

SUMMARY: INTRODUCTION TO SQL

SESSION OVERVIEW:

By the end of this session, the students will be able to:

- Understand databases, the importance of databases, and different types of databases.
- Understand the uses and importance of SQL in real-world scenarios.
- Understand the installation and the interface of SQL.

KEY TOPICS AND EXAMPLES:

Understanding databases, the importance of databases, and different types of databases:

1. What is a database?

A database is an organized collection of data that allows for the efficient storage, retrieval, management, and modification of information. It is designed to handle vast amounts of data and supports processes requiring complex querying, rapid transaction processing, and analytics.

2. What is DBMS?

A Database Management System (DBMS) is specialized software designed to store, retrieve, manage, and manipulate data in databases. It serves as an interface between databases and end-users or application programs, ensuring that data is consistently organized and remains easily accessible. DBMSs provide the tools needed for defining, constructing, manipulating, and administering databases, enabling users to create, read, update, and delete data in a database, a process often referred to by the acronym CRUD (which will be covered in the upcoming sessions.)

3. Importance of DBMS:

a. Improved Data Sharing and Accessibility:

DBMS allows multiple users and applications to access and share data efficiently. By managing data centrally, it ensures that data is accessible to authorized users, enhancing collaboration and productivity.

b. Data Integrity and Security:

DBMS enforces rules to maintain data integrity and consistency across the database. It also provides mechanisms to control access to data, ensuring that only authorized users can access or manipulate sensitive information, thus bolstering data security.

c. Data Management Efficiency:

With a DBMS, data can be stored in a structured manner, making it easier to retrieve, insert, update, and delete data efficiently. This optimizes the use of resources and improves the overall performance of data-driven applications.

d. Enhanced Data Analysis:

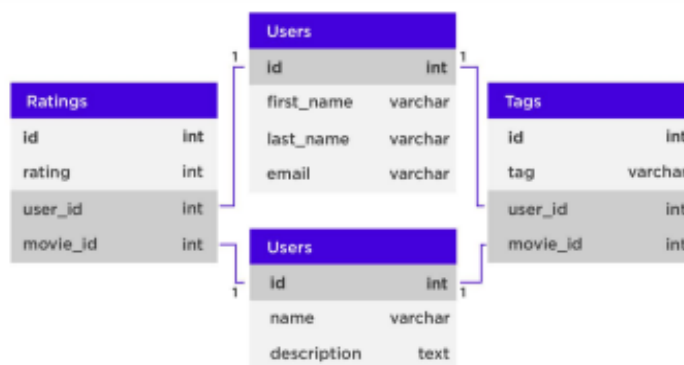
With centralized data management, DBMS facilitates more comprehensive data analysis and reporting. Users can query the database to gather insights, identify trends, and make informed decisions based on real-time data.

4. Types of DBMS:

There are several types of databases, each designed for specific use cases. Here are some of the most common types of databases:

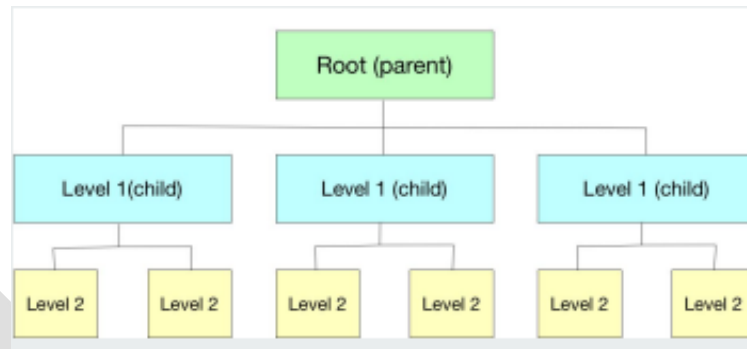
- a. **Relational databases:** The most popular kind of database, these store data in an organized manner using tables. The relational model, which describes how data is arranged in tables and how those tables relate to one another, is the foundation of relational databases. It also supports the concept of joining tables using primary keys, and foreign keys.

