# MySQL Query Rules

## Format

* **SELECT**
  + Fields
* **FROM**
  + Table Name (Alias)
* **JOIN**
* **WHERE**
  + Condition
* **GROUP BY**
  + Fields
* **ORDER BY**
  + Fields

## Data Duplication

* Verify columns to avoid duplication.
* Ensure no duplicate data columns.

## Column Alias

* Use column aliases **only when necessary**.

## Table Join

* Use columns that belong to a primary key or index.
* Ensure columns have the **same data type**.
* Use joins **only when necessary**.

## WHERE Clause

* Use primary key or index column **first**.
* Use date ranges with BETWEEN in **brackets**.
* Sequence columns from the **largest subset to the smallest subset**.

## ORDER BY Clause

* Use **only when necessary**.
* Use only the **required columns**.

## SELECT with Star (SELECT \*)

* **Avoid using SELECT \***.
* Avoid COUNT(\*); instead, use COUNT with a key column.

## Unique Row

* Use LIMIT 1 to ensure a **unique row**.

## Function with WHERE

* Use functions in the WHERE clause **only when necessary**.

## Explain with SELECT

* Use EXPLAIN and check the output for:
  + Missing keys
  + Row filters
  + Nested loops

## Find\_In\_Set()

* Avoid using FIND\_IN\_SET() functions.

## Session Variable

* Avoid using @Session variables; use local variables instead.
* Use the naming format: v\_variableName.

## Variable Name

* Variable names must start with **small ‘v’** followed by an underscore (\_).
  + Example: v\_variable
* Assign default values if possible.
  + Example: v\_variable INT DEFAULT 0

## Parameter Name

* Parameter names must start with **small ‘p’** followed by an underscore (\_).
  + Example: p\_parameter
* Use proper data types for parameters.

# Overview of DBMS

## What is a DBMS?

A **Database Management System (DBMS)** is a software system designed to define, create, manage, and manipulate data in a structured and efficient manner. It allows users and applications to interact with data stored in databases by providing tools for data retrieval, insertion, updating, and deletion.

### Components of DBMS

1. **Hardware**: Physical devices like servers and storage systems where data resides.
2. **Software**: DBMS software that manages the database.
3. **Data**: Organized collection of facts and information.
4. **Users**:
   * **End Users**: People who access the database for queries and reports.
   * **Database Administrators (DBAs)**: Experts managing the DBMS and ensuring data security.
   * **Application Developers**: Build applications that interact with the database.

## Features of DBMS

1. **Data Abstraction**: Hides the complexities of data storage and presents users with an abstract view.
2. **Data Integrity**: Ensures the accuracy and consistency of data.
3. **Data Security**: Provides mechanisms for protecting data from unauthorized access.
4. **Concurrency Control**: Supports multiple users accessing data simultaneously without conflicts.
5. **Backup and Recovery**: Ensures data safety and retrieval in case of failure.
6. **Data Independence**: Changes in database structure do not affect applications.

## Types of DBMS

1. **Relational DBMS (RDBMS)**: Data is stored in tables and uses SQL for operations (e.g., MySQL, PostgreSQL).
2. **Hierarchical DBMS**: Data is organized in a tree-like structure (e.g., IBM Information Management System).
3. **Network DBMS**: Uses a graph-like structure for more complex relationships (e.g., Integrated Data Store).
4. **NoSQL DBMS**: Handles unstructured or semi-structured data (e.g., MongoDB, Cassandra).

# MySQL

## What is MySQL?

**MySQL** is an open-source Relational Database Management System (RDBMS) that uses **Structured Query Language (SQL)**. It is popular for web applications, offering high performance, scalability, and compatibility.

## Key Features of MySQL

1. **High Performance**: Optimized for read-heavy workloads and fast query execution.
2. **Open Source**: Free to use under the GNU General Public License (GPL).
3. **Cross-Platform Support**: Available for Windows, Linux, and macOS.
4. **Data Security**: Provides robust mechanisms like authentication and encryption.
5. **Scalability**: Suitable for small to large-scale applications.
6. **Replication**: Supports master-slave replication for data redundancy.
7. **Transaction Support**: ACID-compliant transactions for reliable operations.

