Chapter 11

Database Performance Tuning and Query Optimization

Note : This chapter slides prepared from the Cengage book of Coronel and Morris 13th Edition and other Internet resources and some material from Pearson Textbook Chapter -19

Topics of Discussion

A. Processes involved in database performance tuning
B. DBMS process of SQL queries in each of 3 phases
C. Role of indexes in speeding up data access
D. Rule-based optimizer and Cost-based optimizer
E. Common practices used to write efficient SQL code
F. Formulate queries and tune the DBMS for optimal performance

Database Performance-Tuning Concepts (1 of 4)

- One of the main functions of a database system is to provide timely answers to end-users.
  - End users interact with the DBMS through the use of queries to generate information, using the following sequence:
    - **1. End-user (client-end) application generates a query**
    - **2. Query is sent to the DBMS (server end)**
    - **3. DBMS (server end) executes the query**
    - **4. DBMS sends the resulting data set to the end-user (client-end) application**
- Goal of database performance is to execute queries as fast as possible
  - Unfortunately, the same query might perform well one day and not so well two months later.
  - Database performance tuning: set of activities and procedures that reduce response time of database system
  - Fine-tuning the performance of a system requires a holistic approach
  - All factors must operate at optimum level with minimal bottlenecks

Database Performance-Tuning Concepts (2 of 4)

Table 11.1 General Guidelines for Better System Performance

	System Resources	Client	Server
Hardware	CPU	The fastest possible Dual-core CPU or higher "Virtualized Client desktop technologies could also be used"	The fastest possible Multiple processors (quad-core technology or higher) Cluster of networked computers "Virtualized server technology could be used"
	RAM	The maximum possible to avoid OS memory to disk swapping	The maximum possible to avoid OS memory to disk swapping
	Storage	Fast SATA/EIDE hard disk with sufficient free hard disk space. solid state drives (SSDs) 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 (SSDs) for faster speed Separate disks for OS, DBMS, and data spaces
	Network	High-speed connection	High-speed connection

Database Performance-Tuning Concepts (3 of 4)

Table 11.1 General Guidelines for Better System Performance (continued..)

	System Resources	Client	Server
Software	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

Database Performance-Tuning Concepts (4 of 4)

- In general, database performance-tuning activities can be divided into those on the client side and those on the server side.
- Client side: SQL performance tuning
  - Generate 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 is configured to respond to clients' requests as fast as possible
  - While making optimum use of existing resources
- DBMS implementations are typically more complex than just a two-tier client/server configuration.
- The network component plays a critical role in delivering messages between clients and servers; this is especially important in distributed databases.