Relational databases that are widely used include **Microsoft SQL Server**, **PostgreSQL**, and **MySQL**. We will be using MySQL throughout the module.



- b. **Hierarchy database:** It stores data in the form of parent-children relationship nodes. It organizes data in a tree-like structure. Data get stored in the form of records that are connected via links. Each child record in the tree will contain only one parent. On the other hand, each parent record can have multiple child records. The addition of data elements requires a lengthy traversal through the database. This kind of data storage was used in Organizational data storage (Eg., CEO -> Director -> Manager -> Analyst). This type of database storage is not used very widely now as these are replaced by more flexible RDBMS.

Examples: IBM Information Management System (IMS), Windows Registry, XML Data Storage.



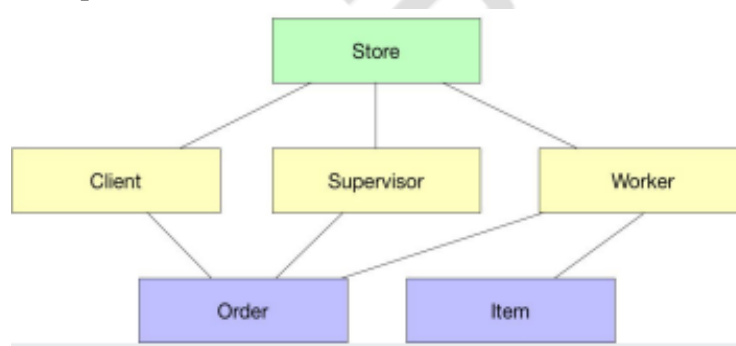
- c. **Network Database:** The representation of data is in the form of nodes connected via links between them. It organizes data in a generalized graph structure. Unlike the hierarchical database, it allows each record to have multiple children and parent nodes. Unable to alter the structure due to its complexity and also in it being highly structurally dependent. Similar to the hierarchical model but allows many-to-many relationships among data points, forming a network-like structure.

This type of storage was used in library systems where -

- Books can be written by one or more Authors.
- Books can belong to one or more Categories (e.g., Fiction, Science, History).
- Authors can write multiple Books.
- Categories can include multiple Books.

Not in use anymore and replaced by RDBMS majorly

Example: RDM Server.



- d. **NoSQL databases (Not Only SQL):** These databases are designed to handle unstructured or semi-structured data, such as social media posts, and product reviews stored in **JSON documents**. Unlike relational databases, NoSQL databases do not use tables or a fixed schema to store data. Instead, they use flexible data models, such as document, key-value, or graph, to store data.

NoSQL databases are designed to address several limitations of relational databases, particularly in terms of scalability, flexibility, and the ability to handle large volumes of unstructured or semi-structured data. NoSQL databases are especially popular in applications requiring rapid development, high performance at scale, and the ability to store and manage data in formats not suited to relational schemas.

Some of the popular NoSQL databases include **MongoDB, Cassandra, and Redis**.

Example:

E-commerce platform that uses NoSQL databases to manage its data:

- **User Profiles (Document Database):** Each user profile contains diverse information that can change over time, such as personal information, addresses, and payment methods.
- **Product Catalog (Wide-Column Store):** Products have a wide range of attributes that vary significantly from one item to another, making wide-column stores suitable for efficiently querying products based on various attributes.
- **Session Data (Key-Value Store):** To quickly access user session information by session ID for a fast and seamless user experience.
- **Recommendations (Graph Database):** To model the complex relationships between users and products and provide personalized product recommendations based on user behavior and preferences.

Advantages of NoSQL Database:

- **Scalability:** NoSQL databases are designed to excel in horizontal scaling, which involves adding more servers to handle increased load.
- **High Performance:** Many NoSQL systems are optimized for specific data models and access patterns which can offer higher performance for certain types of queries. For example, key-value stores are optimized for quick retrieval of data based on a key, and document stores efficiently query data by bypassing the join operations often needed in relational databases.