## Applications of MySQL

* **Web Development**: Powers platforms like WordPress, Drupal, and Joomla.
* **E-commerce**: Used in backends of platforms like Magento and OpenCart.
* **Data Warehousing**: Suitable for storing and analyzing massive datasets.
* **Mobile Applications**: Provides data management for cross-platform mobile apps.

## SQL Commands in MySQL

1. **Data Definition Language (DDL)**: For defining database structures.
   * CREATE, ALTER, DROP
2. **Data Manipulation Language (DML)**: For modifying data.
   * INSERT, UPDATE, DELETE, SELECT
3. **Data Control Language (DCL)**: For controlling access to the database.
   * GRANT, REVOKE
4. **Transaction Control Language (TCL)**: For managing transactions.
   * COMMIT, ROLLBACK, SAVEPOINT

## Sample SQL Query

*-- Retrieve all employees in the Sales department, ordered by their last name*  
**SELECT** first\_name, last\_name  
**FROM** employees  
**WHERE** department = 'Sales'  
**ORDER** **BY** last\_name;

# Database Design

## What is Database Design?

**Database Design** is the process of structuring and organizing data to efficiently store, retrieve, and manage it within a database. It involves creating a logical and physical blueprint of how data will be stored, accessed, and related to other data.

## Key Components of Database Design

1. **Requirement Analysis**:
   * Understand the data requirements of the system and the relationships between data entities.
2. **Conceptual Design**:
   * Create an **Entity-Relationship (ER) Diagram** to visualize entities, attributes, and relationships.
3. **Logical Design**:
   * Map entities to tables and define keys, attributes, and relationships.
4. **Normalization**:
   * Organize tables to reduce redundancy and dependency by following normal forms (1NF, 2NF, 3NF, etc.).
5. **Physical Design**:
   * Define the actual storage structure, indexes, and partitions based on database technology.

## Best Practices for Database Design

1. **Use Meaningful Table and Column Names**: Names should clearly define their purpose.
2. **Minimize Redundancy**: Apply normalization techniques to eliminate duplicate data.
3. **Define Primary and Foreign Keys**: Use keys to maintain data integrity.
4. **Optimize Indexing**: Create indexes for frequently queried columns to improve performance.
5. **Consider Scalability**: Design with future data growth in mind.

# SQL Basics

## What is SQL?

**Structured Query Language (SQL)** is a standardized programming language used to interact with relational databases. It allows users to create, read, update, and delete data efficiently.

## Key Categories of SQL Commands

1. **Data Definition Language (DDL)**: Defines and modifies database structure.
   * Commands: CREATE, ALTER, DROP, TRUNCATE
2. **Data Manipulation Language (DML)**: Manages data in the database.
   * Commands: INSERT, SELECT, UPDATE, DELETE
3. **Data Query Language (DQL)**: Retrieves data from databases.
   * Command: SELECT
4. **Data Control Language (DCL)**: Controls access to the database.
   * Commands: GRANT, REVOKE
5. **Transaction Control Language (TCL)**: Manages database transactions.
   * Commands: COMMIT, ROLLBACK, SAVEPOINT

## Sample SQL Query

*-- Retrieve all products with a price greater than 100*  
**SELECT** product\_name, price  
**FROM** products  
**WHERE** price > 100  
**ORDER** **BY** price **DESC**;

# Best Practices for Writing SQL Queries

1. **Use Descriptive Aliases**: Name columns or tables clearly using aliases for readability.
2. **Avoid SELECT \***: Specify only the required columns to optimize performance.
3. **Apply Filters First**: Use WHERE conditions to reduce the dataset before applying further operations.
4. **Use Joins Efficiently**: Ensure correct keys and data types are used in joins.
5. **Test Queries**: Check for performance issues using EXPLAIN or DESCRIBE tools.

# Data Sorting

## What is Data Sorting?

Data Sorting is the process of arranging rows in a specific order within a database query. Sorting is achieved using the ORDER BY clause in SQL, which organizes data in ascending (ASC) or descending (DESC) order.

## Syntax for Sorting

**SELECT** column\_name(s)  
**FROM** table\_name  
**ORDER** **BY** column\_name [**ASC**|**DESC**];

### Explanation:

* **ASC**: Sorts data in ascending order (default).
* **DESC**: Sorts data in descending order.

