Performance Tuning | Meerza Ahmed

Tuesday, October 17, 2023 1:00 PM

Lecture 9: Link | Syllabus
**Class Week 9/15*

Database Systems by Coronel & Morris

Chapter 11: Database Performance Tuning and Query Optimization

* 11.1 *Database Performance-Tuning Concepts*

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. The end-user (client-end) application generates a query.
2. The query is sent to the DBMS (server end).
3. The DBMS (server end) executes the query.
4. The DBMS sends the resulting data set to the end-user (client-end) application

* End users expect their queries to return results as quickly as possible. How do you know that the performance of a database is good? Regardless of end-user perceptions, *the goal of database performance is to execute queries as fast as possible*. Therefore, database performance must be closely monitored and regularly tuned. Database performance tuning refers to a set of activities and procedures designed to reduce the response time of the database system—that is, to ensure that an end-user query is processed by the DBMS in the minimum amount of time.

* 11.1-a *Performance Tuning: Client and Server*

In general, database performance-tuning activities can be divided into those on the client side and those on the server side.

* On the client side, the objective is to generate a SQL query that returns the correct answer in the least amount of time, using the minimum amount of resources at the server end. The activities required to achieve that goal are commonly referred to as SQL performance tuning.

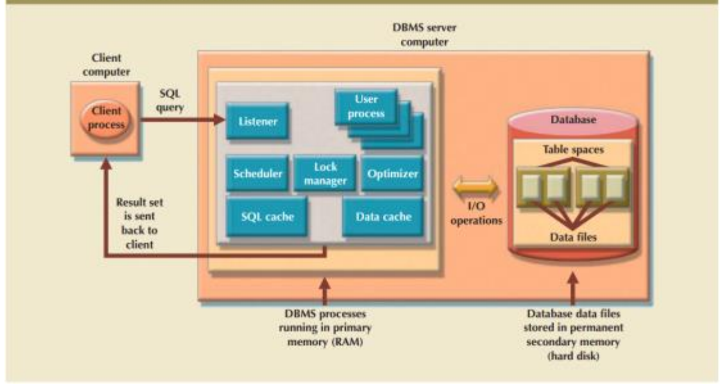
* On the server side, the DBMS environment must be properly configured to respond to clients' requests in the fastest way possible, while making optimum use of existing resources. The activities required to achieve that goal are commonly referred to as DBMS performance tuning.

This chapter covers SQL performance-tuning practices on the client side and DBMS performance-tuning practices on the server side.

* 11.1-b *DBMS Architecture*

The architecture of a DBMS is represented by the processes and structures (in memory and permanent storage) used to manage a database. Such processes collaborate with one another to perform specific functions.

FIGURE 11.1 BASIC DBMS ARCHITECTURE



* Note the following components and functions in Figure 11.1 :

- All data in a database is stored in data files. A data file can contain rows from a single table, or it can contain rows from many different tables. A database administrator (DBA) determines the initial size of the data files that make up the database; however, the data files can automatically expand as required in predefined increments known as extents.
- Data files are generally grouped in file groups or table spaces. A table space or file group is a logical grouping of several data files that store data with similar characteristics.
- The data cache, or buffer cache, is a shared, reserved memory area that stores the most recently accessed data blocks in RAM
- The SQL cache, or procedure cache, is a shared, reserved memory area that stores the most recently executed SQL statements or PL/SQL procedures, including triggers and functions.

* 11.1-c *Database Query Optimization Modes*

Most of the algorithms proposed for query optimization are based on two principles :

- The selection of the optimum execution order to achieve the fastest execution time
- The selection of sites to be accessed to minimize communication costs

Within those two principles, a query optimization algorithm can be evaluated on the basis of its *operation mode* or the *timing of its optimization*.

* Operation modes can be classified as manual or automatic. Automatic query optimization means that the 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 dev.

* Query optimization algorithms can also be classified according to when the optimization is done. Within this timing classification, query optimization algorithms can be static or dynamic.

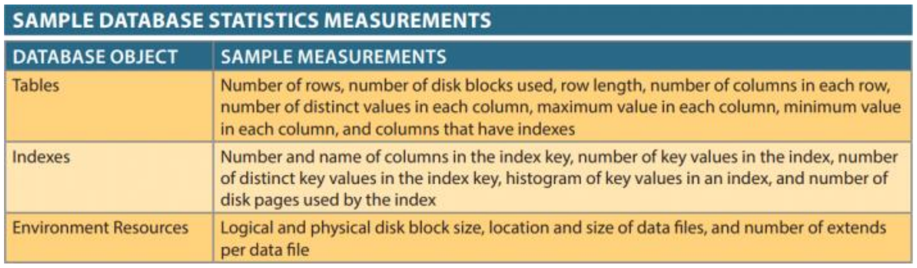
- Static query optimization takes place at compilation time. In other words, the best optimization strategy is selected when the query is compiled by the DBMS. This approach is common when SQL statements are embedded in procedural programming languages such as C# or Visual Basic .NET.
- Dynamic query optimization takes place at execution time. Database access strategy is defined when the program is executed. Therefore, access strategy is dynamically determined by the DBMS at run time, using the most up-to-date information about the database.