Disadvantages of NoSQL Database:

- **Lack of Standardization:** NoSQL databases do not have a standardized query language, leading to variations in how data is manipulated and accessed across different types of NoSQL systems.
- **Consistency:** This can be problematic for applications that require real-time data accuracy and consistency, such as financial transaction systems.
- **Query Complexity:** NoSQL might not perform well with complex queries, such as those involving joins across multiple types of data or hierarchical data structures. This can limit the types of questions that can be easily answered by the database.

5. What is a Relational Database Management System?

A relational database is a type of database that stores data in tables. Each table contains rows and columns, where each column represents a data field, and each row represents a record or an instance of that data. The columns in one table can be related to columns in another table, creating a "relationship" between the tables.

Example:

For example, let's say you're building a customer relationship management (CRM) application. You may have a "customers" table that contains customer data such as name, email address, and phone number. You may also have an "orders" table that contains order data such as order date, order total, and customer ID. By using a relational database, you can

link the "customers" and "orders" tables together using the customer ID column in the "orders" table. This allows you to retrieve order data for a specific customer or group of customers.

6. Components of RDBMS:

a. Tables:

Data is arranged in tables within an RDBMS, which are two-dimensional structures made up of rows and columns. A particular entity, such as a customer, a product, or an order, is represented by each table. Individual instances of the object are represented by rows in a table, often referred to as records or tuples, while characteristics or properties are represented by columns.

b. Relationships:

RDBMS supports relationships between tables. These associations specify the relationship between the data in one table and the data in another. Foreign keys and primary keys are two examples of the keys that are used to construct relationships. Every record in a table is uniquely identified by its primary key, and foreign keys connect records from one table to their corresponding records in another.

Course ID	Course Name	Instructor	Student ID	First Name	Last Name	Age	Grade	Course ID
101	Mathematics	Prof. Anderson	001	John	Smith	18	A	101
102	English	Prof. Thompson	002	Emily	Johnson	19	B	102
103	Science	Prof. Parker	003	Michael	Williams	20	A	103

Example:

- In the above example, we have two tables. The first table represents the instructor's name corresponding to the courses that they teach. The second table represents the students who are enrolled in those courses.
- Let's name both the tables as Instructors_details and Students_details.
- In the Instructors_details, the course ID represents the primary key and in the students_details, the student ID represents the primary key and course ID represents the foreign key which will help us to create a relationship between both the tables.

c. Data integrity:

RDBMS enforces data integrity rules to ensure the accuracy and consistency of data. This includes constraints such as primary key constraints (ensuring uniqueness), foreign key constraints (maintaining referential integrity), NOT NULL (absence of NULL values) and other rules to prevent invalid data entries.

d. Normalization:

RDBMS supports normalization, a process of organizing data to reduce redundancy and dependency. Normalization helps eliminate data anomalies and ensures efficient storage and retrieval of information.

e. **ACID properties:**

RDBMS adheres to the ACID properties, which are crucial for maintaining the reliability of transactions. ACID stands for Atomicity (transactions are treated as a single, indivisible unit), Consistency (database remains in a consistent state after a transaction), Isolation (transactions do not interfere with each other), and Durability (once a transaction is committed, its changes are permanent).

7. **Advantages of RDBMS:**

- **Data Structure and Integrity:** The structured format and data integrity constraints ensure the accuracy and reliability of data storage and retrieval.
- **Scalability and Flexibility:** While RDBMSes are traditionally scaled up (increasing the power of the existing hardware), modern RDBMS solutions offer features for horizontal scalability (distribution across multiple servers).
- **Mature Technologies:** RDBMS technologies are mature, with robust support communities, extensive documentation, and proven reliability.