## Examples of Sorting

1. **Sort by One Column**:

* **SELECT** first\_name, last\_name  
  **FROM** employees  
  **ORDER** **BY** last\_name **ASC**;

1. **Sort by Multiple Columns**:

* **SELECT** first\_name, last\_name, department  
  **FROM** employees  
  **ORDER** **BY** department, last\_name **DESC**;
  + Sorts first by department (ascending by default), then by last\_name in descending order.

## Tips for Efficient Sorting

1. **Use Indexes**: Indexes on sorted columns can speed up sorting operations.
2. **Sort After Filtering**: Apply WHERE clauses before sorting to minimize the dataset.
3. **Limit Results**: Combine sorting with LIMIT to reduce query execution time for large datasets.

* **SELECT** \* **FROM** orders **ORDER** **BY** order\_date **DESC** **LIMIT** 10;

1. **Avoid Sorting on Large Text Columns**: Sorting on text columns (e.g., VARCHAR) can be resource-intensive; use numeric or indexed columns instead.

# Null Value & Keyword

## What is a NULL Value?

A **NULL value** in a database represents missing or unknown data. It indicates that a field does not have a value assigned yet. NULL is different from zero (0) or an empty string (''), as those are actual values.

## Key Points About NULL

1. **Default Behavior**: Fields without values are stored as NULL if allowed.
2. **Comparison**: NULL cannot be compared using standard operators like = or !=. Use the IS NULL or IS NOT NULL keywords.
3. **Mathematical Operations**: Any arithmetic operation involving NULL results in NULL.

## Example Queries

1. **Check for NULL Values**:

* **SELECT** first\_name, last\_name  
  **FROM** employees  
  **WHERE** middle\_name **IS** **NULL**;

1. **Exclude NULL Values**:

* **SELECT** first\_name, last\_name  
  **FROM** employees  
  **WHERE** middle\_name **IS** **NOT** **NULL**;

## Best Practices

1. Use DEFAULT values for fields to minimize NULL values when possible.
2. Avoid excessive use of NULLs in critical fields to ensure data integrity.
3. Handle NULLs explicitly in queries to avoid unexpected results.

# Auto Increment

## What is AUTO\_INCREMENT?

AUTO\_INCREMENT is a feature in databases that automatically generates unique numbers for a column whenever a new row is inserted. It is commonly used for primary keys.

## Syntax for AUTO\_INCREMENT

**CREATE** **TABLE** users (  
 user\_id INT AUTO\_INCREMENT **PRIMARY** **KEY**,  
 username VARCHAR(50) **NOT** **NULL**  
);

## Key Points About AUTO\_INCREMENT

1. **Automatic Value Assignment**: Automatically assigns a unique value starting from 1 (or a specified seed value).
2. **Primary Key Requirement**: AUTO\_INCREMENT columns are typically used as primary keys.
3. **Control the Starting Value**: Use the AUTO\_INCREMENT=100 clause to set the starting point.

## Example Queries

1. **Insert Data**:

* **INSERT** **INTO** users (username) **VALUES** ('JohnDoe');
  + The user\_id will be automatically generated.

1. **Reset AUTO\_INCREMENT**:

* **ALTER** **TABLE** users AUTO\_INCREMENT = 1000;

## Best Practices

1. Use AUTO\_INCREMENT only for unique identifiers, such as IDs.
2. Avoid manual updates to AUTO\_INCREMENT values to prevent conflicts.
3. Ensure no duplicate values by using constraints.

# DDL, DML, DCL, TCL, DQL

## 1. Data Definition Language (DDL)

### What is DDL?

DDL deals with the creation and modification of database objects like tables, indexes, and schemas.

### Key Commands with Examples

1. **CREATE**:  
   Create a new table in the database.

* **CREATE** **TABLE** employees (  
   employee\_id INT **PRIMARY** **KEY**,  
   name VARCHAR(50),  
   department VARCHAR(50)  
  );

1. **ALTER**:  
   Modify the structure of an existing table.

* **ALTER** **TABLE** employees **ADD** **COLUMN** joining\_date DATE;

1. **DROP**:  
   Delete a table or database object.

* **DROP** **TABLE** employees;

1. **TRUNCATE**:  
   Remove all rows from a table without logging individual deletions.

