MySQL Notes

What is a Database?

A database is a collection of interrelated data.

What is DBMS?

DBMS (Database Management System) is software used to create, manage, and organize databases.

What is RDBMS?

RDBMS (Relational Database Management System) is a type of DBMS based on the concept of tables (relations). - Data is organized into tables with rows (records) and columns (attributes). - Examples: MySQL, PostgreSQL, Oracle, etc.

What is SQL?

SQL (Structured Query Language) is used to store, manipulate, and retrieve data from RDBMS. - SQL enables CRUD operations: - **CREATE**: Create databases, tables, and insert records. - **READ**: Fetch data from the database. - **UPDATE**: Modify existing data. - **DELETE**: Remove data or database objects.

> *Note: SQL keywords are case-insensitive (e.g., **SELECT** is the same as **select**).*

SQL vs MySQL

- **SQL**: A language used to interact with RDBMS.
- **MySQL**: An RDBMS that uses SQL.

MYSQL Data Types

Define the type of data stored in a column:

Data Type	Description	Example
CHAR	Fixed-length string (0–255 characters)	**CHAR(50)**
VARCHAR	Variable-length string (0–255 characters)	**VARCHAR(50)**
BLOB	Binary large object (0–65535 bytes)	**BLOB(1000)**

Data Type	Description	Example
INT	Integer (-2,147,483,648 to 2,147,483,647)	**INT**
TINYINT	Small integer (-128 to 127)	**TINYINT**
BIGINT	Large integer (-9 quintillion to 9 quintillion)	**BIGINT**
FLOAT	Decimal number with up to 23 digits precision	**FLOAT**
DOUBLE	Decimal number with 24–53 digits precision	**DOUBLE**
BOOLEAN	Boolean (0 or 1)	**BOOLEAN**
DATE	Date (YYYY-MM-DD)	**DATE**
TIME	Time (HH:MM:SS)	**TIME**
YEAR	Year (4 digits, 1901–2155)	**YEAR**

*Note: Use **VARCHAR** for variable-length strings to save memory.*

Types of MYSQL Commands

1. DQL (Data Query Language)

- Retrieve data from databases.
- Example: **SELECT**.

2. DDL (Data Definition Language)

- Define and manage database structure.
- Examples: **CREATE**, **DROP**, **ALTER**, **RENAME**, **TRUNCATE**.

3. DML (Data Manipulation Language)

- Modify data in databases.
- Examples: **INSERT**, **UPDATE**, **DELETE , REPLACE**.

4. DCL (Data Control Language)

- Control access and permissions.

- Examples: **GRANT**, **REVOKE**.

5. TCL (Transaction Control Language)

- Manage database transactions.
- Examples: **COMMIT**, **ROLLBACK**, **SAVEPOINT**.

MYSQL Notes

---

1. Data Definition Language (DDL)

DDL (Data Definition Language) is a subset of MYSQL responsible for defining and managing the structure of databases and their objects.

Key DDL Commands:

- **CREATE TABLE**
  - Used to create a new table in the database.
  - Specifies table name, column names, data types, constraints, etc.
  - **Example:**

```
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  salary DECIMAL(10, 2)
);
```

- **ALTER TABLE**
  - Used to modify the structure of an existing table.
  - **RENAME:** Just change name of a column not definition.
  - **MODIFY:** Change just definition of column.
  - **CHANGE:** Change both name and definition.
  - **Example:**

```
ALTER TABLE employees ADD COLUMN email VARCHAR(100);
```

- **DROP TABLE**
  - Deletes an existing table along with its data and structure.
  - **Example:**

```
DROP TABLE employees;
```

- **CREATE INDEX**
  - Creates an index on one or more columns to improve query performance.
  - **Example:**

```
CREATE INDEX idx_employee_name ON employees (name);
```

- **DROP INDEX**
  - Removes an existing index from a table.
  - **Example:**

```
DROP INDEX idx_employee_name;
```

- **CREATE CONSTRAINT**
  - Defines constraints to ensure data integrity (e.g., PRIMARY KEY, FOREIGN KEY).
  - **Example:**

```
ALTER TABLE orders ADD CONSTRAINT fk_customer FOREIGN KEY
(customer_id) REFERENCES customers(id);
```

- **TRUNCATE TABLE**
  - Deletes all rows in a table without removing its structure.
  - **Syntax:**

```
TRUNCATE TABLE table_name;
```

---

2. Data Query/Retrieval Language (DQL/DRL)

DQL (Data Query Language) focuses on retrieving data from databases using the **SELECT** statement.