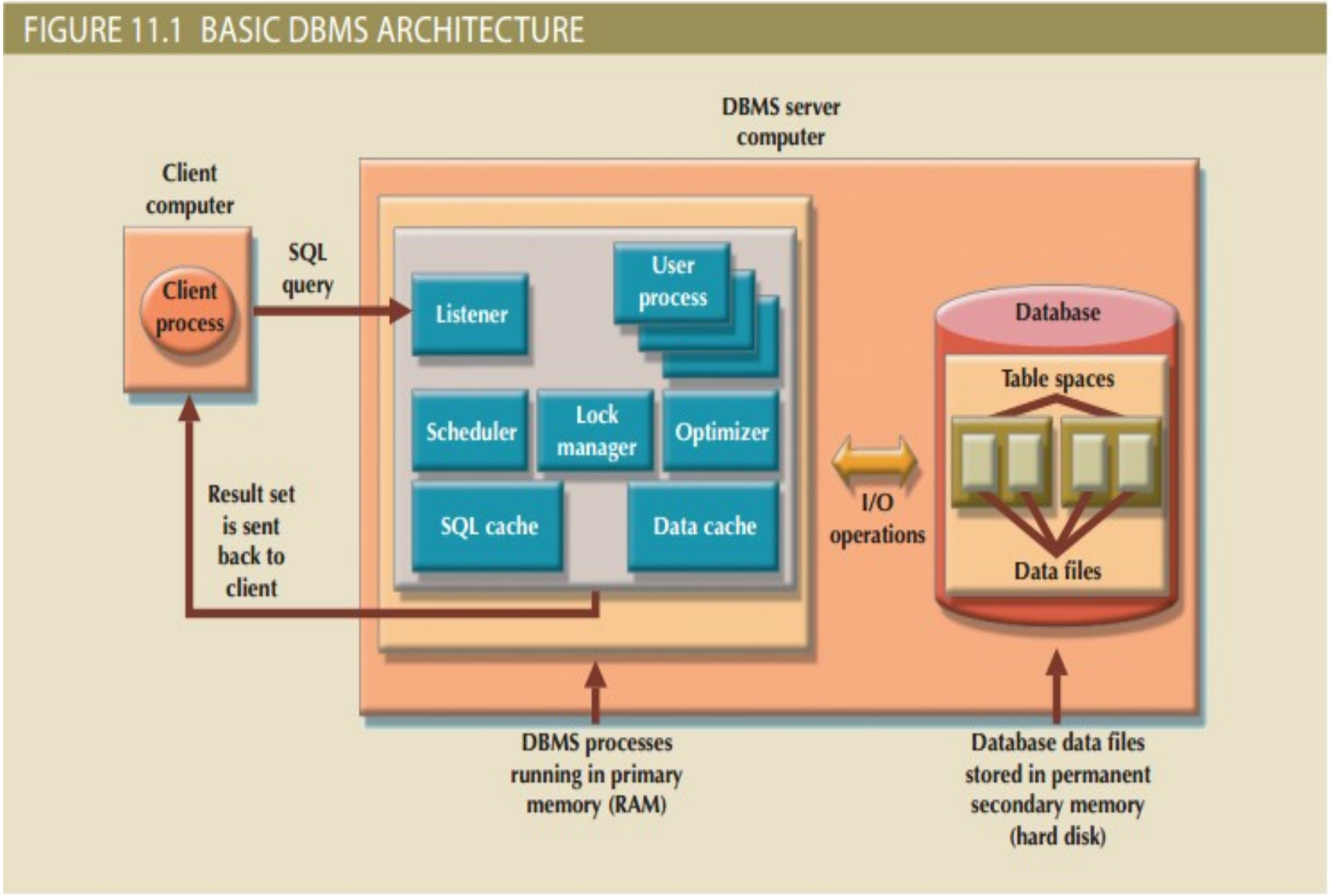
DBMS Architecture (1 of 3)

- Figure 11.1 illustrates some typical DBMS processes. Although the number of processes and their names vary from vendor to vendor, the functionality is similar.

  - Listener: The listener process listens for clients' requests and handles the processing of the SQL requests to other DBMS processes.
  - User: The DBMS creates a user process to manage each client session. Therefore, when you log on to the DBMS, you are assigned a user process.
  - Scheduler: The scheduler process organizes the concurrent execution of SQL requests.
  - Lock manager: This process manages all locks placed on database objects, including disk pages.
  - Optimizer: The optimizer process analyzes SQL queries and finds the most efficient way to access the data.

DBMS Architecture (2 of 3)

FIGURE 11.1 BASIC DBMS ARCHITECTURE



DBMS Architecture (3 of 3)

- 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
  - Logical grouping of several data files that store data with similar characteristics known as table space
- Data cache or buffer cache: shared, reserved memory area
  - Stores most recently accessed data blocks in RAM
- SQL cache or procedure cache: shared, reserved memory
  - Stores most recently executed SQL statements or PL/SQL procedures
- DBMS retrieves data from permanent storage and places them in RAM
  - Data is retrieved from the data files and placed in the data cache
  - 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

Database Query Optimization Modes (1 of 3)

- Most Algorithms proposed for query optimization are based on the selection of:
    1. Optimum order to achieve the fastest execution time
    2. Sites to be accessed to minimize communication costs
- Within that 2 principles algorithms evaluated based on:
    - Operation modes
    - 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 optimization be selected and scheduled by the end-user or programmer

Database Query Optimization Modes (2 of 3)

- Classification based on the timing of optimization
- Static query optimization:
  - Takes place at compilation time
  - Best optimization strategy is selected when the query is compiled by the DBMS.
- 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

Database Query Optimization Modes (3 of 3)

- Finally, Classification can be based on type of information used to optimize the query.
  - -Statistically based query optimization algorithm
  - -Rule-based query optimization algorithm

- 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

Database Statistics (1 of 2)

- Database statistics means measurements about database objects; that provides a snapshot of database characteristics.
  - Number of processors used
  - Processor speed
  - Temporary space available

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

Database Statistics (2 of 2)