* **TRUNCATE** **TABLE** employees;

## 2. Data Manipulation Language (DML)

### What is DML?

DML is used for inserting, updating, deleting, and retrieving data from the database.

### Key Commands with Examples

1. **INSERT**:  
   Add new rows to a table.

* **INSERT** **INTO** employees (employee\_id, name, department, salary)  
  **VALUES** (1, 'John Doe', 'IT', 75000.00);

1. **UPDATE**:  
   Modify existing rows in a table.

* **UPDATE** employees  
  **SET** salary = 80000.00  
  **WHERE** employee\_id = 1;

1. **DELETE**:  
   Remove specific rows from a table.

* **DELETE** **FROM** employees  
  **WHERE** department = 'HR';

## 3. Data Query Language (DQL)

### What is DQL?

DQL focuses on data retrieval using the SELECT command.

### Key Command with Example

1. **SELECT**:  
   Retrieve specific columns or all columns from a table.

* **SELECT** name, department, salary  
  **FROM** employees  
  **WHERE** salary > 70000.00  
  **ORDER** **BY** salary **DESC**;

## 4. Data Control Language (DCL)

### What is DCL?

DCL manages user access and permissions in the database.

### Key Commands with Examples

1. **GRANT**:  
   Assign specific privileges to a user.

* **GRANT** **SELECT**, **INSERT** **ON** employees **TO** 'user1'@'localhost';

1. **REVOKE**:  
   Remove assigned privileges from a user.

* **REVOKE** **INSERT** **ON** employees **FROM** 'user1'@'localhost';

## 5. Transaction Control Language (TCL)

### What is TCL?

TCL manages database transactions, ensuring consistency and reliability.

### Key Commands with Examples

1. **COMMIT**:  
   Save changes made by the current transaction.

* **BEGIN**;  
  **UPDATE** employees **SET** salary = 90000.00 **WHERE** employee\_id = 2;  
  **COMMIT**;

1. **ROLLBACK**:  
   Undo changes made by the current transaction.

* **BEGIN**;  
  **UPDATE** employees **SET** salary = 65000.00 **WHERE** employee\_id = 3;  
  **ROLLBACK**;

1. **SAVEPOINT**:  
   Set a savepoint within a transaction.

* **BEGIN**;  
  **UPDATE** employees **SET** department = 'Finance' **WHERE** employee\_id = 4;  
  **SAVEPOINT** sp1;  
  **UPDATE** employees **SET** salary = 72000.00 **WHERE** employee\_id = 4;  
  **ROLLBACK** **TO** sp1;  
  **COMMIT**;

## Summary

* **DDL**: Defines the structure of the database.
* **DML**: Manipulates the data within the database.
* **DQL**: Retrieves data from the database.
* **DCL**: Controls access and permissions.
* **TCL**: Manages database transactions for consistency.

## 1. LIMIT

### What is LIMIT?

The LIMIT clause restricts the number of rows returned in a query result. It is especially useful when fetching a sample or paginated data.

### Key Points

* **Syntax**:
* **SELECT** column\_names **FROM** table\_name **LIMIT** number\_of\_rows OFFSET offset\_value;
  + OFFSET specifies the starting point for the rows to be returned.

### Example

1. Fetch the first 5 rows from the employees table:

* **SELECT** name, department, salary  
  **FROM** employees  
  **LIMIT** 5;

1. Fetch rows starting from the 6th row (pagination):

* **SELECT** name, department, salary  
  **FROM** employees  
  **LIMIT** 5 OFFSET 5;

## 2. Aggregate Functions

### What are Aggregate Functions?

Aggregate functions perform calculations on a set of values and return a single value. Common functions include SUM(), AVG(), COUNT(), MAX(), and MIN().

### Examples

1. Calculate the total salary:

* **SELECT** SUM(salary) **AS** total\_salary  
  **FROM** employees;

1. Find the average salary in the IT department:

* **SELECT** AVG(salary) **AS** average\_salary  
  **FROM** employees  
  **WHERE** department = 'IT';

1. Count the number of employees:

* **SELECT** COUNT(\*) **AS** total\_employees  
  **FROM** employees;

1. Find the highest and lowest salary:

* **SELECT** MAX(salary) **AS** highest\_salary, MIN(salary) **AS** lowest\_salary  
  **FROM** employees;

## 3. Sub-Queries

### What is a Sub-Query?

A sub-query is a query nested inside another query. It is used to perform operations that depend on the results of another query.