* Finally, query optimization techniques can be classified according to the type of information that is used to optimize the query. For example, queries may be based on statistically based or rule-based algorithms.

- A statistically based query optimization algorithm uses statistical information about the database. The statistics provide information about database characteristics such as size, number of records, average access time, number of requests serviced, and number of users with access rights.
- The statistical information is managed by the DBMS and is generated in one of two different modes: dynamic or manual. In the dynamic statistical generation mode, the DBMS automatically evaluates and updates the statistics after each data access operation. In the manual statistical generation mode, the statistics must be updated periodically through a user-selected utility such as IBM's RUNSTAT command, which is used by DB2 DBMSs.
- A rule-based query optimization algorithm is based on a set of user-defined rules to determine the best query access strategy. The rules are entered by the end user or database administrator, and they are typically general in nature.

*** 11.1-d** *Database Statistics*

Another DBMS process that plays an important role in query optimization is gathering database statistics. The term database statistics refers to a number of measurements about database objects, such as number of processors used, processor speed, and temporary space available. Such statistics provide a snapshot of database characteristics.



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

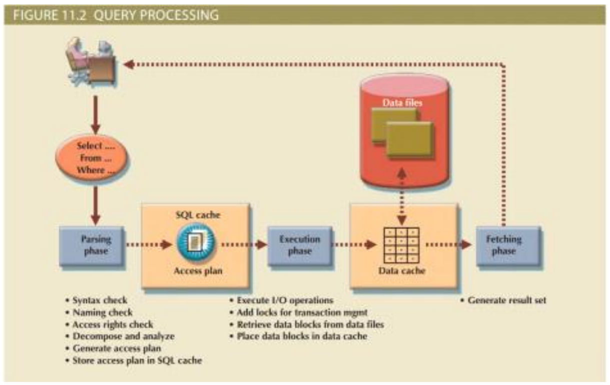
*** 11.2** *Query Processing*

What happens at the DBMS server end when the client's SQL statement is received?

In simple terms, the DBMS processes a query in three phases:

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

FIGURE 11.2 QUERY PROCESSING



*** 11.2-a** *SQL Parsing Phase*

The SQL parsing activities are performed by the query optimizer, which analyzes the SQL query and finds the most efficient way to access the data. This process is the most time-consuming phase in query processing.

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

Once the SQL statement is transformed, the DBMS creates what is commonly known as an access plan or execution plan. An access plan is the result of parsing SQL statement; it contains the series of steps a DBMS will use to execute the query and return the result set in the most efficient way

*** 11.2-b** *SQL Execution Phase*

In this phase, all I/O operations indicated in the access plan are executed. When the execution plan is run, the proper locks—if needed—are acquired for the data to be accessed, and the data is retrieved from the data files and placed in the DBMS's data cache. All transition management commands are processed during the parsing and execution phases of query processing

*** 11.2-c** *SQL Fetching Phase*

After the parsing and execution phases are completed, all rows that match the specified condition(s) are retrieved, sorted, grouped, and aggregated (if required). During the fetching phase, the rows of the resulting query result set are returned to the client. The DBMS might use temporary table space to store temporary data. In this stage, the database server coordinates the movement of the result set row from the server cache to the client cache. For example, a given query result set might contain 9,000 rows; the server would send the first 100 rows to the client and then wait for the client to request the next set of rows, until the entire result set is sent to the client.

*** 11.2-d** *Query Processing Bottlenecks*

As you have seen, the execution of a query requires the DBMS to break down the query into a series of interdependent I/O operations to be executed in a collaborative manner. The more complex a query is, the more complex the operations are, which means that bottlenecks are more likely. A query processing bottleneck is a delay introduced in the processing of an I/O operation that causes the overall system to slow down. In the same way, the more components a system has, the more interfacing is required among the components, increasing the likelihood of bottlenecks. Within a DBMS, five components typically cause bottlenecks : *CPU, RAM, Hard disk, Network, and Application code*.

