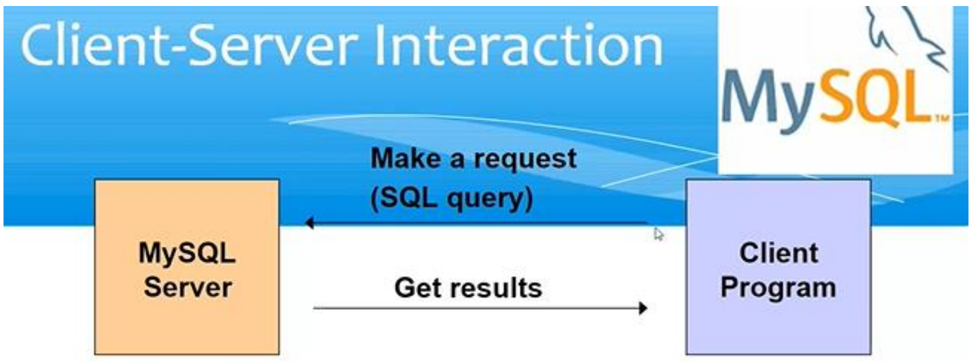
# Day 2 Assignment: Amaan Shaikh

## Overview of Topics Discussed

**Components of RDBMS:** Users → DBMS → Database

**Types:** Relational, Operational, Distributed, Data Warehouse, End-User DB



**Client program can be a MySQL command line client, GUI client, or a program written in any language such as C, Perl, PHP, Java that has an interface to the MySQL server.**

**Why MySQL**: Web Applications use LAMP Stack for development. MySQL is used for storage and logging of data. Client-server model.

**L** - Linux, **A** - Apache HTTP Server, **M** - MySQL, **P** - PHP or Python.

- Standardised Language/Syntax
- Ease of Use
- Powerful Queries - with integrity and consistency constraints.
- Integratable and Scalable.

**Port no**: 3306

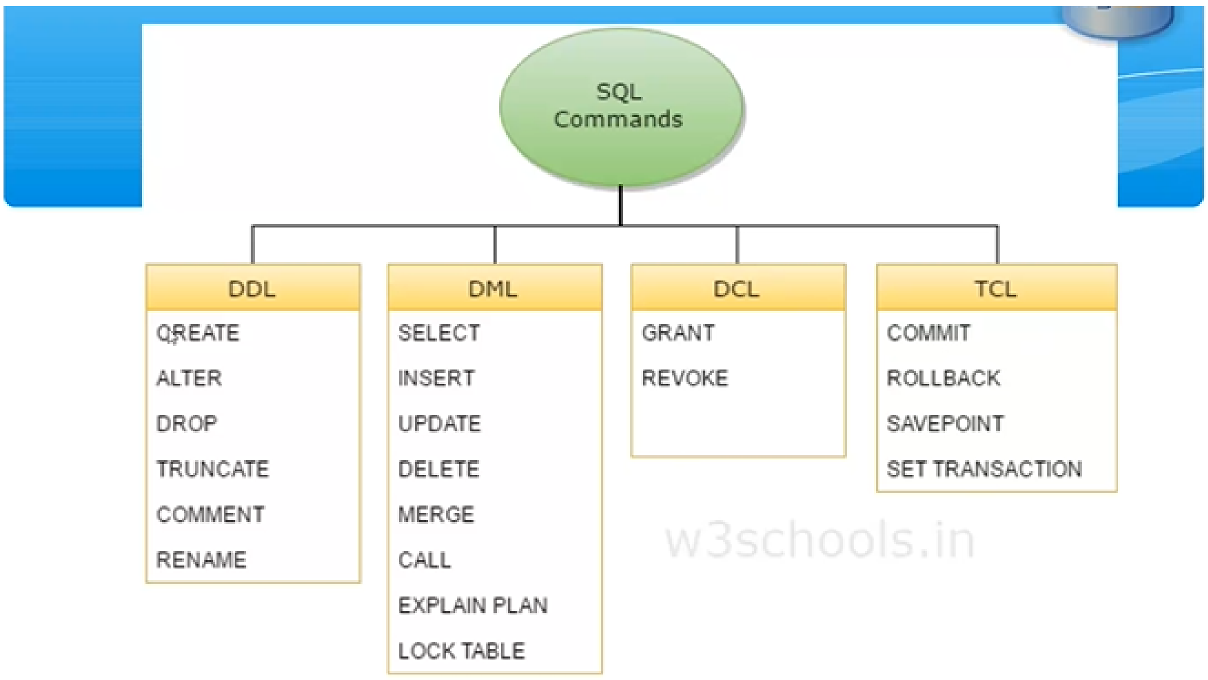
**RAM :** 1 MB

**Connectors** - ODBC, Net, J, Python, C, C++

**Storage Engine** - A **storage engine** in MySQL is the software component responsible for managing how data is stored, retrieved, and updated in a database. It determines the specific way data is handled on disk, including support for transactions, indexing, locking, and other database features. MySQL allows you to choose different storage engines (like InnoDB, MyISAM, etc.) depending on your performance and data integrity requirements.