### Examples

1. Find employees with a salary higher than the average:

* **SELECT** name, salary  
  **FROM** employees  
  **WHERE** salary > (**SELECT** AVG(salary) **FROM** employees);

1. List departments with more than 10 employees:

* **SELECT** department  
  **FROM** employees  
  **WHERE** department **IN** (  
   **SELECT** department  
   **FROM** employees  
   **GROUP** **BY** department  
   **HAVING** COUNT(\*) > 10  
  );

## 4. Joins

### What are Joins?

Joins are used to combine rows from two or more tables based on a related column.

### Types of Joins and Examples

1. **INNER JOIN**: Returns rows with matching values in both tables.

* **SELECT** e.name, d.department\_name  
  **FROM** employees e  
  **INNER** **JOIN** departments d **ON** e.department\_id = d.department\_id;

1. **LEFT JOIN**: Returns all rows from the left table and matching rows from the right table.

* **SELECT** e.name, d.department\_name  
  **FROM** employees e  
  **LEFT** **JOIN** departments d **ON** e.department\_id = d.department\_id;

1. **RIGHT JOIN**: Returns all rows from the right table and matching rows from the left table.

* **SELECT** e.name, d.department\_name  
  **FROM** employees e  
  **RIGHT** **JOIN** departments d **ON** e.department\_id = d.department\_id;

1. **FULL OUTER JOIN**: Returns all rows when there is a match in either table.

* **SELECT** e.name, d.department\_name  
  **FROM** employees e  
  **FULL** **OUTER** **JOIN** departments d **ON** e.department\_id = d.department\_id;

## 5. Unions

### What is a UNION?

The UNION operator combines the results of two or more SELECT statements. By default, it removes duplicate rows.

### Examples

1. Combine two queries:

* **SELECT** name **FROM** employees  
  **UNION**  
  **SELECT** name **FROM** managers;

1. Retain duplicates using UNION ALL:

* **SELECT** name **FROM** employees  
  **UNION** **ALL**  
  **SELECT** name **FROM** managers;

## 6. Index

### What is an Index?

An index improves query performance by allowing faster retrieval of rows. It is created on columns to speed up searches.

### Examples

1. Create an index on the name column:

* **CREATE** **INDEX** idx\_name **ON** employees(name);

1. Remove an index:

* **DROP** **INDEX** idx\_name **ON** employees;

1. Query optimization using indexes:

* **SELECT** \* **FROM** employees **WHERE** name = 'John';

## 7. View

### What is a View?

A view is a virtual table created using a query. It does not store data but provides a way to simplify complex queries.

### Examples

1. Create a view to list high-salaried employees:

* **CREATE** **VIEW** high\_salary\_employees **AS**  
  **SELECT** name, department, salary  
  **FROM** employees  
  **WHERE** salary > 75000;

1. Query the view:

* **SELECT** \* **FROM** high\_salary\_employees;

1. Drop a view:

* **DROP** **VIEW** high\_salary\_employees;

## 8. Backup & Restore

### What is Backup & Restore?

Backup creates a copy of the database for recovery in case of data loss, while Restore retrieves the database from a backup.

### Examples

1. Backup a database:

* mysqldump -u root -p employees\_db > employees\_db\_backup.sql

1. Restore a database:

* mysql -u root -p employees\_db < employees\_db\_backup.sql

## 9. EXPLAIN Keyword

### What is EXPLAIN?

The EXPLAIN keyword provides details about how MySQL executes a query, including indexes, join types, and performance.

### Example

1. Analyze a query:

* **EXPLAIN** **SELECT** e.name, d.department\_name  
  **FROM** employees e  
  **INNER** **JOIN** departments d **ON** e.department\_id = d.department\_id;

### Output Details:

* **id**: Query sequence number.
* **select\_type**: Type of query (e.g., SIMPLE, SUBQUERY).
* **table**: Table being accessed.
* **type**: Join type (e.g., ALL, index).
* **key**: Index used.

## Best Practices

* Use LIMIT and indexes to optimize queries.
* Leverage aggregate functions for summaries.
* Use joins effectively to combine data.
* Regularly back up and test restore processes.