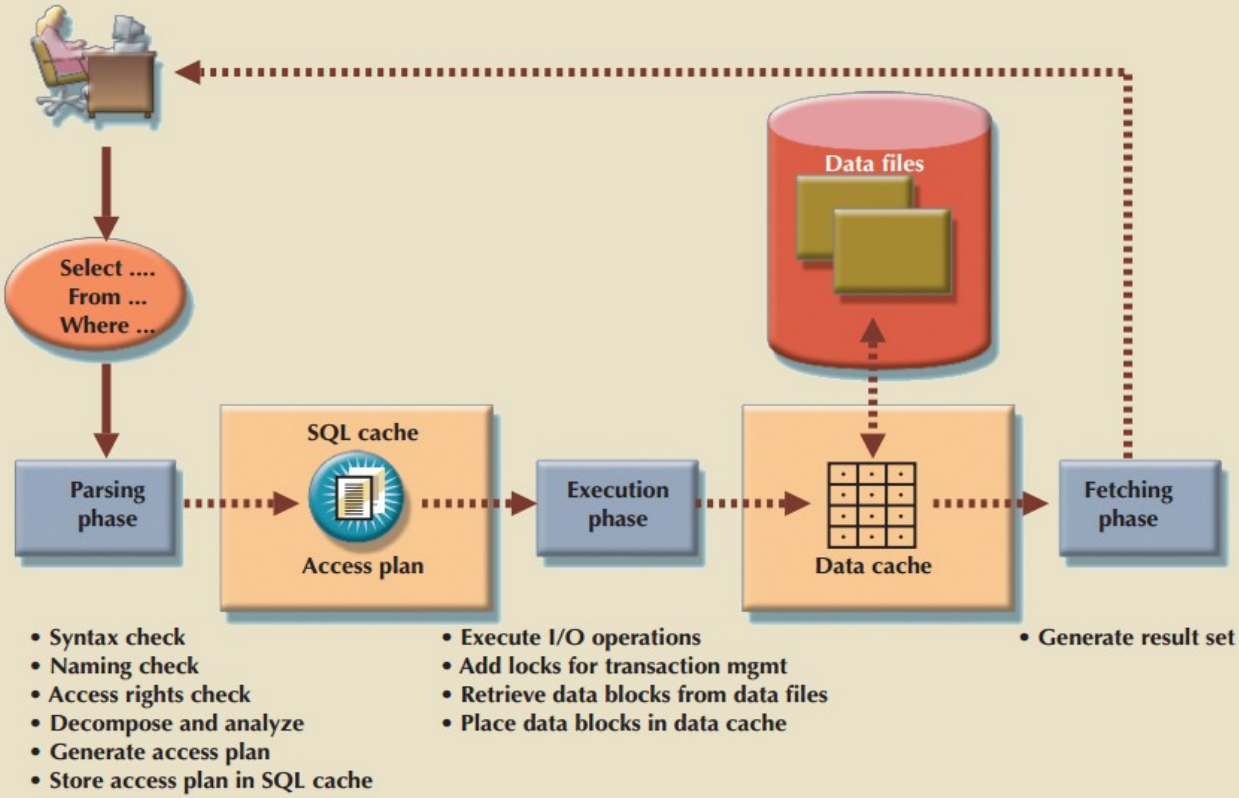
- Database statistics can be gathered manually by the DBA or automatically by the DBMS. For example, many DBMS vendors support the ANALYZE command in SQL to gather statistics.
- In Oracle, use ANALYZE TABLE/INDEX <object_name> COMPUTE STATISTICS;
- In MySQL, use ANALYZE TABLE <object_name> ;
- In SQL Server, use UPDATE STATISTICS <object_name>; , where object name refers to a table or a view.
- For example, to generate statistics for the EMPLOYEE table:
  - In Oracle: ANALYZE TABLE EMPLOYEE COMPUTE STATISTICS;
  - In MySQL: ANALYZE TABLE EMPLOYEE;
  - In SQL Server: UPDATE STATISTICS EMPLOYEE;
- When you generate statistics for a table, all related indexes are also analyzed. However, you could generate statistics for a single index by using the following command in SQL Server. where EMP_NDX is the name of the index.
  - UPDATE STATISTICS EMPLOYEE EMP_NDX;.

Query Processing (1 of 4)

What happen at the DBMS server when the client`s SQL statement is received?

