Session 20

Performance Optimization and Scalability

Objectives

- ❑ Describe performance optimization
- ❑ Describe query related functions
- ❑ Explain capability of MySQL for scaling and availability

Performance

Performance is a measure of time that is required to complete a task.

- ❑ It is the response time measured in terms of tasks and time and not by resources.
- ❑ A database server's performance is thus measured by the time it takes to process the query. The unit is the time taken per query.

Performance Optimization

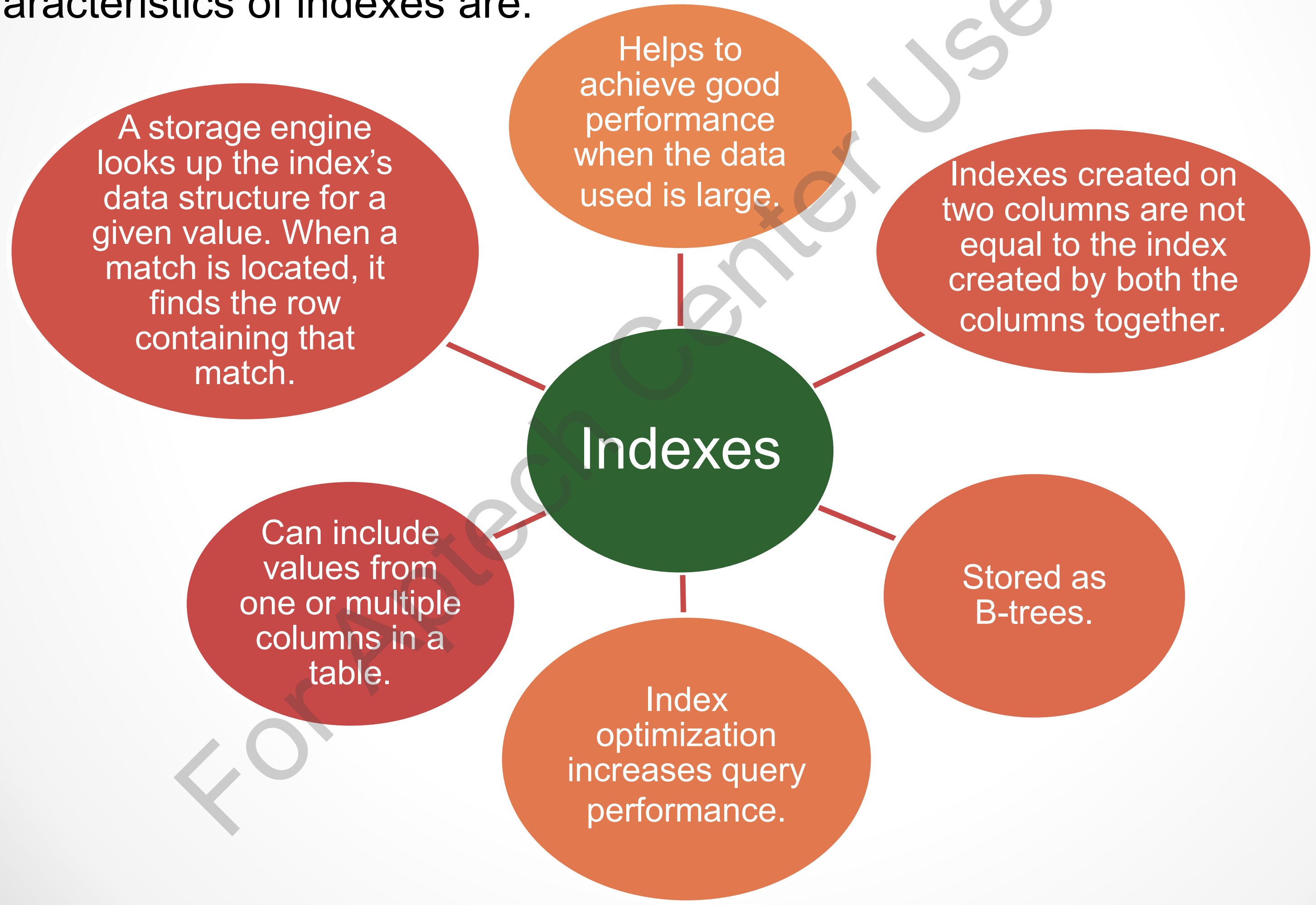
Performance optimization is the process of using techniques to reduce the response time for a given workload.