KEY DQL COMMANDS:

- **SELECT**
  - Used to select specific columns or all columns from a table.
  - **Syntax:**

    ```
    SELECT column1, column2 FROM table_name;
    SELECT * FROM table_name;
    ```

  - **Example:**

    ```
    SELECT CustomerName, City FROM Customers;
    ```

- **WHERE**
  - Filters records based on specific conditions.
  - **Syntax:**

    ```
    SELECT * FROM Customers WHERE Country = "INDIA"';
    ```

- **DISTINCT**
  - Removes duplicate rows in query results.
  - **Syntax:**

    ```
    SELECT DISTINCT column1 FROM table_name;
    ```

- **LIKE**
  - Searches for patterns in data.
  - **Examples:**

    ```
    SELECT * FROM employees WHERE first_name LIKE 'J%';
    SELECT * FROM Customers WHERE CustomerName LIKE '__a';
    ```

- **ORDER BY**
  - Sorts the query result by specified columns.
  - **Syntax:**

```
SELECT product_name, price FROM products ORDER BY price DESC;
```

- **GROUP BY**
  - Groups rows based on one or more columns, often used with aggregate functions.
  - **Syntax:**

```
SELECT department, AVG(salary) FROM employees GROUP BY department;
```

---

3. Data Manipulation Language (DML)

DML (Data Manipulation Language) involves commands to modify data within the database.

Key DML Commands:

- **INSERT**
  - Adds new records to a table.
  - **Syntax:**

```
INSERT INTO employees (first_name, last_name, salary) VALUES ('John', 'Doe', 50000);
```

- **UPDATE**
  - Modifies existing records in a table.
  - **Syntax:**

```
UPDATE employees SET salary = 55000 WHERE first_name = 'John';
```

- **DELETE**
  - Removes records from a table.
  - **Syntax:**

```
DELETE FROM employees WHERE last_name = 'Doe';
```

---

4. Data Control Language (DCL)

DCL (Data Control Language) focuses on access rights and database security.

Key DCL Commands:

- **GRANT**
  - Provides privileges to users or roles.
  - **Syntax:**

    GRANT SELECT ON **Employees** TO **Analyst;**
- **REVOKE**
  - Removes privileges from users or roles.
  - **Syntax:**

    REVOKE SELECT ON **Employees** FROM **Analyst;**

Transaction Control Language (TCL)

Transaction Control Language (TCL) deals with the management of transactions within a database.
TCL commands are used to control the initiation, execution, and termination of transactions, which are sequences of one or more MYSQL statements that are executed as a single unit of work.
Transactions ensure data consistency, integrity, and reliability in a database by grouping related operations together and either committing or rolling back changes based on the success or failure of those operations.

There are three main TCL commands in MYSQL: **COMMIT**, **ROLLBACK**, and **SAVEPOINT**.

1. COMMIT

The COMMIT command is used to permanently save the changes made during a transaction.

- It makes all the changes applied to the database since the last COMMIT or ROLLBACK command permanent.
- Once a COMMIT is executed, the transaction is considered successful, and the changes are made permanent.

Example:

Committing changes made during a transaction:

```
UPDATE Employees
SET Salary = Salary * 1.10
WHERE Department = 'Sales';
COMMIT;
```

---

2. ROLLBACK

The ROLLBACK command is used to undo changes made during a transaction.
- It reverts all the changes applied to the database since the transaction began.
- ROLLBACK is typically used when an error occurs during the execution of a transaction, ensuring that the database remains in a consistent state.