1. **Parsing**: DBMS parses the SQL query and chooses the most efficient access/execution plan
2. **Execution**: DBMS executes the SQL query using the chosen execution plan
3. **Fetching**: DBMS fetches the data and sends the result back to the client

FIGURE 11.2 QUERY PROCESSING



Query Processing (2 of 4)

- **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.
  - Finds most efficient way to access data
- Parsing a SQL query requires several steps, in which the SQL query is:
    - Validated for syntax compliance
    - Validated against the data dictionary to ensure that table names and column names are correct
    - Validated against the data dictionary to ensure that the user has proper access rights
    - Analyzed and decomposed into more atomic components
    - Optimized through transformation into a fully equivalent but more efficient SQL query
    - Prepared for execution by determining the most efficient execution or access plan

Query Processing (3 of 4)

- Access plans: result of parsing a SQL statement, contains a series of steps the DBMS will use to execute the query and return the result set in the most efficient way
  - Access plan exists for query in SQL cache: DBMS reuses it
  - No access plan: optimizer evaluates various plans and chooses one to be placed in SQL cache for use

Table 11.3 Sample DBMS Access Plan I/O Operations

Operation	Description
Table scan (full)	Reads the entire table sequentially, from the first row to the last, one row at a time (slowest)
Table access (row ID)	Reads a table row directly, using the row ID value (fastest)
Index scan (range)	Reads the index first to obtain the row IDs and then accesses the table rows directly (faster than a full table scan)
Index access (unique)	Used when a table has a unique index in a column
Nested loop	Reads and compares a set of values to another set of values, using a nested loop style (slow)
Merge	Merges two data sets (slow)
Sort	Sorts a data set (slow)

Query Processing (4 of 4)

- **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
- **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

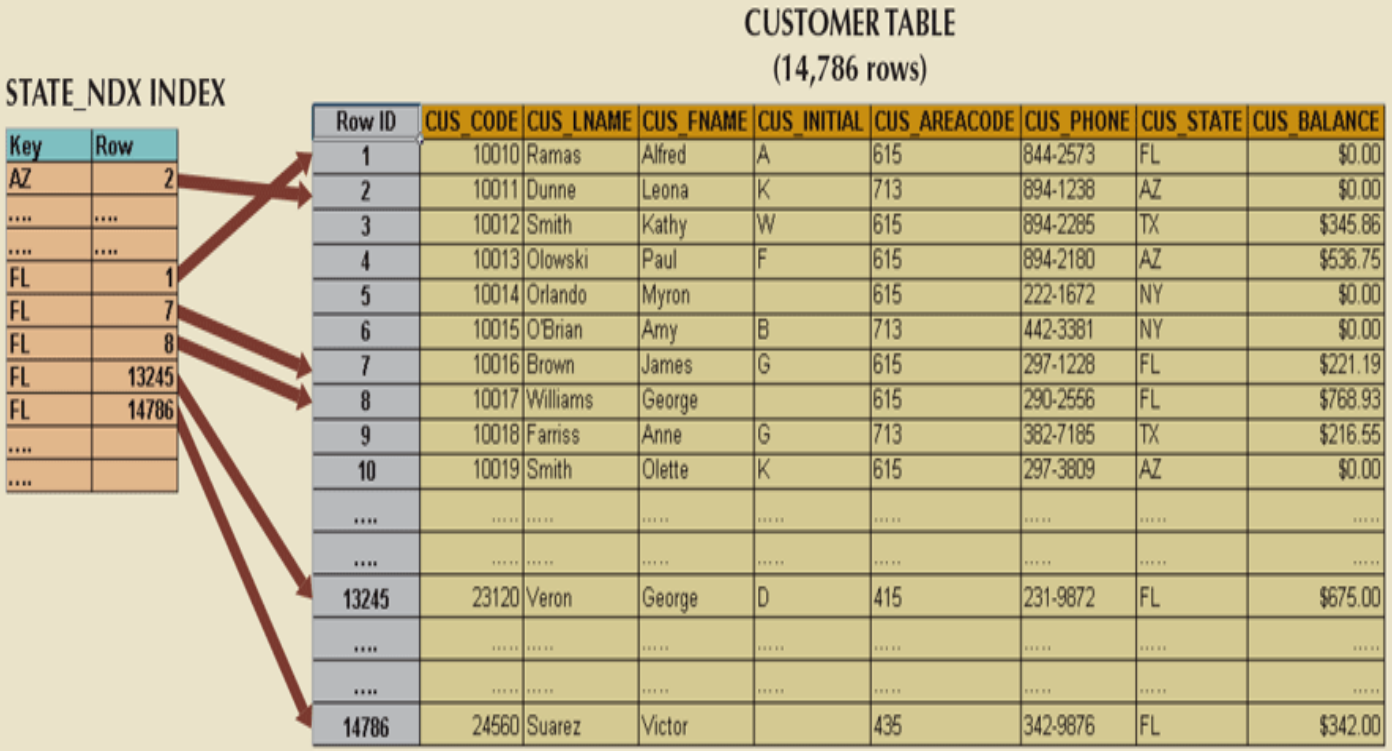
Query Processing Bottlenecks: Delay introduced in the processing of an I/O operation that slows the system. Caused by the: CPU, RAM, Hard disk, Network, Application code

Indexes and Query Optimization (1 of 6)

- **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
  - The pointers are the row IDs for the actual table rows. Conceptually, a data index is similar to a book index.
  - More efficient than a full table scan;
  - Index data is preordered and the amount of data is usually much smaller.
  - When performing searches, it is almost always better for the DBMS to use the index to access a table than to scan all rows in a table sequentially.