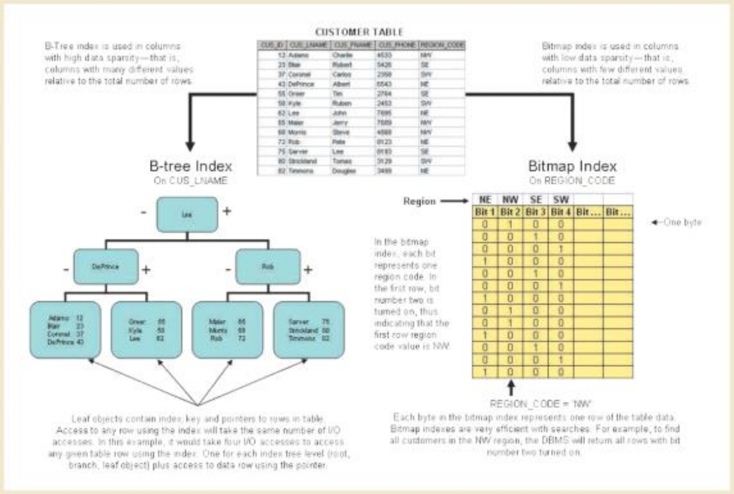
*** 11.3** *Indexes and Query Optimization*

Indexes are crucial in speeding up data access because they facilitate searching, sorting, and using aggregate functions and even join operations. The improvement in data access speed occurs because an index is an ordered set of values that contains 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. When you use a book index, you look up a word, which is similar to the index key

Most DBMSs implement indexes using one of 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 value points to an entry in a hash table, which in turn points to the actual location of the data row. This type of index is good for simple and fast lookup operations based on equality conditions—for example, LNAME="Scott" and FNAME="Shannon
- 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. The B-tree index is used mainly in tables in which column values repeat a relatively small number of times.
- 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. (See Figure 11.4.) Bitmap indexes tend to use less space than B-tree indexes because they use bits instead of bytes to store their data.

FIGURE 11.4 B-TREE AND BITMAP INDEX REPRESENTATION



CUSTOMER TABLE

CUS_ID	CUS_LNAME	CUS_FNAME	CUS_PHONE	REGION_CODE
12	Adams	Charlie	4533	NW
23	Blair	Robert	5426	SE
37	Coronel	Carlos	2358	SW
43	DePrince	Albert	6543	NE
55	Greer	Tim	2764	SE
58	Kyle	Ruben	2453	SW
62	Lee	John	7695	NE
65	Maier	Jerry	7689	NW
68	Morris	Steve	4588	NW
72	Rob	Pete	8123	NE
75	Server	Lee	8183	SE
80	Strickland	Tomas	3129	SW
82	Timmons	Douglas	3489	NE

B-Tree index is used in columns with high data sparsity—that is, columns with many different values relative to the total number of rows.

Bitmap index is used in columns with low data sparsity—that is, columns with few different values relative to the total number of rows.

NE	NW	SE	SW		
Bit 1	Bit 2	Bit 3	Bit 4	Bit ...	Bit ...
0	1	0	0		
0	0	1	0		
0	0	0	1		
1	0	0	0		
0	0	1	0		
0	0	0	1		
1	0	0	0		
0	1	0	0		
0	1	0	0		
1	0	0	0		
0	0	1	0		
0	0	0	1		
1	0	0	0		

In the bitmap index, each bit represents one region code. In the first row, bit number two is turned on, thus indicating that the first row region code value is NW.

Leaf objects contain index key and pointers to rows in table. Access to any row using the index will take the same number of I/O accesses. In this example, it would take four I/O accesses to access any given table row using the index: One for each index tree level (root, branch, leaf object) plus access to data row using the pointer.

Each byte in the bitmap index represents one row of the table data. Bitmap indexes are very efficient with searches. For example, to find all customers in the NW region, the DBMS will return all rows with bit number two turned on.

*** 11.4** *Optimizer Choices*

Query optimization is central during the parsing phase in query processing. In this phase, the DBMS must choose what indexes to use, how to perform join operations, which table to use first, and so on. Each DBMS has its own algorithms for deter mining the most efficient way to access the data.

The query optimizer can operate in one of two modes :

- A 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.
- A cost-based optimizer uses sophisticated algorithms based on statistics about the objects being accessed to determine the best approach to execute a query. In this case, the optimizer process adds up the processing cost, the I/O costs, and the resource costs (RAM and temporary space) to determine the total cost of a given execution plan.

*** 11.5** *SQL Performance Tuning*

SQL performance tuning is evaluated from the client perspective. Therefore, the goal is to illustrate common practices used to write efficient SQL code.