8. **Disadvantages of RDBMS:**

- **Complexity and Overhead:** The requirement for predefined schemas and normalization can introduce complexity and overhead, especially for rapidly evolving data models.
- **Scalability Limits:** For very large databases or highly distributed environments, traditional RDBMS scalability might be challenged, requiring more complex solutions or the adoption of NoSQL databases in some cases.

9. **What makes SQL so important?**

a. **Data Manipulation and Retrieval:**

The data manipulation and retrieval tools enable the extraction of meaningful information from large and complex datasets through sophisticated queries, including joining tables, filtering records, and aggregating data.

b. **Data Modification:**

SQL allows users to modify data within databases. Whether it's inserting new records, updating existing ones, or deleting unnecessary data, SQL provides powerful commands to make these changes accurately and efficiently.

c. **Database Design and Structure:**

SQL is crucial for designing the structure of databases, including defining tables, relationships between tables, and specifying data types. This helps in creating organized and efficient databases that can scale with the growing volume of data.

FUN TIME:

The Kitchen Analogy

Think of your kitchen as a database. Now, this "kitchen database" needs to organize various items (data) like utensils, appliances, and groceries. A schema in this context would be the blueprint or plan that defines where and how these items are stored.

Components of the Kitchen Schema

Cabinets and Drawers (Tables):

- Each cabinet or drawer is like a table in a database schema.
- For example, one drawer is designated for utensils (one table), and another cabinet is for dishes (another table).

Shelves/Sections within Cabinets (Columns):

- Inside each drawer or cabinet (table), there are sections or shelves. These are like columns in a database table.
- In the utensils drawer, one section might be for knives, another for forks, and so on. Each section specifies the type of item it holds, similar to how each column in a database table specifies the type of data it stores (e.g., text, date, number).

Organizing Principle (Rows):

- Each item placed in its specific location within a drawer or cabinet represents a row in a database table.
- If you add a set of steak knives to the knife section of the utensils drawer, each steak knife is like a row in the "utensils" table under the "knives" column.

Relationships:

- Imagine you have a recipe book (another table) in your kitchen. Each recipe lists the utensils and ingredients (data from other tables) needed.
- The relationship is like saying, "To make an omelet (a row in the recipe book), you need a whisk and a frying pan (data from the utensils table) and eggs and cheese (data from the groceries table)."

Rules (Constraints):

- You might have a rule: "No perishable food items should be stored outside the refrigerator." In a database, this is similar to a constraint, like "Email addresses in the contacts table must be unique."

Special Tools (Views, Stored Procedures):

- *Special kitchen gadgets that perform specific tasks can be thought of as views or stored procedures. For instance, a coffee maker (a view) prepares your coffee by filtering water through coffee grounds, providing you with ready-to-drink coffee without the need to interact directly with the water or the coffee grounds.*

Understanding Through the Analogy

The schema organizes the kitchen (database) in a way that makes it easy to find and use items (data). Just as a well-organized kitchen is essential for efficient cooking, a well-designed database schema is crucial for efficient data storage, retrieval, and manipulation.

Understanding the uses and importance of SQL in real-world scenarios:

SQL (Structured Query Language) is considered to be a very crucial software in different types of industries due to its powerful ability to manage, manipulate, and query relational databases. Its widespread adoption is driven by several key factors that make it indispensable for data-driven decision-making, operational management, and strategic planning across various sectors.

1. Education system:

- a. Imagine a university that maintains separate databases for enrollment, attendance, instructor assignments, course schedules, and student course ratings. These databases are linked through a common field, such as the student ID or course ID, to provide a comprehensive view of each student's academic journey.

Databases:

i. Enrollment database:

Purpose: Tracks which students are enrolled in which courses.

Example Fields:

- **StudentID:** Unique identifier for each student.
- **CourseID:** Unique identifier for each course.
- **EnrollmentDate:** Date when the student enrolled in the course.

