MySQL Objects

Tables

- The most fundamental (core) object in a RDB
- Stores data in
  - **columns** (attributes): “Name”, “Age”
  - **rows** (records): “Jessie”, “24”
- Each table has a **unique** name

```
CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    department VARCHAR(50),
    salary DECIMAL(10, 2)
);
```

Queries

- Queries are used to **interact** with the data stored in tables
- Retrieving data (**SELECT**)
- Inserting new data (**INSERT**)
- Updating existing data (**UPDATE**)
- Deleting data (**DELETE**)

```sql
-- SELECT
SELECT * FROM students;

-- INSERT
INSERT INTO students (name, age, grade) VALUES ('Alice', 20, 'A');

-- UPDATE
UPDATE students SET grade = 'B' WHERE name = 'Alice';

-- DELETE
DELETE FROM students WHERE id = 1;
```

- **Rules** enforced on data columns to ensure **data integrity**.
- Primary Key (**PRIMARY KEY**)
- Foreign Key (**FOREIGN KEY**)
- Unique (**UNIQUE**)
- Not Null (**NOT NULL**)

```
CREATE TABLE courses (
    course_id INT AUTO_INCREMENT PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL UNIQUE
);
```

Constraints

Views

- **Virtual** tables created by a query.
- Do **not** store data themselves but store the **query logic**

  information_schema.VIEWS
- Useful for **simplifying** queries, providing **security**, and presenting data in a **particular format**.

Note: A view always shows **up-to-date** data. The database engine recreates the view, every time a user queries it

```
CREATE VIEW high_salary_employees AS
SELECT name, department, salary
FROM employees
WHERE salary > 5000;
```

Indexes

- Indexes **improves** the **speed** of data retrieval
- When to use:
  - Frequent Search Conditions
  - Large Tables
  - Unique Constraints
  - Sorting and Grouping
  - Joins

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

- Useful for complex business logic and improving performance
- Can be invoked with a single call

```
DELIMITER //
CREATE PROCEDURE GetEmployeeCount()
BEGIN
    SELECT COUNT(*) AS total_employees FROM employees;
END //
DELIMITER ;
```

```
CALL GetEmployeeCount();
```

Stored Procedures

- For simple or complex **calculations** that return a single value
- Can be used in SQL statements wherever an expression is valid

```
DELIMITER //
CREATE FUNCTION CalculateBonus(salary DECIMAL(10, 2)) RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    RETURN salary * 0.10;
END //
DELIMITER ;
```

```
SELECT name, salary, CalculateBonus(salary) AS bonus FROM employees;
```

Functions

- Scheduled tasks that run at specified intervals.
- Used for **automating** maintenance tasks, **data archiving**, etc.

```
CREATE EVENT archive_old_records
ON SCHEDULE EVERY 1 MONTH
DO
DELETE FROM employees WHERE salary < 3000;
```

Events

Partitions

- Divide a table into **smaller**, more **manageable** pieces while still treating it as a single table.
- Can improve performance and simplify maintenance for large tables.

```
CREATE TABLE sales (
    id INT AUTO_INCREMENT PRIMARY KEY,
    sale_date DATE,
    amount DECIMAL(10, 2)
)
PARTITION BY RANGE (YEAR(sale_date)) (
    PARTITION p0 VALUES LESS THAN (2020),
    PARTITION p1 VALUES LESS THAN (2021),
    PARTITION p2 VALUES LESS THAN (2022)
);
```

```
SELECT * FROM sales PARTITION (p2);
```

Triggers

- Database objects that are **automatically executed** or fired when certain events occur (e.g., INSERT, UPDATE, DELETE).
- Used to enforce business rules, maintain data integrity, and audit changes.

```
CREATE TRIGGER before_employee_insert
BEFORE INSERT ON employees
FOR EACH ROW
SET NEW.salary = IFNULL(NEW.salary, 0);
```

- Users are accounts that can connect to the MySQL server.
- Roles are collections of privileges that can be assigned to users.
- Used for managing access control and security.

```
CREATE USER 'new_user'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT ON employees TO 'new_user'@'localhost';
```

Users and Roles