- It is not about reducing CPU usage, which is limiting resource consumption.
- A database server's performance can be optimized by analyzing the time taken by the server to execute a query and reduce the surplus work it is performing to achieve the same result.

Indexes 1-2

Indexes in MySQL (also called 'keys') are data structures used by storage engines to find rows quickly.

Characteristics of indexes are:


Indexes
Helps to achieve good performance when the data used is large.
Indexes created on two columns are not equal to the index created by both the columns together.
Stored as B-trees.
Index optimization increases query performance.
Can include values from one or multiple columns in a table.
A storage engine looks up the index's data structure for a given value. When a match is located, it finds the row containing that match.

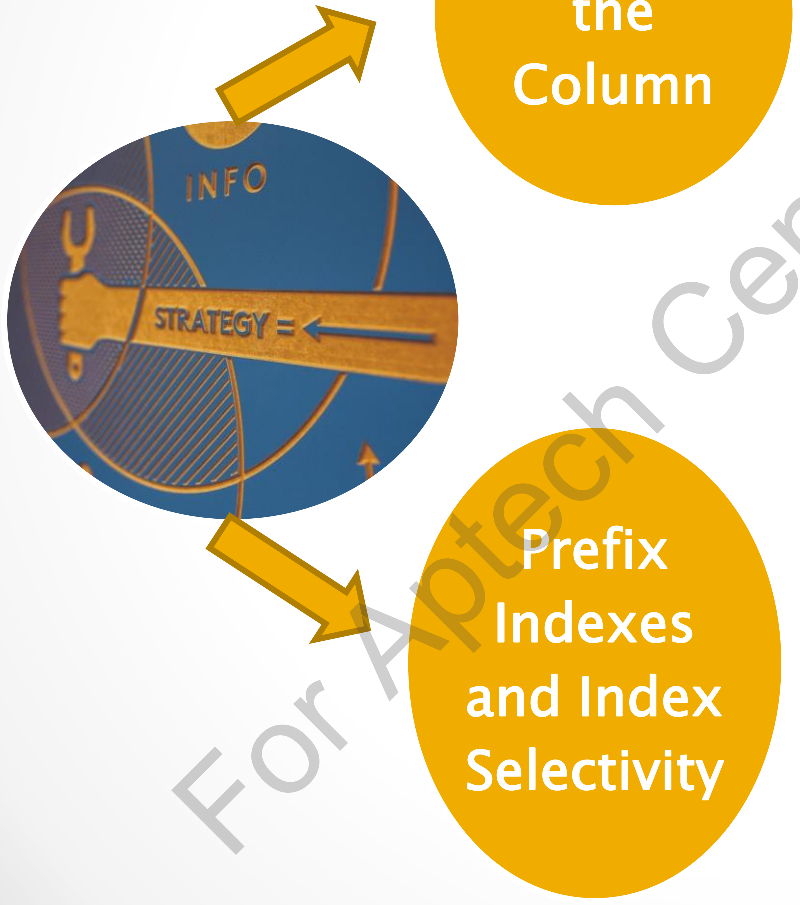
Indexes 2-2

MySQL uses indexes in the following scenarios:

- To eliminate rows
- To look up rows in a table with a multiple-column index
- To obtain rows from different tables when using joins
- To obtain rows that match a WHERE clause faster.
- To compare non-binary string columns, when both columns are using the same character set.
- To compare columns of same type as the values need not be converted.
- To obtain the MIN() or MAX() value for a particular indexed column key_column.
- To sort or group a table when the operation is performed on a leftmost prefix of an index

Indexing Strategies 1-4

Following are strategies to create correct indexes and use them appropriately:



- MySQL cannot apply indexes on columns except when they are isolated in the query.
- Isolating means the columns must not be included in an expression or in a function in the query.