Example Entry:

StudentID: 001, CourseID: BIO101, EnrollmentDate: 2021-09-01

ii. Attendance Database:

Purpose: Records student attendance for each course session.

Example Fields:

- **AttendanceID:** Unique identifier for each attendance record.
- **CourseID:** Identifies the course.
- **StudentID:** Identifies the student.
- **Date:** The date of the class.

- **AttendanceStatus:** Indicates whether the student was present, absent, etc.

Example Entry:

AttendanceID: 12345, CourseID: BIO101, StudentID: 001, Date: 2021-09-02, AttendanceStatus: Present

iii. **Instructors Database:**

Purpose: Lists which instructors teach which courses.

Example Fields:

- **InstructorID:** Unique identifier for each instructor.
- **Name:** Name of the instructor.
- **CourseID:** Courses they teach.

Example Entry:

InstructorID: 9001, Name: Dr. Smith, CourseID: BIO101

iv. **Course Schedule Database:**

Purpose: Provides details about when and where each course is held.

Example Fields:

- **CourseID:** Unique identifier for each course.
- **RoomNumber:** Location of the course.
- **TimeSlot:** Time when the course is scheduled.

Example Entry:

CourseID: BIO101, RoomNumber: Room 101, TimeSlot: Wednesdays 10AM

v. **Ratings Database:**

Purpose: Captures student ratings and feedback for courses.

Example Fields:

- **RatingID:** Unique identifier for each rating entry.
- **StudentID:** Identifies the student.
- **CourseID:** Identifies the course.
- **Rating:** Numerical rating or feedback.

Example Entry:

RatingID: 7890, StudentID: 001, CourseID: BIO101, Rating: 5

vi. **Course Database:**

Purpose: Captures details of all the courses.

Example Fields:

- **CourseId:** Unique identifier for each course entry.
- **Name:** Identifying the name of the course.
- **Description:** Describes the course.

vii. **Student Database:**

Purpose: Captures details of all the students.

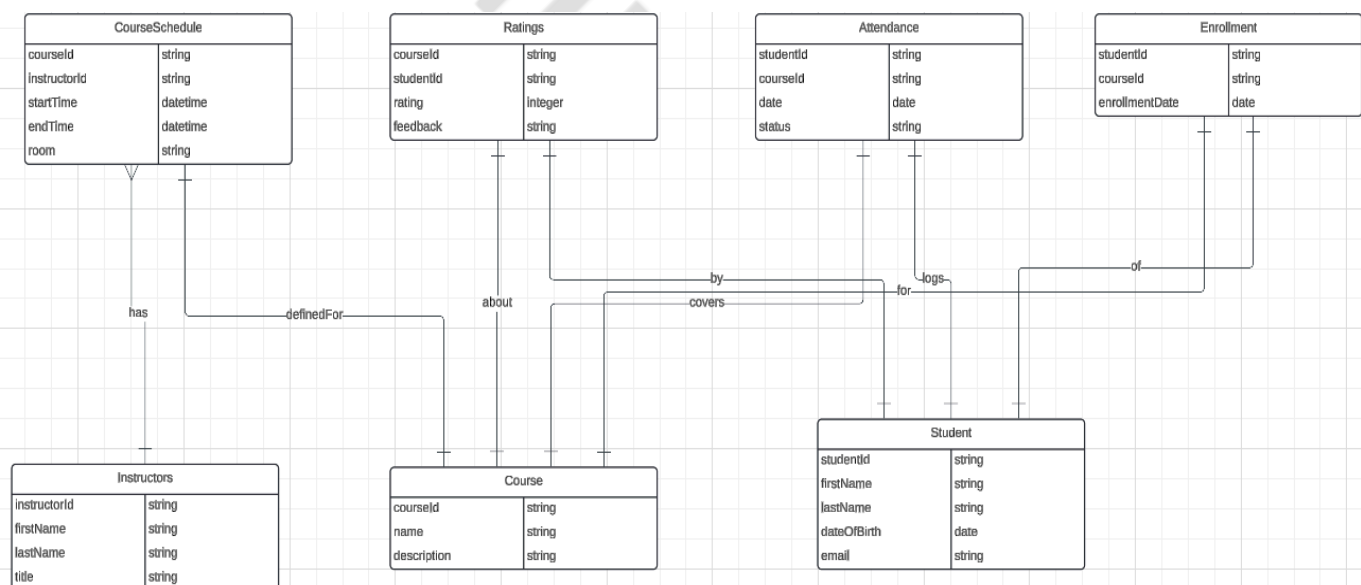
- **studentID:** Unique identifier of the students.
- **FirstName:** Provides the first name of the student.
- **LastName:** Provides the last name of the student.
- **DateofBirth:** Provides the date of birth of the student.
- **Email:** Helps get the student's email address.