EXAMPLE:

Rolling back changes due to an error during a transaction:

```
BEGIN;
UPDATE Inventory
SET Quantity = Quantity - 10
WHERE ProductID = 101;

-- An error occurs here
ROLLBACK;
```

---

3. SAVEPOINT

The SAVEPOINT command creates a named point within a transaction, allowing you to set a point to which you can later ROLLBACK if needed.

- SAVEPOINTs are useful when you want to undo part of a transaction while preserving other changes.

SYNTAX:

```
SAVEPOINT savepoint_name;
```

EXAMPLE:

Using SAVEPOINT to create a point within a transaction:

```
BEGIN;
UPDATE Accounts
SET Balance = Balance - 100
WHERE AccountID = 123;

SAVEPOINT before_withdrawal;

UPDATE Accounts
SET Balance = Balance + 100
WHERE AccountID = 456;

-- An error occurs here
ROLLBACK TO before_withdrawal;

-- The first update is still applied
COMMIT;
```

JOINS

In a DBMS, a join is an operation that combines rows from two or more tables based on a related column between them.
Joins are used to retrieve data from multiple tables by linking them together using a common key or column.

Types of Joins

1. Inner Join
2. Outer Join
3. Cross Join
4. Self Join
5. Natural Join

---

1. Inner Join

An inner join combines data from two or more tables based on a specified condition, known as the join condition.

- The result of an inner join includes only the rows where the join condition is met in all participating tables.
- It filters out non-matching rows and returns only rows with matching values in both tables.

Syntax:

```
SELECT columns FROM table1 INNER JOIN table2
ON table1.column = table2.column;
```

- **columns**: Specific columns to retrieve from the tables.
- **table1** and **table2**: Names of the tables you are joining.
- **column**: Common column used to match rows between the tables.
- The **ON** clause specifies the join condition.

Example:

Customers Table:

CustomerID	CustomerName
1	Priyansh
2	Parth
3	Keyur
4	Other

Orders Table:

OrderID	CustomerID	Product
101	1	Laptop
102	3	Smartphone
103	2	Headphones
104		Something

Query:

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
```

```
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
Priyansh	Laptop
Parth	Headphones
Keyur	Smartphone

---

2. Outer Join

Outer joins combine data from two or more tables based on a specified condition, including rows that do not have matching values in both tables.

Types:

1. **Left Outer Join (Left Join)**: Returns all rows from the left table and matching rows from the right table.
2. **Right Outer Join (Right Join)**: Returns all rows from the right table and matching rows from the left table.
3. **Full Outer Join (Full Join)**: Returns all rows from both tables, including matches and non-matches.

Example: Left Outer Join

Query:

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
Priyansh	Laptop
Parth	Headphones

CustomerName	Product
Keyur	Smartphone
Other	NULL

Example: Right Outer Join

Query:

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
RIGHT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
Priyansh	Laptop
Keyur	Smartphone
Parth	Headphones
NULL	Something

Example: Full Outer Join

Query:

```
SELECT Customers.CustomerName, Orders.Product
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Result:

CustomerName	Product
Priyansh	Laptop
Parth	Headphones

CustomerName	Product
Keyur	Smartphone
NULL	Something
Other	NULL

---

3. Cross Join

A cross join, or Cartesian product, combines every row from one table with every row from another table.
- It generates all possible combinations of rows from both tables.

Syntax:

```
SELECT columns FROM table1 CROSS JOIN table2;
```

Example:

Students Table:

StudentID	StudentName
1	Parth
2	Keyur

Courses Table:

CourseID	CourseName
101	Maths
102	Science

Query:

```
SELECT Students.StudentName, Courses.CourseName
FROM Students
CROSS JOIN Courses;
```

Result:

StudentName	CourseName
Parth	Maths
Parth	Science
Keyur	Maths
Keyur	Science

---

4. Self Join

A self join involves joining a table with itself, often used to find relationships within the same table.

Syntax:

```
SELECT columns
FROM table1 AS alias1
JOIN table1 AS alias2
ON alias1.column = alias2.column;
```

Example:

Employees Table:

EmployeeID	EmployeeName	ManagerID
1	Priyansh	3
2	Keyur	3
3	Pinal	4
4	Dr. Viral	NULL

Query:

```
SELECT e1.EmployeeName AS Employee, e2.EmployeeName AS Manager
FROM Employees AS e1
```

```
JOIN Employees AS e2
ON e1.ManagerID = e2.EmployeeID;
```

RESULT:

EMPLOYEE	MANAGER
PRIYANSH	PINAL
KEYUR	PINAL
PINAL	DR. VIRAL

SET OPERATIONS

Set operations in MYSQL are used to combine or manipulate the result sets of multiple **SELECT** queries. They allow you to perform operations similar to those in set theory, such as union, intersection, and difference, on the data retrieved from different tables or queries.

Set operations provide powerful tools for managing and manipulating data, enabling you to analyze and combine information in various ways.

There are four primary set operations in MYSQL:

- **UNION**
- **INTERSECT**
- **UNION ALL**

---

1. UNION

The **UNION** operator combines the result sets of two or more **SELECT** queries into a single result set. It removes duplicates by default, meaning that if there are identical rows in the result sets, only one instance of each row will appear in the final result.

EXAMPLE

Assume we have two tables: **Customers** and **Suppliers**.

Customers Table:

CustomerID	CustomerName
1	Parth
2	Keyur

Suppliers Table:

SupplierID	SupplierName
101	ABC
102	XYZ

UNION Query:

```
SELECT CustomerName
FROM Customers
UNION
SELECT SupplierName
FROM Suppliers;
```

Result:

CustomerName
Parth
Keyur
ABC
XYZ

---

2. INTERSECT

The **INTERSECT** operator returns the common rows that exist in the result sets of two or more **SELECT** queries. It only returns distinct rows that appear in all result sets.

Example

Using the same tables as before:

```
SELECT CustomerName
FROM Customers
INTERSECT
SELECT SupplierName
FROM Suppliers;
```

Result:

CustomerName
(Empty Set)

---

3. UNION ALL

The **UNION ALL** operator performs the same function as the **UNION** operator but does not remove duplicates from the result set. It simply concatenates all rows from the different result sets.

Example

Using the same tables as before:

```
SELECT CustomerName
FROM Customers
UNION ALL
SELECT SupplierName
FROM Suppliers;
```

Result:

CustomerName
Parth
Keyur
ABC

CustomerName
XYZ

SUBQUERY

Subqueries, also known as nested queries or inner queries, allow you to use the result of one query (the inner query) as the input for another query (the outer query). Subqueries are often used to retrieve data that will be used for filtering, comparison, or calculation within the context of a larger query.

They are a way to break down complex tasks into smaller, manageable steps.

Syntax

```
SELECT columns
FROM table
WHERE column OPERATOR (
    SELECT column
    FROM table
    WHERE condition
);
```

Example

Consider two tables: **Products** and **Orders**.

Products Table:

ProductID	ProductName	Price
1	Laptop	1000
2	Smartphone	500
3	Headphone	50

Orders Table:

OrderID	ProductID	Quantity
101	1	2
102	3	1

Query: Retrieve the product names and quantities for orders with a total cost greater than the average price of all products.