- Indexing the first few characters than the complete value reduces the space and makes them less selective.
- The best way is to select a prefix of enough length to offer correct selectivity but is small enough to reduce space.

Indexing Strategies 2-4

Some more strategies that help to enhance the performance for most queries are:

- Using Multicolumn Indexes (The Index Merge methodology) helps to obtain rows with numerous range scans and to merge their output into a single file.
- Choosing proper order of columns in an index such that the rows are ordered and combined in a way that will benefit the query.
- Using Index Scans for Sorts rather than scanning every row it finds in the index.
- Avoiding creation of duplicate indexes, and removing them if found.
- Discarding the indexes that exist on the server, but are simply not used.

Indexing Strategies 3-4

- All tables in InnoDB have a unique index to store rows' data that is termed as **clustered index**.
- Mostly, this index is the same as the primary key.
- To enhance the performance, analyzing how InnoDB utilizes this type of index is important:

When a PRIMARY KEY is not declared, MySQL identifies the first UNIQUE index in which all the key columns are NOT NULL and InnoDB considers it as the clustered index.

When a PRIMARY KEY or suitable UNIQUE index is not found, InnoDB creates a clustered index on a column that consists of row ID values and this index is hidden.

When a PRIMARY KEY is declared for a table, InnoDB considers it as the clustered index.

Indexing Strategies 4-4

An index that includes (covers) all the data that is necessary for a query is called a **covering index**. Some of its features are:

- Indexes must be created for the full query and not just the WHERE clause.
- These indexes are a powerful tool and can dramatically increase the performance as reading index entries is much quicker than reading the complete row data.
- These indexes are arranged based on their index values, thus I/O-bound accesses perform lesser transactions.
- These indexes are beneficial for InnoDB tables because of clustered indexes.

Disadvantages of Indexes

- When creating indexes on column/columns, MySQL also generates a distinct ordered file that holds only the field(s) that need to be sorted.
- Indexes need plenty of disk space.
- When using indexes with larger tables, the index file can touch the maximum file size allowed on the operating system.
- Indexes reduce the performance of queries such as INSERT, UPDATE, and DELETE.
- In a data file, MySQL has to internally manage pointers to the inserted rows. This causes an overhead on the performance.

Query Caching

Query cache saves data of a SELECT statement along with result sent to the client. Later, if a matching statement is issued, instead of processing the statement again, the server simply obtains the results from query cache.

Advantages

It is shared by multiple sessions, thus, allowing a result set generated by one client to be sent to another client when the same query is executed.

It is suitable for tables that are not modified very often and for which the server gets many identical queries.

When tables are updated, any matching entries in the query cache are removed. Thus, it does not save any irrelevant data.

Disadvantages

When the query cache is extremely large, the overhead involved to manage the cache increases, possibly beyond the benefit of enabling it.

Server workload impacts the efficiency of query cache.

Not suitable for SELECT statements that read from frequently altered tables.

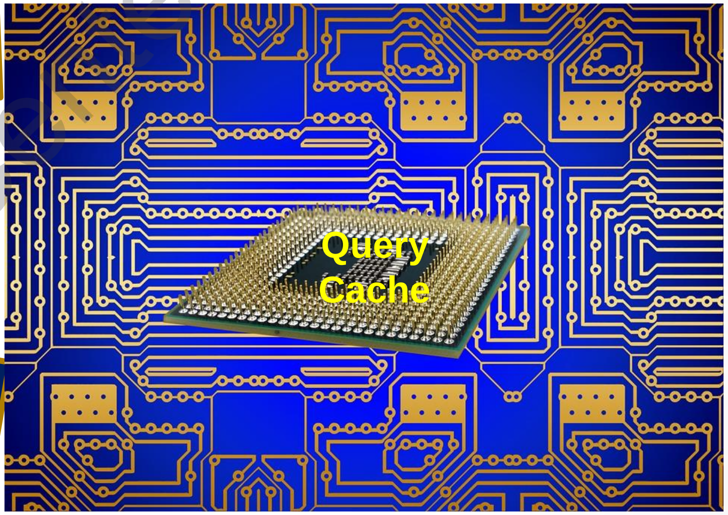
Query Cache Operation 1-3

Following are instances when data can be saved in query cache:

In transactions with InnoDB tables

SELECT SQL_CALC_FOUND_ROWS ... queries and the value returned by SELECT FOUND_ROWS() query

The output from a SELECT query on a view



Query Cache Operation 2-3

Query cache cannot be used under following circumstances:

- Prepared statements that use binary protocol by executing mysql_stmt_prepare() and mysql_stmt_execute()
- User-Defined Functions (UDFs) or stored functions
- User variables, local stored program variables, or partitioned tables
- Tables in mysql, INFORMATION_SCHEMA, or performance_schema database are referred
- Statements within transactions have SERIALIZABLE isolation level
- TEMPORARY tables are referred
- If tables are not used
- If warnings are generated
- If column-level privilege is provided to the user on any of the referred tables

Query Cache Operation 3-3

Query cache cannot be used if the query includes any of the functions listed in the following table:

Function Name		
AES_DECRYPT() (as of 5.7.4)	AES_ENCRYPT() (as of 5.7.4)	BENCHMARK()
CONNECTION_ID()	CONVERT_TZ()	CURDATE()
CURRENT_USER()	CURTIME()	DATABASE()
ENCRYPT() with one parameter	FOUND_ROWS()	GET_LOCK()
IS_FREE_LOCK()	IS_USED_LOCK()	LAST_INSERT_ID()
LOAD_FILE()	MASTER_POS_WAIT()	NOW()
PASSWORD()	RAND()	RANDOM_BYTES()
RELEASE_ALL_LOCKS()	RELEASE_LOCK()	SLEEP()
SYSDATE()	UNIX_TIMESTAMP() with no parameters	USER()
UUID()	UUID_SHORT()	

Query Analysis 1-4

- Performance of the database can be enhanced by using the MySQL Query Analyzer to monitor query performance.
- This analyzer identifies the SQL code causing a slowdown. It uses graphs to analyze the database performance issues.
- Developers and administrators compare MySQL server activity with the executing queries using the correlated graphs by:

Selecting an area on a graph to view the queries being processed at the chosen period.

Using the timeframes with other filtering options such as Query Type, Execution Counts, and First Seen to fine-tune the queries.