Linking the Databases

By using the StudentID and CourseID as common fields across these databases, the university can interconnect the data to gain insights such as:

- **Student Profiles:** Aggregating data from all databases to view a student's course enrollments, attendance records, course schedules, instructor interactions, and course feedback.
- **Course Performance Analysis:** Analyzing attendance and rating data to determine how different factors like classroom engagement or instructor influence student satisfaction and success.

For instance, if the university wants to review a student's engagement in a course, they can pull data showing the student's enrollment, attendance records, interactions with the instructor, and feedback about the course. This holistic view helps in understanding the student's overall experience and identifying areas for improvement in teaching methods or course content.



Real incidences in the education industry:

(Commonly encountered scenarios in education industries)

- Enrollment Anomalies:** An SQL query identified a set of students who were mistakenly not enrolled in mandatory courses due to a clerical error. The problem was quickly rectified, and the students were enrolled in the required courses.
- Grade Inconsistencies:** A university used SQL to audit grades and discovered inconsistencies due to a misconfiguration in the grade calculation algorithm. The SQL-based analysis helped correct the students' records.

Understanding the installation and the interface of SQL:

1. Installation of MySQL and related applications:

The installation guide for downloading MySQL, MySQL Shell, and MySQL Workbench has been attached.

The students need to follow the guide to download and set up the purpose of MySQL.

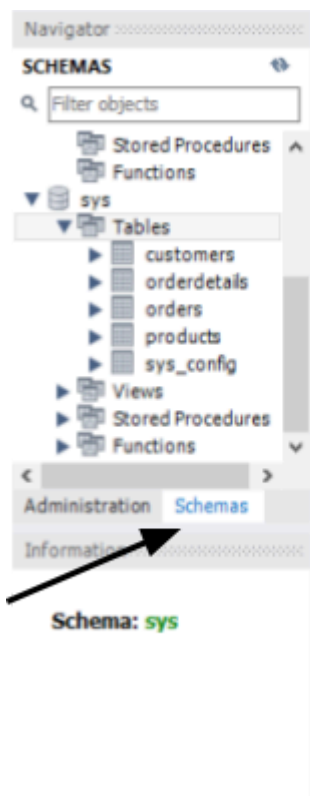
- [MySQL Installation Guide- Windows](#)
- [MySQL Installation Guide- MacOS](#)

NOTE: If the student faces any sort of problem while downloading MySQL into their system, the student needs to clear the doubt during the session itself.

2. Understanding the interface of MySQL Workbench:

The interface of MySQL Workbench can be confusing at times and might be overwhelming to get the desired options in the software. This is the reason we will be going through each and every important feature in MySQL Workbench to get you acquainted with the platform. The following are some of the important MySQL Workbench features:

a. Navigator:

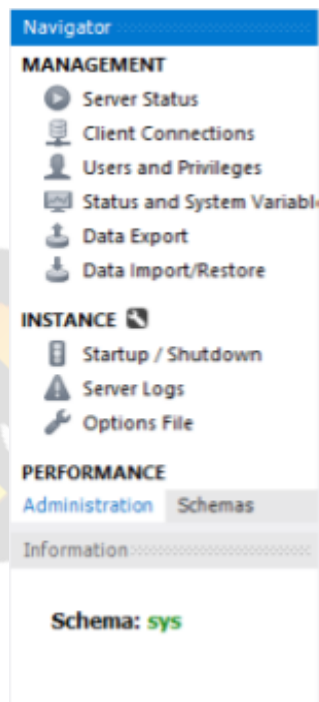


The Navigator in MySQL Workbench is a fundamental component that serves as a centralized panel providing quick access to various functionalities related to database management and development. It plays a crucial role in streamlining workflows within MySQL Workbench by allowing users to easily navigate through database objects, perform administrative tasks, and access tools for SQL development.

Schemas navigator:

- Viewing Database Objects
- Creating New Objects
- Modifying Existing Objects
- Exporting and Importing Schema

Figure: Represents the use cases of the schema navigator



The Administration Navigator in MySQL Workbench, often referred to as the "Instance" or "Server" Navigator depending on the Workbench version and context, provides a comprehensive set of tools for database administration. It facilitates various server management tasks, making it an essential component for database administrators (DBAs). Key features are as follows:

- Provides real-time insights into the health and performance of the MySQL server.

- Enables the creation, editing, and deletion of user accounts.

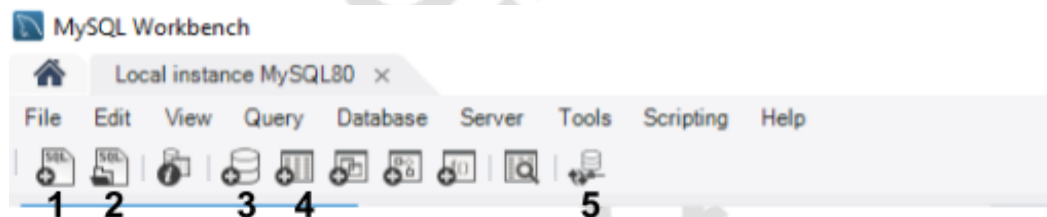
- Facilitates the scheduling and management of database backups.

- Supports the import and export of database data and structures.

- Assists in performing routine maintenance tasks such as checking table integrity.

Figure: Represents the use cases of the administration navigator

b. Main Toolbar:



New Query Tab: Opens a new tab within the SQL Editor where you can write and execute SQL queries.

New SQL Script: Allows users to quickly load and work with pre-written SQL scripts.

New Schema in the connected server: Launches the wizard to create a new database schema on the connected MySQL server.

New table in the active schema: It's a fundamental operation for expanding the database schema to accommodate new types of information as an application evolves.

Reconnect to DBMS: Used to re-establish a lost connection to the database server, ensuring continued database access without the need to manually disconnect and reconnect.

c. Query Editor:

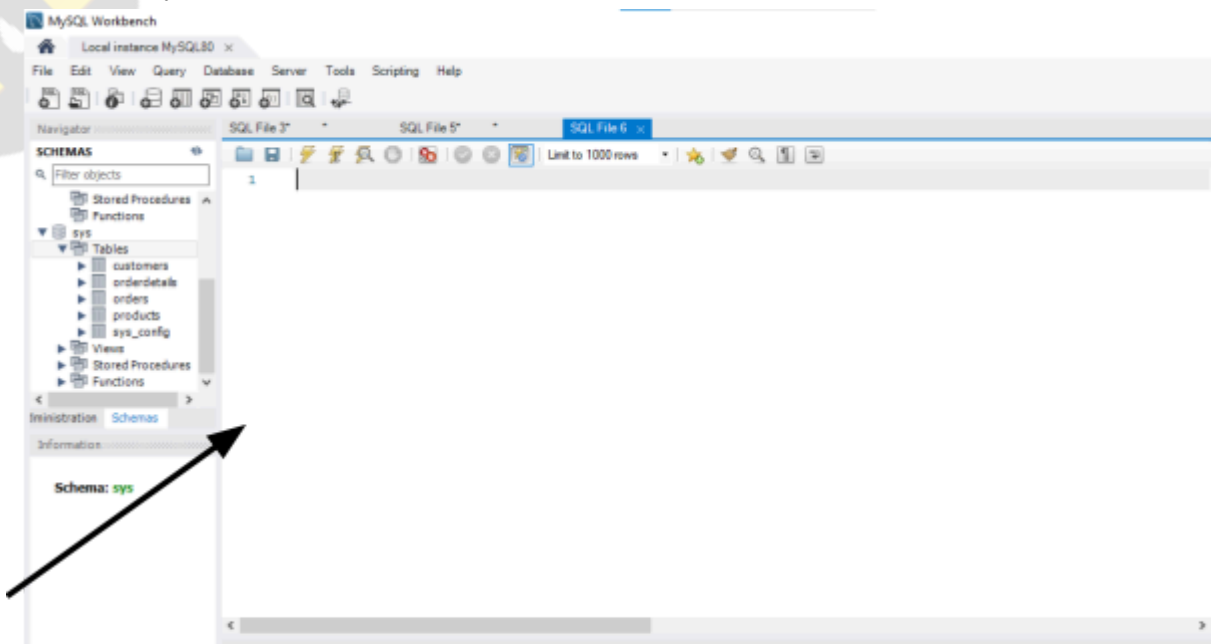


Execute: Runs the SQL query currently written in the active SQL Editor tab.

Save File: Saves the current SQL script or model that you're working on.

Open File: Allows you to open an existing SQL script file into a new tab of the SQL Editor.

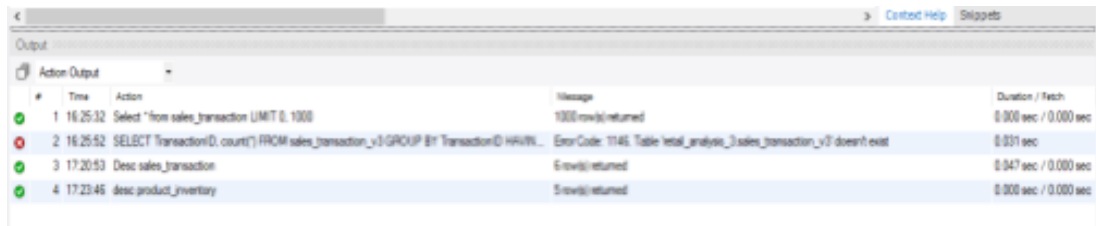
d. Query Editor:



The arrow represents the area called as the Query Editor. The Query Editor in MySQL Workbench is a feature-rich environment where users can write, edit, format, and execute SQL queries.

e. Action Output:

NOTE: The student must not panic referring to the image attached. The outputs will vary depending on the query you input. The idea of the image attached is just to give an overview of the action output.



#	Time	Action	Message	Duration / Fetch
1	16:25:32	Select * from sales_transaction LIMIT 0, 1000	1000 row(s) returned	0.000 sec / 0.000 sec
2	16:25:52	SELECT TransactionID, count(*) FROM sales_transaction_v3 GROUP BY TransactionID HAVING...	Error Code: 1146, Table 'retail_analytic_3.sales_transaction_v3' doesn't exist	0.031 sec
3	17:20:53	Desc sales_transaction	6 row(s) returned	0.047 sec / 0.000 sec
4	17:23:45	desc product_inventory	5 row(s) returned	0.000 sec / 0.000 sec

This is the output section where we get certain messages regarding the query which have been executed. The messages could be regarding:

- The different types of errors in the queries, example syntax error or column doesn't exist, etc.

- If the queries got successfully executed or not, and if yes then how many rows got affected with the help of the query.

- The duration of executing and fetching the queries.