Indexes and Query Optimization (2 of 6)

FIGURE 11.3 INDEX REPRESENTATION FOR THE CUSTOMER TABLE



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

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

- For example, Figure 11.3 shows the index representation of a CUSTOMER table with 14,786 rows and the index STATE_NDX on the CUS_STATE.
- Suppose you submit the following query:

```
SELECT CUS_NAME, CUS_STATE FROM CUSTOMER WHERE CUS_STATE = 'FL'
```

Indexes and Query Optimization (3 of 6)

- If there is no index, the DBMS will perform a full-table scan and read all 14,786 customer rows.
- Assuming that the index STATE_NDX is created (and analyzed), the DBMS will automatically use the index to locate the first customer with a state equal to 'FL' and then proceed to read all subsequent CUSTOMER rows, using the row IDs in the index as a guide.
- Assuming that only 5 rows meet the condition CUS_STATE = 'FL' there are five accesses to the index and five accesses to the data, for a total of 10 I/O accesses.
- The DBMS would be saved from reading approximately 14,776 (14786 -10), I/O requests for customer rows that do not meet the criteria. That is a lot of CPU cycles!

Indexes and Query Optimization (4 of 6)

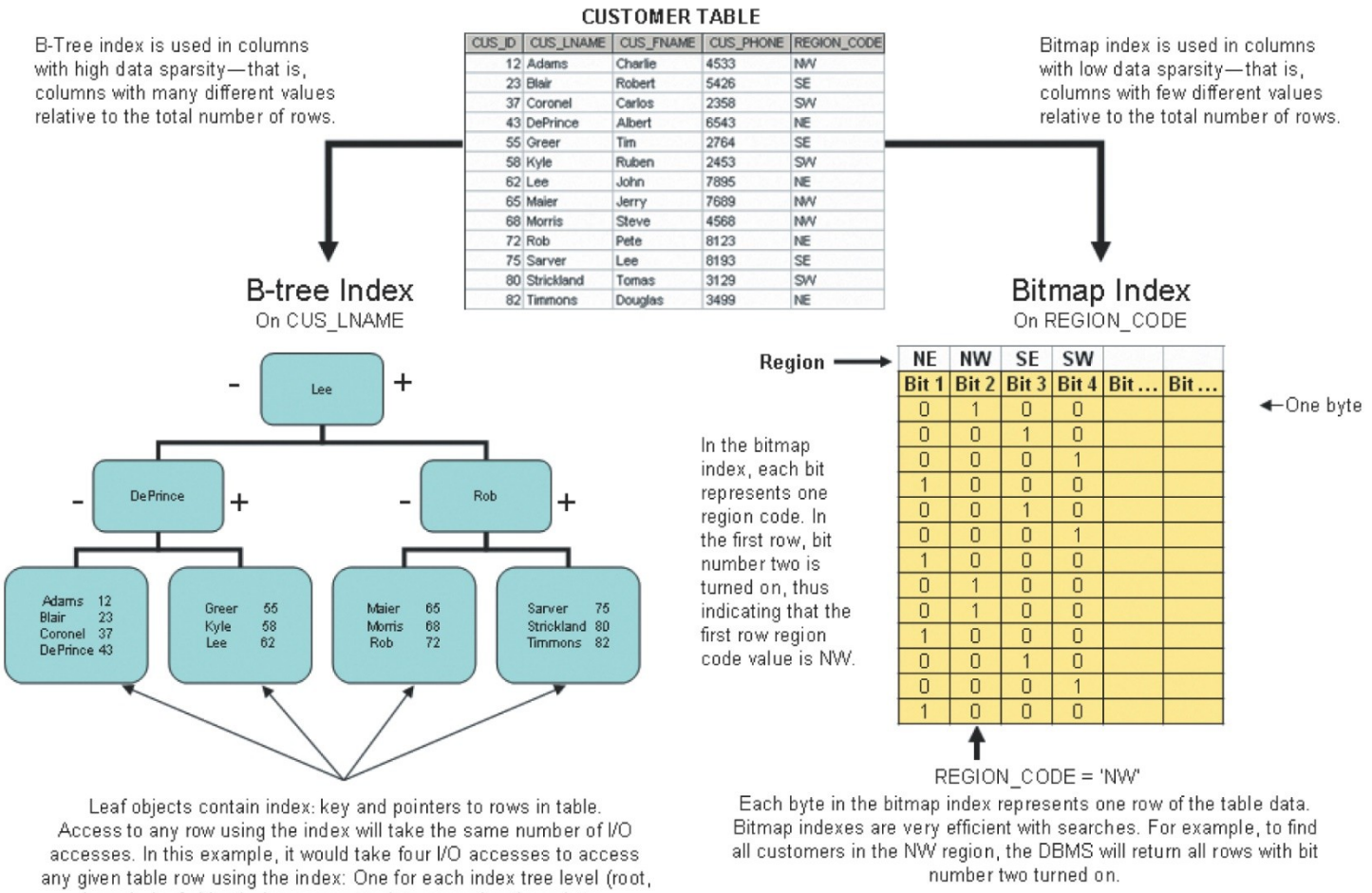
- **Data sparsity** : determines the need for an index is the data sparsity of the column you want to index.
- Data sparsity refers to different values a column could have.
- For example, a STU_SEX column can have only 2 values, M or F; thus column have low sparsity. In contrast, the STU_DOB column stores different date values; have high sparsity.
- Knowing the sparsity helps you decide whether the use of an index is appropriate.
- **Data Structure** : Most DBMSs implement indexes using the following data structures:
  - Hash index: A hash index is based on an ordered list of hash values. A hash algorithm is used to create a hash value from a key column.
  - This type of index is good for simple and fast lookup operations based on equality conditions.
    - For example, LNAME="Scott" and FNAME="Shannon".

