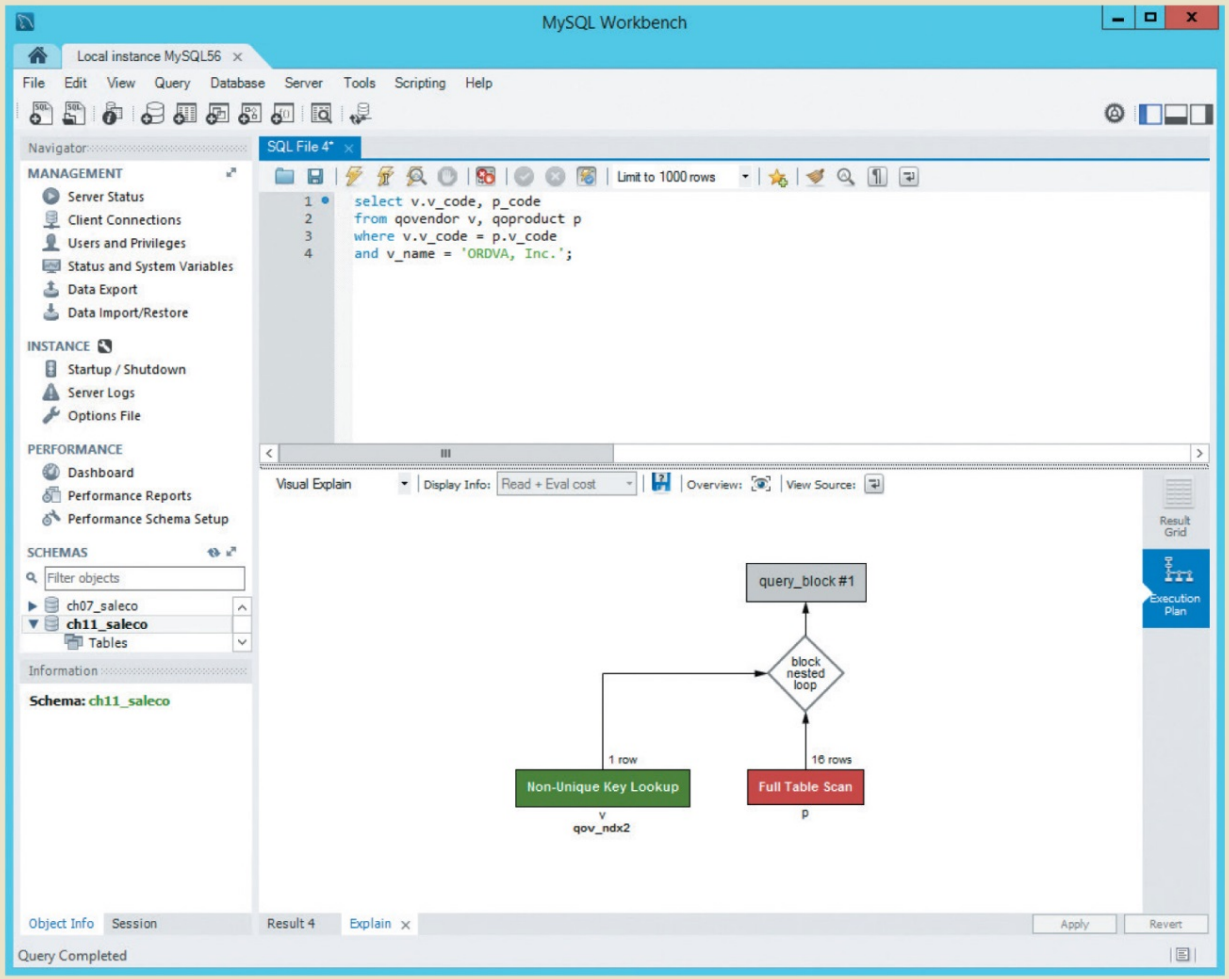


FIGURE 11.14 MICROSOFT SQL SERVER TOOLS FOR QUERY OPTIMIZATION