```
SELECT ProductName, Quantity
FROM Products
WHERE Price * Quantity > (
    SELECT AVG(Price)
    FROM Products
);
```

Result:

ProductName	Quantity
Laptop	2

Backup and Restore Guide

Backup and restore processes are essential for maintaining data integrity, minimizing downtime, and ensuring business continuity in case of data loss or corruption. Below is a detailed guide explaining the process, its pros and cons, types, and step-by-step instructions.

What is Backup and Restore?

- **Backup:** The process of creating copies of data to prevent data loss in case of failure, corruption, or disasters.
- **Restore:** The process of retrieving data from a backup to bring the system back to a previous state.

Importance of Backup and Restore

1. **Data Security:** Protects against accidental deletion, corruption, or loss.

2. **Business Continuity:** Minimizes downtime by ensuring data recovery.
3. **Compliance:** Meets regulatory requirements for data protection.
4. **Disaster Recovery:** Enables recovery after natural disasters or cyberattacks.

Steps for Backup and Restore

Backup Process

1. **Open MySQL Workbench**
2. Go to **Server > Data Export**.
3. **Select Database [from right panel].**
4. Set the export location under **Export to Dump Project Folder** or **Export to Self-Contained File**.
5. Click the **Start Export** button.
6. Check the specified folder to ensure the `.sql` dump file is created.

Restore Process

1. **Open MySQL Workbench**
2. **Server > Data Import**.
3. **Import from Self-Contained File** if restoring from a `.sql` file. Browse and select the backup file.
4. **Choose the target database for the import. If it doesn't exist, create a new database first.**
5. Click the **Start Import** button**.**
6. Check the target database to ensure the data is restored.

EXPLAIN Keyword in MySQL

The **EXPLAIN** keyword in MySQL is a powerful tool used to understand and optimize query performance. It provides insight into how the MySQL query optimizer executes a query, allowing developers and database administrators to identify inefficiencies and improve execution times.

Why Use EXPLAIN?

1. **Query Optimization:** Understand how MySQL executes a query and identify potential performance bottlenecks.

2. **Index Utilization:** Check if the query uses indexes effectively.
3. **Join Analysis:** Examine how tables are joined and ensure joins are optimized.
4. **Execution Plan:** Analyze the step-by-step process of query execution.

Syntax

```
EXPLAIN SELECT ...;
```

OR

```
EXPLAIN [EXTENDED] SELECT ...;
```

Parameters:

- **SELECT ...**: The MYSQL query to analyze.
- **EXTENDED**: Provides additional details about query execution.

Understanding EXPLAIN Output

When you run the **EXPLAIN** command, it returns a result set with the following columns:

1. id

- The sequence number of the query step.
- Higher numbers are executed later in the query plan.

2. select_type

- The type of query, such as **SIMPLE**, **PRIMARY**, **UNION**, **DEPENDENT SUBQUERY**, etc.
- **Examples:**
  - **SIMPLE**: A straightforward SELECT query without subqueries or unions.
  - **PRIMARY**: The main query in a subquery or join.

3. table

- The table being accessed in this step of the query.

4. type

- The type of access being used.

- **Examples:**
  - **ALL**: Full table scan (inefficient).
  - **index**: Full index scan.
  - **ref**: Join query using references.
  - **eq_ref**: Primary key or unique key lookup.

---

5. **POSSIBLE_KEYS**
   - The indexes that MySQL can use to execute the query.

---

6. **KEY**
   - The actual index used by MySQL to process the query.

---

7. **KEY_LEN**
   - The length of the key used.

---

8. **REF**
   - The column or constant being compared to the index.

---

9. **ROWS**
   - The estimated number of rows MySQL will examine.

---

10. **FILTERED**
    - The percentage of rows that will be filtered by the condition.

---

11. **EXTRA**
    - Additional information about the query execution.
    - **Examples:**
      - **Using index**: Query retrieves data directly from the index.
      - **Using where**: A WHERE clause is used to filter results.
      - **Using temporary**: A temporary table is used.
      - **Using filesort**: MySQL performs an extra sort.

EXAMPLE

Sample Query:

```
EXPLAIN SELECT * FROM Employees WHERE Department = 'Sales';
```

Sample Output:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Employees	ref	dept_index	dept_index	10	const	50	100.00	Using where

Explanation:

- **type: ref** indicates the query uses an index to filter rows.
- **key:** The index **dept_index** is being used.
- **rows:** MySQL estimates that 50 rows will match.
- **Extra: Using where** indicates filtering happens at this step.

Tools for Advanced Query Analysis

1. **EXPLAIN EXTENDED:**
   - Provides additional details about the query plan.
   - Use with **SHOW WARNINGS** to see rewritten queries.
2. **ANALYZE FORMAT=JSON:**
   - Provides a detailed and structured JSON output of the execution plan.
   - Syntax: **EXPLAIN ANALYZE FORMAT=JSON SELECT ...;**

By using **EXPLAIN**, you can gain a deeper understanding of query behavior, identify inefficiencies, and make informed decisions to optimize your MySQL queries.