Indexes and Query Optimization (5 of 6)

- B-tree index. The B-tree index is an ordered data structure organized as an upside-down tree. (See Figure 11.4.)
- The index tree is stored separately from the data. The lower-level leaves of the B-tree index contain the pointers to the actual data rows. B-tree indexes are “self-balanced,” which means that it takes approximately the same amount of time to access any given row in the index. This is the default and most common type of index used in databases.
- Bitmap index. A bitmap index uses a bit array (0s and 1s) to represent the existence of a value or condition.
- These indexes are used mostly in data warehouse applications in tables with a large number of rows in which a small number of column values repeat many times.
- Bitmap indexes tend to use less space than B-tree indexes because they use bits instead of bytes to store their data.

Indexes and Query Optimization (6 of 6)

FIGURE 11.4 B-TREE AND BITMAP INDEX REPRESENTATION



Optimizer Choices (1 of 4)

- Query optimization is the central activity in the parsing phase, during that, the DBMS must choose what indexes to use, how to perform join operations, which table to use first, and so on.
- **Rule-based optimizer:** uses preset rules and points to determine the best approach to execute a query.
  - The rules assign a "fixed cost" to each SQL operation; the costs are then added to yield the cost of the execution plan.
  - For example, a full table scan has a set cost of 10, while a table access by row ID has a set cost of 3.
- **Cost-based optimizer:** uses sophisticated algorithms based on statistics about the objects being accessed to determine the best approach to execute a query.
  - Optimizer process adds up the processing cost, I/O costs, and resource costs (RAM and temporary space) to determine the total cost of a given execution plan

Optimizer Choices (2 of 4)

- **Example**: To understand the function of the query optimizer, consider a simple example.
- Assume that you want to list all products provided by a vendor based in Florida. To acquire that information, you could write the following query:

```
SELECT  P_CODE, P_DESCRIPT, P_PRICE, V_NAME, V_STATE
FROM    PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE
        AND VENDOR.V_STATE = 'FL';
```

- Furthermore, assume that the database statistics indicate the following:
  - The PRODUCT table has 7,000 rows.
  - The VENDOR table has 300 rows.
  - 10 vendors are located in Florida.
  - 1000 products come from vendors in Florida.

Optimizer Choices (3 of 4)

- Table 11.4 shows two sample access plans for the previous query and their respective I/O costs.

