Mysql Database

1. Introduction to MySQL

What is MySQL?

- **Definition**: MySQL is a popular open-source relational database management system (RDBMS).
- **Key Features**:
  - Supports large databases with millions of rows.
  - ACID compliance for transactional support.
  - Scalable and cross-platform.
- **Use Cases**:
  - Web applications.
  - Data warehousing.
  - E-commerce.

---

2. MySQL Basics

2.1 Database Basics

- **Database**: A collection of related data.
- **Schema**: Structure of a database (tables, views, indexes).
- **Table**: A collection of rows and columns.

Basic Commands

- Create a database:

  ```
  CREATE DATABASE school;
  ```

- Use a database:

```
USE school;
```

- Drop a database:

  ```
  DROP DATABASE school;
  ```

2.2 Data Types

Common Data Types in MySQL:

1. **Numeric**:
   - INT: Integer values.
   - DECIMAL: Fixed-point numbers.
   - FLOAT: Floating-point numbers.
2. **String**:
   - VARCHAR: Variable-length string.
   - CHAR: Fixed-length string.
   - TEXT: Large text.
3. **Date and Time**:
   - DATE: Stores dates (YYYY-MM-DD).
   - DATETIME: Stores date and time (YYYY-MM-DD HH:MM:SS).

2.3 Creating Tables

- Syntax:

  ```
  CREATE TABLE table_name (
      column_name data_type constraints
  ```

```
);
```

- Example:

```
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    grade CHAR(1)
);
```

2.4 Inserting Data

- Syntax:

```
INSERT INTO table_name (column1, column2) VALUES (value
1, value2);
```

- Example:

```
INSERT INTO students (name, age, grade) VALUES ('Alice',
14, 'A');
```

2.5 Querying Data

- Select all records:

```
SELECT * FROM students;
```

- Select specific columns:

  ```
  sql
  Copy code
  SELECT name, age FROM students;
  ```

2.6 Updating Data

- Syntax:

  ```
  UPDATE table_name SET column_name = value WHERE conditio
  n;
  ```

- Example:

  ```
  sql
  Copy code
  UPDATE students SET grade = 'A' WHERE name = 'Bob';
  ```

2.7 Deleting Data

- Syntax:

  ```
  DELETE FROM table_name WHERE condition;
  ```

- Example:

  ```
  DELETE FROM students WHERE name = 'Alice';
  ```

---

3. Intermediate Queries

3.1 Constraints

- Constraints enforce rules on data in tables:
    - PRIMARY KEY: Uniquely identifies rows.
    - FOREIGN KEY: References another table.
    - UNIQUE: Ensures unique values.
    - NOT NULL: Disallows NULL values.

Example:

```
CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    student_id INT,
    FOREIGN KEY (student_id) REFERENCES students(id)
);
```

3.2 Joins

- Combine rows from two or more tables based on a related column.

Inner Join:

- Matches rows in both tables.

```
SELECT students.name, courses.course_name
FROM students
INNER JOIN courses ON students.id = courses.student_id;
```

Left Join:

- All rows from the left table and matched rows from the right table.

```
SELECT students.name, courses.course_name
FROM students
LEFT JOIN courses ON students.id = courses.student_id;
```

3.3 Aggregations

- COUNT, SUM, AVG, MAX, MIN.

Example:

```
SELECT COUNT(*) AS total_students FROM students;
SELECT AVG(age) AS average_age FROM students;
```

3.4 Grouping Data

- Group rows sharing a property.

Example:

```
SELECT grade, COUNT(*) AS student_count
FROM students
GROUP BY grade;
```

4. Advanced Queries

4.1 Subqueries

- Query inside another query.

Example:

```
SELECT name FROM students WHERE age = (SELECT MAX(age) FROM
students);
```

4.2 Views

- Virtual tables based on a query.

Example:

```
CREATE VIEW top_students AS
SELECT name, grade FROM students WHERE grade = 'A';
SELECT * FROM top_students;
```

4.3 Stored Procedures

- Predefined SQL code for reusability.

Example:

```
DELIMITER $$
CREATE PROCEDURE GetAllStudents()
BEGIN
    SELECT * FROM students;
END$$
DELIMITER ;
CALL GetAllStudents();
```

4.4 Triggers

- Execute code in response to events.

Example:

```
CREATE TRIGGER before_insert_students
BEFORE INSERT ON students
FOR EACH ROW
SET NEW.grade = 'B';
```

4.5 Transactions

- Control and manage changes in batches.

Example:

```
START TRANSACTION;
INSERT INTO students (name, age, grade) VALUES ('Chris', 1
6, 'A');
ROLLBACK; -- Undo the transaction
COMMIT;   -- Save the transaction
```

4.6 Indexing

- Improve query performance.

Example:

```
CREATE INDEX idx_name ON students(name);
```

4.7 Window Functions

- Perform calculations across rows.

Example:

```
SELECT name, age, RANK() OVER (ORDER BY age DESC) AS age_ra
nk
FROM students;
```

---

5. Real-World Examples

5.1 User Authentication

Tables:

```
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE,
    password VARCHAR(100)
);
```

Query:

```
SELECT * FROM users WHERE username = 'admin' AND password =
'password123';
```

5.2 E-Commerce Analytics

Top-selling product:

```
SELECT product_id, SUM(quantity) AS total_sold
FROM orders
GROUP BY product_id
ORDER BY total_sold DESC
```

```
LIMIT 1;
```

5.3 Active Users

Query:

```
SELECT COUNT(*) AS active_users
FROM users
WHERE last_login > NOW() - INTERVAL 30 DAY;
```