**Tables in MySQL** - InnoDB, MyISAM, Heap, Archive tables, NDB, CSV.

## SQL Commands Summary:



Data Types: **Integer:** INT, TINYINT, SMALLINT, MEDIUMINT, BIGINT

**Decimal:** DECIMAL, FLOAT, REAL, DOUBLE

**Chars:** CHAR, VARCHAR, TEXT

**Binary:** BINARY, VARBINARY

**Date and Time**: DATE, DATETIME, TIMESTAMP, TIME, YEAR

Other: ENUM, SET, BOOLEAN.

## TCL (Transaction Control Language) Commands

TCL commands are used to manage transactions in SQL databases. They ensure the integrity and consistency of the database. **They are used ONLY with DML Commands.**

**1.COMMIT**:

- **Purpose**: Saves all changes made during the current transaction.
- BEGIN TRANSACTION;
- INSERT INTO Employees (EmpID, FirstName, LastName) VALUES (1, 'John', 'Doe');
- COMMIT;

**2.ROLLBACK**:

- **Purpose**: Undoes all changes made during the current transaction.
- BEGIN TRANSACTION;
- INSERT INTO Employees (EmpID, FirstName, LastName) VALUES (1, 'John', 'Doe');
- ROLLBACK;

**3.SAVEPOINT**:

- **Purpose**: Sets a point within a transaction to which you can later roll back.
- BEGIN TRANSACTION;
- INSERT INTO Employees (EmpID, FirstName, LastName) VALUES (1, 'John', 'Doe');
- SAVEPOINT Savepoint1;
- INSERT INTO Employees (EmpID, FirstName, LastName) VALUES (2, 'Jane', 'Smith');
- ROLLBACK TO Savepoint1;
- COMMIT;

**4.SET TRANSACTION**:

- **Purpose**: Configures the transaction properties such as isolation level.
- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
- BEGIN TRANSACTION;

- INSERT INTO Employees (EmpID, FirstName, LastName) VALUES (1, 'John', 'Doe');
- COMMIT;

**Set operators** – Union, Intersect, Minus, Union All

(Number and order of columns must be same, Data Types must be compatible)

| **Operator** | **Purpose** | **Duplicates** | **Example** |
|---|---|---|---|
| UNION | Combines results and removes duplicates | No | SELECT FirstName FROM Employees UNION SELECT FirstName FROM Contractors; |
| UNION ALL | Combines results including duplicates | Yes | SELECT FirstName FROM Employees UNION ALL SELECT FirstName FROM Contractors; |
| INTERSECT | Returns common results between queries | No | SELECT FirstName FROM Employees INTERSECT SELECT FirstName FROM Contractors; |
| MINUS (or EXCEPT in SQL Server) | Returns rows from the first query not in the second | No | SELECT FirstName FROM Employees EXCEPT SELECT FirstName FROM Contractors; |

## DCL (Data Control Language) Commands

DCL commands are used to control access to data in the database. They manage permissions and user access. Here are the primary DCL commands with examples:

**1. GRANT**:

**Purpose**: Gives specific privileges to users or roles.

**Example**:
GRANT SELECT, INSERT ON Employees TO User1;

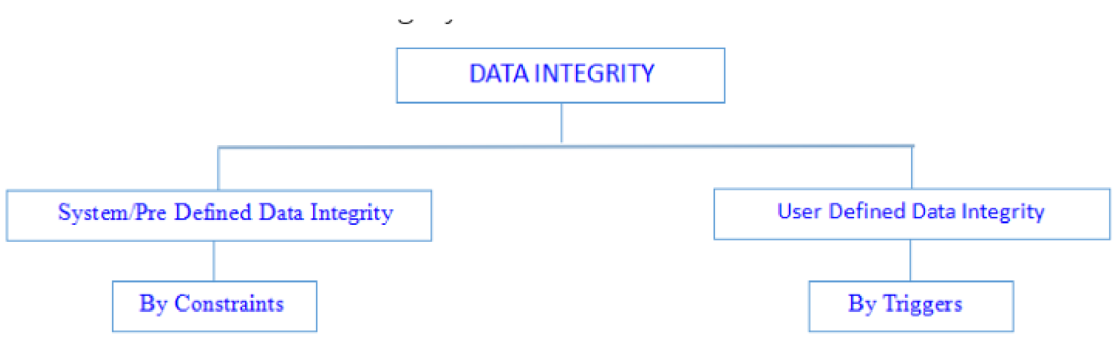
**2. REVOKE**:

**Purpose**: Removes specific privileges from users or roles.

**Example**:
REVOKE SELECT, INSERT ON Employees FROM User1;

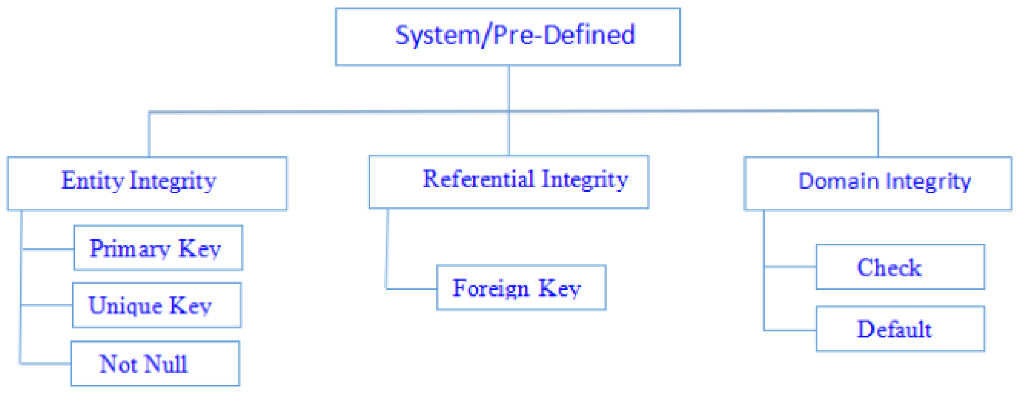
## Data Integrity in SQL:

**Data integrity** in a database ensures that the data is accurate, consistent, and reliable.


DATA INTEGRITY
System/Pre Defined Data Integrity
User Defined Data Integrity
By Constraints
By Triggers

### System/Pre-Defined Integrity

System or pre-defined integrity is enforced by the database management system (DBMS) itself. These are built-in rules and constraints provided by the DBMS to maintain data accuracy and consistency. Key components include:



1. **Entity Integrity**:
   - **Definition**: Ensures that each row in a table is uniquely identifiable.
   - **Implementation**: Achieved through primary keys.
   - **Example**: A table Employees has a column EmpID which is the primary key and must be unique for each employee.
2. **Referential Integrity**:
   - **Definition**: Ensures that relationships between tables are maintained consistently.
   - **Implementation**: Enforced using foreign keys.

- **Example**: In a Orders table, a CustomerID column might be a foreign key that references the CustomerID in the Customers table.

3. **Domain Integrity**:
    - **Definition**: Ensures that data entered into a column falls within a specific range or set of values.
    - **Implementation**: Enforced using data types, constraints, and default values.
    - **Example**: A BirthDate column must have a data type of DATE and should not accept values outside valid date ranges.

**User-Defined Integrity**

User-defined integrity involves rules and constraints created by database users or administrators to meet specific business requirements or rules that are not covered by system integrity. It includes:

**1.Custom Constraints**:

- **Definition**: Constraints defined by users to enforce business rules.
- **Implementation**: Created using CHECK constraints.

**Example**: A CHECK constraint ensures that the Salary column in the Employees table must be greater than zero:

```
ALTER TABLE Employees

ADD CONSTRAINT chk_salary CHECK (Salary > 0);
```

**2. Triggers**:

- **Definition**: Automated actions performed in response to specific events on a table (e.g., insert, update, delete).
- **Implementation**: Created using TRIGGER statements.

**3. Stored Procedures**:

- **Definition**: Predefined SQL code that can be executed as needed, often used to enforce complex business rules.
- **Implementation**: Created using CREATE PROCEDURE.

**Math Functions**

1. **ABS(x)**: Returns the absolute value of x.
    - **Example**: SELECT ABS(-10); returns 10.
2. **CEILING(x)**: Returns the smallest integer greater than or equal to x.
    - **Example**: SELECT CEILING(4.2); returns 5.
3. **FLOOR(x)**: Returns the largest integer less than or equal to x.
    - **Example**: SELECT FLOOR(4.7); returns 4.
4. **ROUND(x, d)**: Rounds x to d decimal places.
    - **Example**: SELECT ROUND(123.4567, 2); returns 123.46.
5. **POWER(x, y)**: Returns x raised to the power of y.
    - **Example**: SELECT POWER(2, 3); returns 8.
6. **SQRT(x)**: Returns the square root of x.
    - **Example**: SELECT SQRT(16); returns 4.
7. **LOG(x)**: Returns the natural logarithm of x.
    - **Example**: SELECT LOG(10); returns approximately 2.3026.
8. **EXP(x)**: Returns e raised to the power of x.
    - **Example**: SELECT EXP(1); returns approximately 2.7183.