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**

Optimizer Choices (4 of 4)

- **Optimizer hints**: special instructions for the optimizer
- Embedded in the SQL command text

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 (1 of 4)

- 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
- A carefully written query almost always outperforms a poorly written one
- This section are related to the use of the SELECT statement, and in particular, the use of indexes and how to write conditional expressions.
  1. Index Selectivity
  2. Conditional expression

SQL Performance Tuning (2 of 4)

- **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
- Cannot always be used to improve performance.
- As a general rule, indexes are likely to be used:
  - When an indexed column appears by itself in the search criteria of a WHERE or HAVING clause
  - When an indexed column appears by itself in a GROUP BY or ORDER BY clause
  - When a MAX or MIN function is applied to an indexed column
  - When the data sparsity on the indexed column is high

SQL Performance Tuning (2 of 4)

- Here are some general guidelines for creating and using indexes:
  - Create indexes for each single attribute used in a WHERE, HAVING, ORDER BY, or GROUP BY clause.
  - Do not use indexes in small tables or tables with low sparsity.
  - Declare primary and foreign keys so the optimizer can use the indexes in join operations.
  - Declare indexes in join columns other than PK or FK.
- Function-based index: You cannot always use an index to improve performance. The reason is that in some DBMSs, indexes are ignored when you use functions in the table attributes.
- A function-based index is an index based on a specific SQL function or expression.
  - For example, you could create an index on YEAR(INV_DATE).
- Function-based indexes are especially useful when dealing with derived attributes.
  - For example, create an index on EMP_SALARY + EMP_COMMISSION.

SQL Performance Tuning (3 of 4)

- **Conditional Expression**: are Expressed within WHERE or HAVING clauses of a SQL statement
- Restricts the output of a query to only rows matching 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
  - 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

SQL Performance Tuning (4 of 4)

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

In Table 11.6, note that an operand can be:

- A simple column name such as P_PRICE or V_STATE
- A literal or a constant such as the value 10.00 or the text 'FL'
- An expression such as P_MIN * 1.1

DBMS Performance Tuning (1 of 3)

- DBMS performance tuning includes global tasks such as managing the DBMS processes in primary memory (allocating memory for caching purposes) and managing the structures in physical storage (allocating space for the data files).
- DBMS performance tuning at server end focuses on setting parameters used for:
  - Data cache: he data cache size must be set large enough to permit as many data requests as possible to be serviced from the cache.
  - SQL cache: he SQL cache stores the most recently executed SQL statements. (The same query will likely be submitted by many different users. In such cases, DBMS will parse the query only once and execute it many times, using the same access plan
  - Sort cache: he sort cache is used as a temporary storage area for ORDER BY or GROUP BY operations
  - Optimizer mode: Most DBMSs operate in one of two optimization modes:

DBMS Performance Tuning (2 of 3)