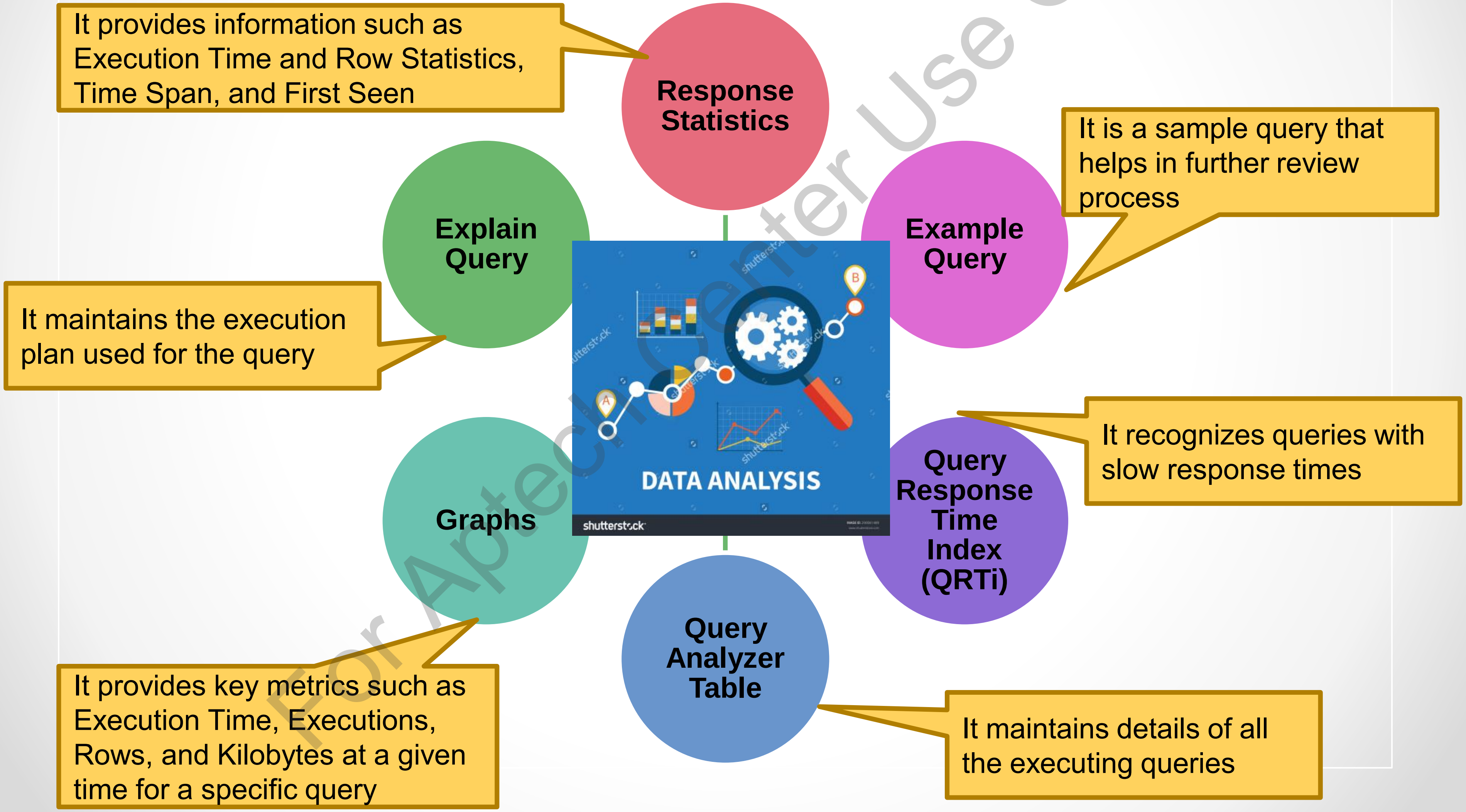
Query Analysis 2-4

MySQL Query Analyzer offers the following advantages to the developers:

- Determines expensive queries affecting the performance of applications.
- Visualizes the complete query activity to obtain an insight into the performance beyond the scope of query statistics.
- Filters issues such as poor indexing and whole table scans with the help of advanced global search options.
- Identifies and fixes the SQL code that is causing a performance slowdown.

Query Analysis 3-4

Following detailed information helps in identifying the queries that can be tuned for better performance:



It provides information such as Execution Time and Row Statistics, Time Span, and First Seen