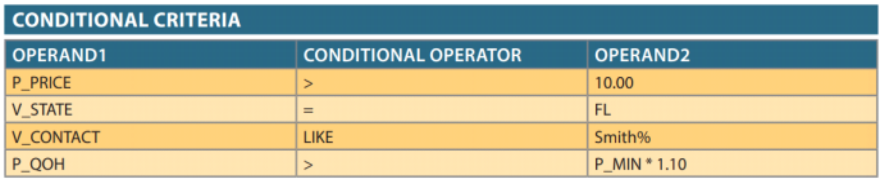
A few words of caution are appropriate :

- Most current-generation relational DBMSs perform automatic query optimization at the server end.
- Most SQL performance optimization techniques are DBMS-specific and, therefore, are rarely portable, even across different versions of the same DBMS. Part of the reason for this behavior is the constant advancement in database technologies.

* Although SQL data manipulation statements include many different commands such as INSERT, UPDATE, DELETE, and SELECT, most recommendations in this section are related to the use of the SELECT statement, and in particular, the use of indexes and how to write conditional expressions.

*** 11.5-b** *Conditional Expressions*

A conditional expression is normally placed within the WHERE or HAVING clauses of a SQL statement. Also known as conditional criteria, a conditional expression restricts the output of a query to only the rows that match the conditional criteria.



CONDITIONAL CRITERIA

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

* 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 * 1.10

* **11.7** *DBMS Performance Tuning*

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).

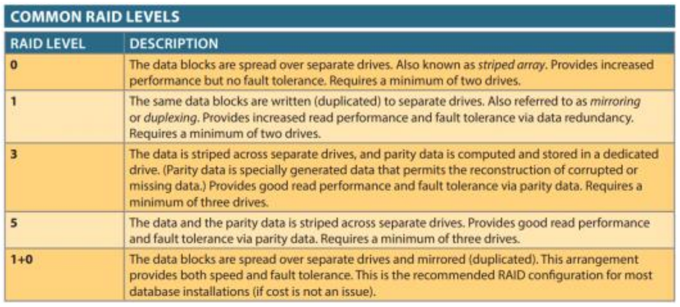
DBMS performance tuning at the server end focuses on setting the parameters used for :
- *Data cache*. The data cache size must be set large enough to permit as many data requests as possible to be serviced from the cache. Each DBMS has settings that control the size of the data cache; some DBMSs might require a restart. This cache is shared among all database users. The majority of primary memory resources will be allocated to the data cache

- *SQL cache*. The SQL cache stores the most recently executed SQL statements (after the SQL statements have been parsed by the optimizer). Generally, if you have an application with multiple users accessing a database, the same query will likely be submitted by many different users. In those cases, the DBMS will parse the query only once and execute it many times, using the same access plan. In that way, the second and subsequent SQL requests for the same query are served from the SQL cache, skipping the parsing phase.

- *Sort cache*. The sort cache is used as a temporary storage area for ORDER BY or GROUP BY operations, as well as for index-creation functions.

- Optimizer mode. Most DBMSs operate in one of two optimization modes: cost-based or rule-based. Others automatically determine the optimization mode based on whether database statistics are available. For example, the DBA is responsible for generating the database statistics that are used by the cost-based optimizer. If the statistics are not available, the DBMS uses a rule-based optimizer.

* For performance, 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. In-memory database systems are optimized to store large portions (if not all) of the database in primary (RAM) storage rather than secondary (disk) storage. These systems are becoming popular because increasing performance demands of modern database applications (such as Business Analytics and Big Data), diminishing costs, and technology advances of components (such as flash-memory and solid state drives.)



COMMON RAID LEVELS

RAID LEVEL	DESCRIPTION
0	The data blocks are spread over separate drives. Also known as *striped array*. Provides increased performance but no fault tolerance. Requires a minimum of two drives.
1	The same data blocks are written (duplicated) to separate drives. Also referred to as *mirroring* or *duplexing*. Provides increased read performance and fault tolerance via data redundancy. Requires a minimum of two drives.
3	The data is striped across separate drives, and parity data is computed and stored in a dedicated drive. (Parity data is specially generated data that permits the reconstruction of corrupted or missing data.) Provides good read performance and fault tolerance via parity data. Requires a minimum of three drives.
5	The data and the parity data is striped across separate drives. Provides good read performance and fault tolerance via parity data. Requires a minimum of three drives.
1+0	The data blocks are spread over separate drives and mirrored (duplicated). This arrangement provides both speed and fault tolerance. This is the recommended RAID configuration for most database installations (if cost is not an issue).

* Use RAID (Redundant Array of Independent Disks) to provide both performance improvement and fault tolerance, and a balance between them. Fault tolerance means that in case of failure, data can be reconstructed and retrieved. RAID systems use multiple disks to create virtual disks storage volumes) formed by several individual disks. This table describes the most common RAID configurations.