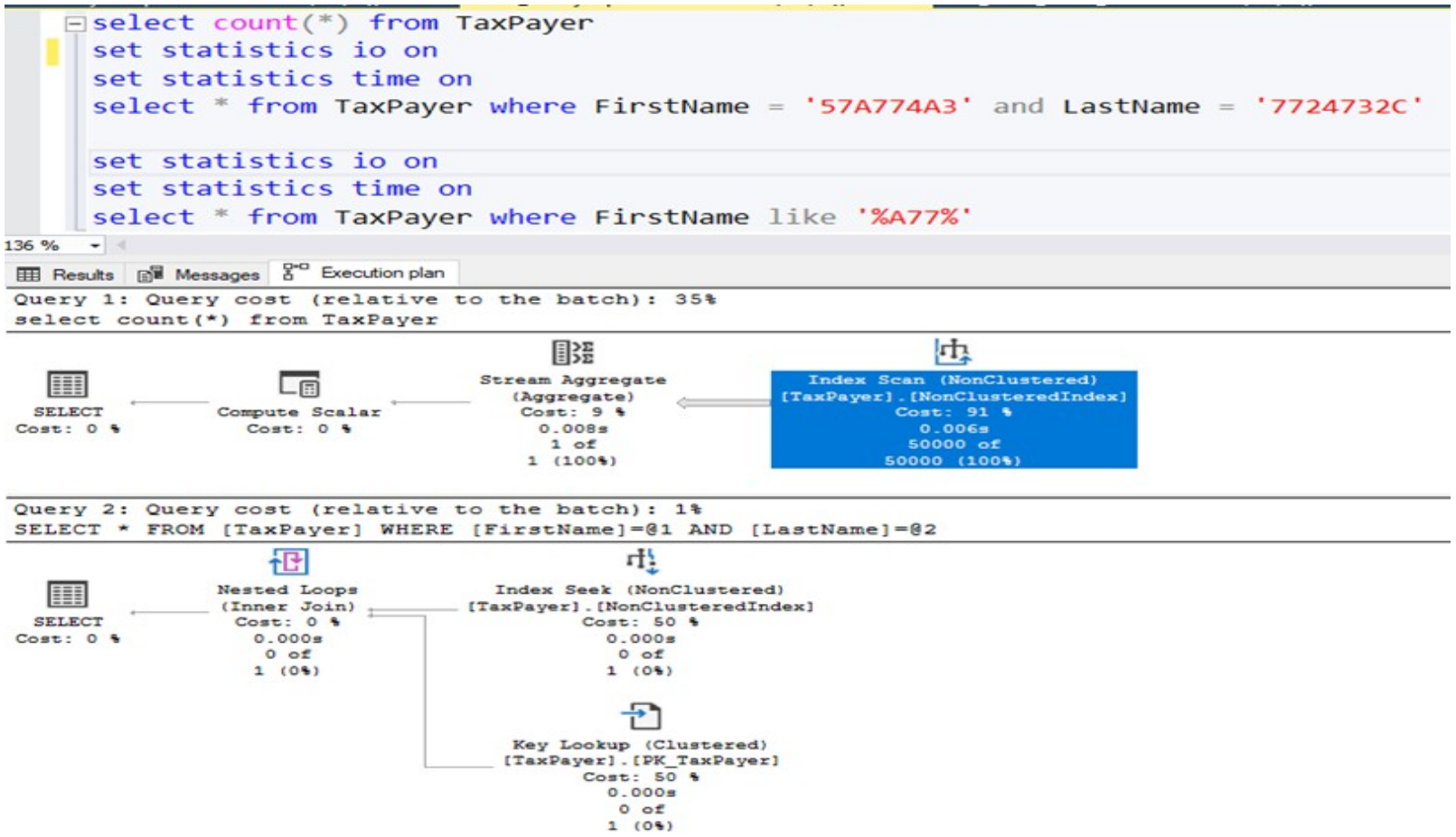
- In-memory database: From the performance point of view, it would be optimal to have the entire database stored in primary memory to minimize costly disk access. This is why several database vendors offer in-memory database options for their main products.
- Store large portions of the database in primary storage
    - These systems are becoming popular
    - Increasing performance demands of modern database applications
    - Diminishing costs
    - Technology advances of components

DBMS Performance Tuning (3 of 3)

- Recommendations for physical storage of databases
  - Utilize I/O accelerators
  - 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
  - Take advantage of the various table storage organizations in the database
  - Partition tables based on usage
  - Apply denormalized tables where appropriate
  - Store computed and aggregate attributes in tables

Lab Demonstration:

- Query optimization and performance Tuning Lab available at your myCourse using MS SQL Server 2017.
- File: "*Tax_Database_lab for Query optimization*"



Indexes of MS SQL Server 2017

- Clustered indexes are stored physically on the table. This means they are the fastest.
- There can be only one clustered index for a table.
- Usually made on the primary key.

- All other indexes must be non-clustered.
- A non-clustered index has a duplicate of the data from the indexed columns kept ordered together with pointers to the actual data rows (pointers to the clustered index if there is one).
- There can be only 249 non-clustered indexes for a table(can support upto 999)
- Can be made on any key.

Summary

- Database performance tuning refers to a set of activities and procedures designed to ensure that an end-user query is processed by the DBMS in the least amount of time
- Database statistics refer to a number of measurements gathered by the DBMS that describe a snapshot of the database objects' characteristics
- DBMSs process queries in three phases: parsing, execution, and fetching
- Indexes are crucial in the process that speeds up data access
- During query optimization, the DBMS must choose what indexes to use, how to perform join operations, which table to use first, and so on
- A rule-based optimizer uses preset rules and points to determine the best approach to execute a query
- SQL performance tuning deals with writing queries that make good use of the statistics
- Query formulation deals with how to translate business questions into specific SQL code to generate the required results
- DBMS performance tuning includes tasks such as managing the DBMS processes in primary memory (allocating memory for caching purposes) and managing the structures in physical storage (allocating space for the data files)

Chapter end questions

- 11.1. What is SQL performance tuning?
- 11.2. What is database performance tuning?
- 11.4. What are database statistics, and why are they important? How are database statistics obtained?
- 11.8. In simple terms, the DBMS processes a query in three phases. What are the phases, and what is accomplished in each phase?
- 11.9. If indexes are so important, why not index every column in every table? (Include a brief discussion of the role played by data sparsity.)
- 11.10. What is the difference between a rule-based optimizer and a cost-based optimizer?