It is a sample query that helps in further review process

It maintains the execution plan used for the query

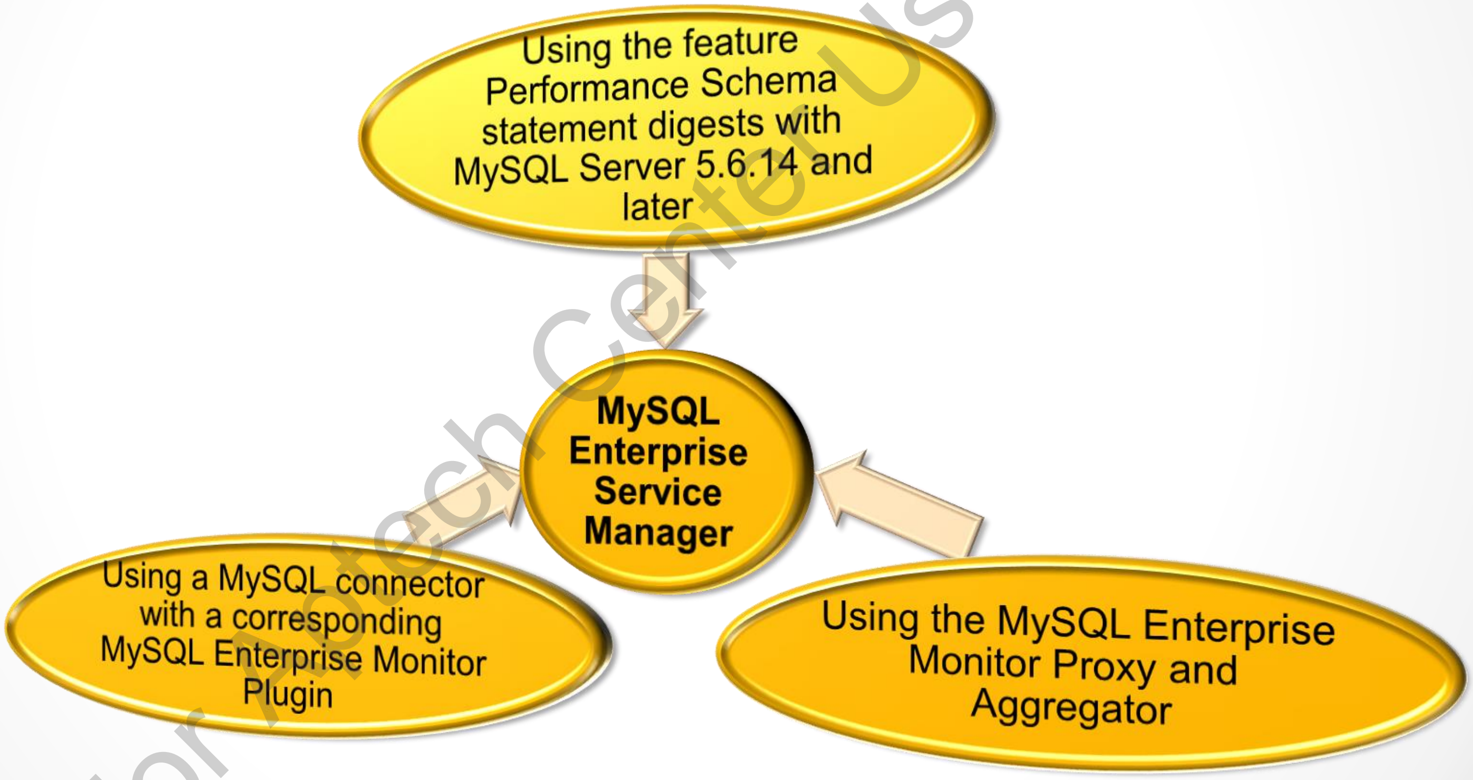
It recognizes queries with slow response times

It provides key metrics such as Execution Time, Executions, Rows, and Kilobytes at a given time for a specific query

It maintains details of all the executing queries

Query Analysis 4-4

The different methods to provide query information to MySQL Enterprise Service Manager to evaluate on the Query Analyzer page are:


Using the feature Performance Schema statement digests with MySQL Server 5.6.14 and later
MySQL Enterprise Service Manager
Using a MySQL connector with a corresponding MySQL Enterprise Monitor Plugin
Using the MySQL Enterprise Monitor Proxy and Aggregator

Optimizing Queries with EXPLAIN 1-2

Characteristics of EXPLAIN statement that gives information about how a statement is processed in MySQL:

Allowed statements for **EXPLAIN** are **SELECT**, **DELETE**, **INSERT**, **REPLACE**, and **UPDATE**.

When used with allowable statements, the statement execution plan is obtained from the optimizer.

When used with **FOR CONNECTION connection_id**, provides the execution plan.

For versions prior to MySQL 5.7.3, **EXTENDED** keyword is required to get the execution plan.

For MySQL 5.7.3 and later, **EXTENDED** is not required as extended output is enabled by default.

Optimizing Queries with EXPLAIN 2-2

For versions prior to MySQL 5.7.3, **`PARTITIONS`** keyword is required to analyze queries consisting of partitioned tables.

For MySQL 5.7.3 and later, **`PARTITIONS`** is not required as partition information is enabled by default.

Use **`FORMAT`** option to choose the output format.