**String Functions**

1. **CONCAT(string1, string2, ...)**: Concatenates multiple strings into one.
    - **Example**: SELECT CONCAT('Hello', ' ', 'World'); returns 'Hello World'.
2. **LEN(string)**: Returns the length of string.
    - **Example**: SELECT LEN('Hello'); returns 5.
3. **SUBSTRING(string, start, length)**: Returns a substring from string starting at start position with length.
    - **Example**: SELECT SUBSTRING('Hello World', 1, 5); returns 'Hello'.
4. **UPPER(string)**: Converts all characters in string to uppercase.
    - **Example**: SELECT UPPER('hello'); returns 'HELLO'.
5. **LOWER(string)**: Converts all characters in string to lowercase.
    - **Example**: SELECT LOWER('HELLO'); returns 'hello'.
6. **TRIM(string)**: Removes leading and trailing spaces from string.

   - **Example**: SELECT TRIM(' Hello '); returns 'Hello'.

7. **REPLACE(string, old_substring, new_substring)**: Replaces occurrences of old substring with new substring in string.
   - **Example**: SELECT REPLACE('Hello World', 'World', 'SQL'); returns 'Hello SQL'.
8. REVERSE(string): Reverses the order of characters in the string.
   - **Example**: SELECT REVERSE(‘Hello’); returns ‘olleH’.

**Date Functions**

1. **GETDATE()**: Returns the current date and time.
   - **Example**: SELECT GETDATE(); might return 2024-08-20 15:45:00.000.
2. **DATEADD(datepart, number, date)**: Adds a specified number of date parts (e.g., days, months) to a date.
   - **Example**: SELECT DATEADD(day, 10, '2024-08-20'); returns 2024-08-30.
3. **DATEDIFF(datepart, startdate, enddate)**: Returns the difference between two dates in specified date parts.
   - **Example**: SELECT DATEDIFF(day, '2024-08-20', '2024-08-30'); returns 10.
4. **FORMAT(date, format)**: Returns the date formatted according to the specified format.
   - **Example**: SELECT FORMAT(GETDATE(), 'yyyy-MM-dd'); returns 2024-08-20.
5. **YEAR(date)**: Returns the year part of the date.
   - **Example**: SELECT YEAR('2024-08-20'); returns 2024.
6. **MONTH(date)**: Returns the month part of the date.
   - **Example**: SELECT MONTH('2024-08-20'); returns 8.
7. **DAY(date)**: Returns the day part of the date.
   - **Example**: SELECT DAY('2024-08-20'); returns 20.
8. **DATEPART(datepart, date)**: Returns a specific part of the date (e.g., year, month, day).
   - **Example**: SELECT DATEPART(year, '2024-08-20'); returns 2024.

**ORDER BY**

- **Purpose**: Sorts the result set of a query by one or more columns.
- **Examples**:
    - **Ascending Order**: SELECT * FROM employees ORDER BY salary ASC;
    - **Descending Order**: SELECT * FROM employees ORDER BY salary DESC;
    - **Multiple Columns**: SELECT * FROM employees ORDER BY department, salary DESC;

**GROUP BY**

- **Purpose**: Groups rows that have the same values into summary rows, often used with aggregate functions.
- **Examples**:
    - **Basic Grouping**: SELECT department, COUNT(*) FROM employees GROUP BY department;
    - **Multiple Columns**: SELECT department, job_title, AVG(salary) FROM employees GROUP BY department, job_title;

**HAVING**

- **Purpose**: Filters groups based on aggregate functions, used in conjunction with GROUP BY.
- **Examples**:
    - **Basic Filtering**: SELECT department, COUNT(*) FROM employees GROUP BY department HAVING COUNT(*) > 5;
    - **Complex Condition**: SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;

**Aggregation Functions**

- **Purpose**: Perform calculations on a set of values to return a single value, often used with GROUP BY.
- **Common Functions**:
    - **COUNT(column)**: Returns the number of rows (or non-null values) in a column.
        - **Example**: SELECT COUNT(*) FROM employees;
    - **SUM(column)**: Returns the sum of values in a column.
        - **Example**: SELECT SUM(salary) FROM employees;

- **AVG(column)**: Returns the average value of a column.
  - **Example**: SELECT AVG(salary) FROM employees;
- **MIN(column)**: Returns the minimum value in a column.
  - **Example**: SELECT MIN(salary) FROM employees;
- **MAX(column)**: Returns the maximum value in a column.
  - **Example**: SELECT MAX(salary) FROM employees;

**COMPUTE Clause**

The COMPUTE clause is used to generate summary reports by applying aggregate functions such as SUM, AVG, COUNT, etc., on a set of result rows. However, it is important to note that the COMPUTE clause is deprecated in modern versions of SQL Server and should generally be replaced with GROUP BY.

**Example**:

```
SELECT DepartmentID, Salary

FROM Employees

ORDER BY DepartmentID

COMPUTE SUM(Salary) BY DepartmentID;
```