Table Design to Optimize Data

- Designing tables such that they reduce the storage space results in huge performance improvements.
- MySQL supports numerous table types and row formats.
- The techniques that help to achieve better performance and reduce storage space are:
  - **Table Columns**: Choose the most efficient, preferably the smallest data types.
  - **Row Format**: Using `COMPACT` format reduces the row storage space by about 20% thus, allowing the CPU to be utilized for other operations.
  - **Indexes**: Keeping the primary index as short as possible ensures the rows are identified easily and efficiently.
  - **Joins**: For a table scanned frequently, in some instances, it is recommended to break it into two. Define columns with identical data in different tables with identical data types, to increase the performance of joins.
  - **Normalization**: Avoid redundant data (third normal form). Allocate unique IDs to lengthy values such as names and addresses and when the values need to be repeated, use the IDs.

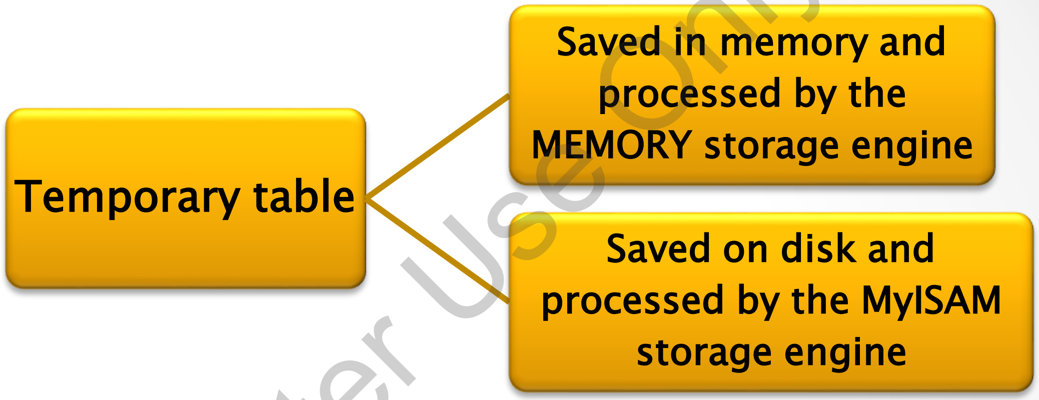
Temporary Tables 1-2

Following are scenarios when server generates temporary tables internally while executing queries and the user has no control over this:

- Assessing UNION statements.
- Assessing views that include the TEMPTABLE algorithm, UNION, or aggregation.
- Assessing statements that use ORDER BY clause and a different GROUP BY clause.
- Assessing DISTINCT used with ORDER BY.
- Assessing queries that include the option SQL_SMALL_RESULT.
- Assessing UPDATE statements defined on multiple tables.
- Assessing GROUP_CONCAT() or COUNT(DISTINCT) expressions.

Temporary Tables 2-2

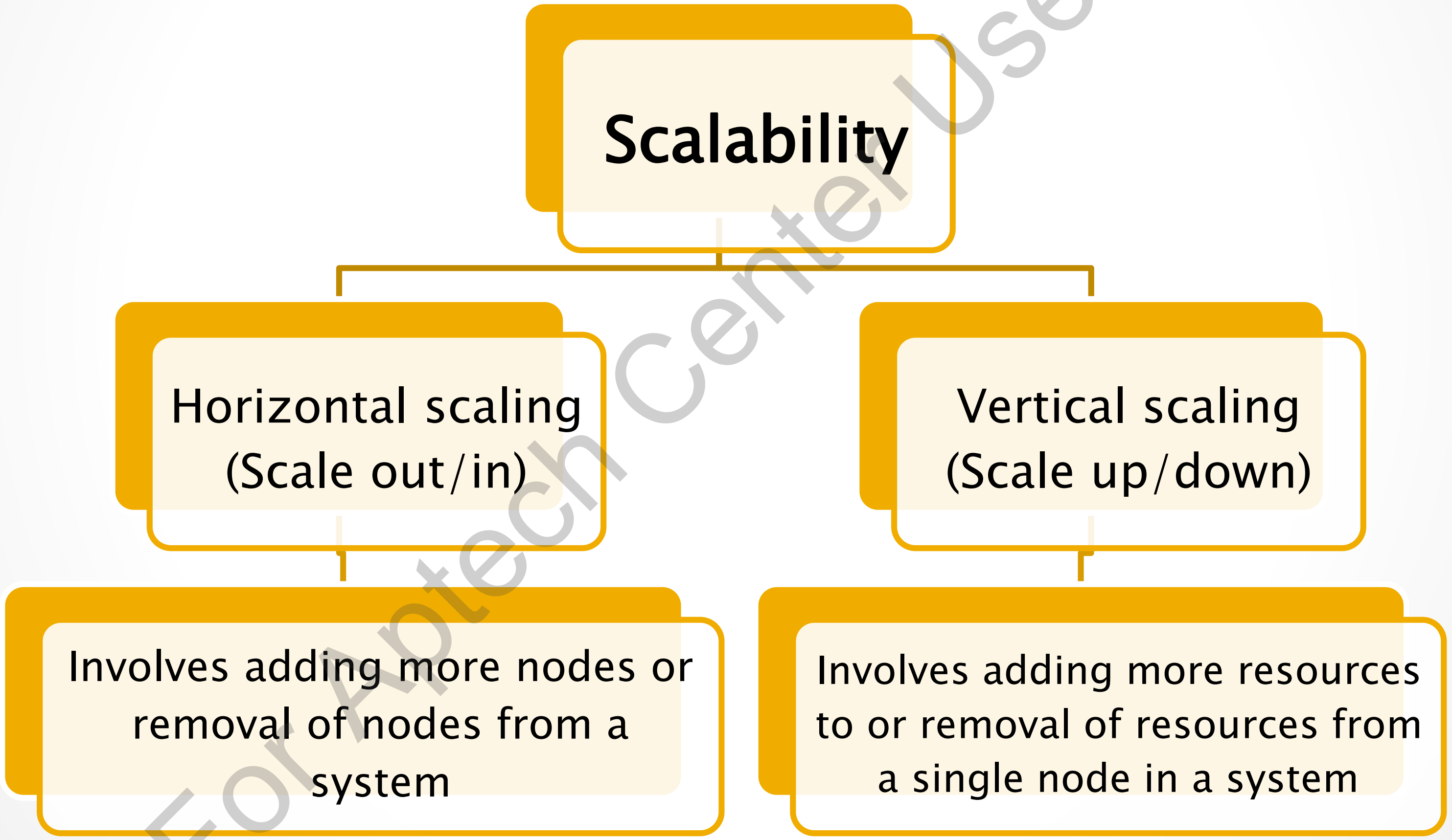
An internal temporary table can be stored in two ways:


Temporary table
Saved in memory and processed by the MEMORY storage engine
Saved on disk and processed by the MyISAM storage engine

- An internal temporary table, which is an in-memory table, is changed to an on-disk table when the size is too big.
- Following are conditions that prevent usage of in-memory temporary tables and instead, on-disk table is used:
  - Using a BLOB or TEXT column in the table.
  - Using any string column in a GROUP BY or DISTINCT clause, which exceeds 512 bytes.
  - Using any string column with the length exceeding 512 in the SELECT list, if UNION or UNION ALL is utilized.
  - Using BLOB for some columns along with SHOW COLUMNS and DESCRIBE statements.

Scaling MySQL

Scalability is the capability of a system, network, or process to manage a growing amount of work, or its ability to enlarge so as to adapt to that growth.


Scalability
Horizontal scaling (Scale out/in)
Vertical scaling (Scale up/down)
Involves adding more nodes or removal of nodes from a system
Involves adding more resources to or removal of resources from a single node in a system

Summary 1-2

- Performance is a measure of time that is required to complete a task. Performance optimization is the process of using techniques to reduce the response time for a given workload.
- Indexes in MySQL (also called 'keys') are data structures used by storage engines to find rows quickly.
- Index optimization is the most effective technique to increase query performance.
- The query cache saves the data of a SELECT statement along with the result sent to the client. Later, if a matching statement is issued, instead of processing the statement again, the server simply obtains the results from the query cache.

Summary 2-2

- Performance of the database can be enhanced by using the MySQL Query Analyzer to monitor query performance.
- Designing tables such that they reduce the storage space results in huge performance improvements as the data written to and read from the disk is also reduced.
- Scalability is the capability of a system, network, or process to manage a growing amount of work, or its ability to enlarge so as to adapt to that growth.