SQL Complete Learning Guide

Tamil & English Interview Preparation Notes

---

1. DDL (Data Definition Language) - தரவு வரையறை மொழி

Theory | கோட்பாடு

DDL என்பது தரவுத்தளம் மற்றும் அட்டவணை கட்டமைப்பை உருவாக்க, மாற்ற மற்றும் நீக்க பயன்படுகிறது.

DDL is used to create, modify, and delete database and table structures.

Commands | கட்டளைகள்

CREATE DATABASE

```sql
CREATE DATABASE company_db;
```

Tamil: புதிய தரவுத்தளம் உருவாக்குங்கள் **English:** Create a new database

CREATE TABLE

```sql
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    salary DECIMAL(10,2),
    department VARCHAR(30)
);
```

ALTER TABLE - ADD Column

```sql
ALTER TABLE employees ADD COLUMN age INT;
```

ALTER TABLE - MODIFY Column

```sql
ALTER TABLE employees MODIFY COLUMN salary DECIMAL(12,2);
```

ALTER TABLE - DROP Column

```
sql

ALTER TABLE employees DROP COLUMN age;
```

RENAME TABLE

```
sql

RENAME TABLE employees TO staff;
```

DROP TABLE

```
sql

DROP TABLE employees;
```

TRUNCATE TABLE

```
sql

TRUNCATE TABLE employees;
```

Constraints | கட்டுப்பாடுகள்

```
sql

CREATE TABLE students (
    student_id INT PRIMARY KEY,             -- Primary Key
    student_name VARCHAR(50) NOT NULL,       -- Not Null
    email VARCHAR(100) UNIQUE,               -- Unique
    age INT CHECK (age >= 18),               -- Check
    course_id INT,
    FOREIGN KEY (course_id) REFERENCES courses(course_id), -- Foreign Key
    DEFAULT registration_date = CURRENT_DATE -- Default
);
```

- **Primary Key (PK):** தனித்துவமான அடையாளம் | Unique identifier
- **Foreign Key (FK):** வேறு அட்டவணை குறிப்பு | Reference to another table
- **NOT NULL:** மதிப்பு கட்டாயம் | Value required
- **UNIQUE:** இரண்டு மொழி | No duplicate values
- **CHECK:** நிபந்தனை சரிபார்ப்பு | Validate values

- **DEFAULT:** முன்னிருப்பு மதிப்பு | Default value

---

2. DML (Data Manipulation Language) - தரவு நிர்வாக மொழி

Theory | கோட்பாடு

DML என்பது அட்டவணையில் தரவைச் சேர்க்க, புதுப்பிக்க மற்றும் நீக்க பயன்படுகிறது.

DML is used to insert, update, and delete data in tables.

INSERT

sql

```
-- Single row insert
INSERT INTO employees (emp_id, emp_name, salary, department)
VALUES (1, 'Raj Kumar', 50000, 'IT');

-- Multiple row insert
INSERT INTO employees VALUES
(2, 'Priya Singh', 55000, 'HR'),
(3, 'Amit Patel', 60000, 'Finance');
```

UPDATE

sql

```
-- Update specific records
UPDATE employees
SET salary = 65000
WHERE emp_id = 1;

-- Update multiple columns
UPDATE employees
SET salary = 70000, department = 'Management'
WHERE department = 'IT';

-- SAFE UPDATE (Enable)
SET SQL_SAFE_UPDATES = 1;  -- Always use WHERE clause
```

Important: Always use WHERE clause to avoid updating all records accidentally.

DELETE

sql

```
-- Delete specific records
DELETE FROM employees
WHERE emp_id = 1;

-- Delete with condition
DELETE FROM employees
WHERE salary < 30000;

-- SAFE DELETE (Enable)
SET SQL_SAFE_UPDATES = 1;  -- Always use WHERE clause
```

Warning: Always enable safe mode - அபாய சாதனம் | Always specify WHERE clause

Tasks - DML Practice | பயிற்சி

1. Insert 5 employees with different departments
2. Update all IT department salaries by 10%
3. Delete employees with salary < 40000
4. Insert employee without department
5. Update department where emp_id = 3
6. Delete all records (use TRUNCATE instead)
7. Update multiple columns for one employee
8. Insert batch of employees using SELECT
9. Delete with multiple conditions
10. Update salary with calculation (salary * 1.05)

---

3. DQL (Data Query Language) - தரவு வினா மொழி

Theory | கோட்பாடு

DQL என்பது அட்டவணையிலிருந்து தரவைக் கொண்டு வர பயன்படுகிறது.

DQL is used to retrieve data from tables.

SELECT - Basic

sql

```
-- All columns
SELECT * FROM employees;

-- Specific columns
SELECT emp_id, emp_name, salary FROM employees;

-- With alias
SELECT emp_name AS 'Employee Name', salary AS 'Salary Amount' FROM employees;
```

WHERE Clause

```sql
-- Comparison operators
SELECT * FROM employees WHERE salary > 50000;
SELECT * FROM employees WHERE department = 'IT';

-- Logical operators
SELECT * FROM employees WHERE salary > 50000 AND department = 'IT';
SELECT * FROM employees WHERE department = 'IT' OR department = 'HR';

-- IN operator
SELECT * FROM employees WHERE department IN ('IT', 'HR', 'Finance');

-- BETWEEN
SELECT * FROM employees WHERE salary BETWEEN 40000 AND 60000;

-- LIKE pattern
SELECT * FROM employees WHERE emp_name LIKE 'A%';  -- Starts with A
SELECT * FROM employees WHERE emp_name LIKE '%Kumar';  -- Ends with Kumar
```

Tasks - SELECT WHERE | பயிற்சி

1. Select all employees from IT department

2. Select employees earning more than 55000

3. Select employees with name starting with 'P'

4. Select employees from IT OR HR

5. Select employees earning between 40000-60000

6. Select all columns for emp_id = 5

7. Select with multiple conditions (AND, OR)

DISTINCT

```sql
-- Remove duplicates
SELECT DISTINCT department FROM employees;

-- Count unique departments
SELECT COUNT(DISTINCT department) FROM employees;
```

Tasks - DISTINCT | பயிற்சி

1. Find all unique departments
2. Count distinct job titles
3. Find unique combinations of department and salary range

ORDER BY - Sorting | வரிசைப்படுத்துதல்

```sql
-- Ascending (default)
SELECT * FROM employees ORDER BY salary ASC;

-- Descending
SELECT * FROM employees ORDER BY salary DESC;

-- Multiple columns
SELECT * FROM employees ORDER BY department ASC, salary DESC;
```

Tasks - ORDER BY | பயிற்சி

1. List employees sorted by salary highest to lowest
2. Sort by department ascending, then salary descending
3. Find top 5 highest paid employees using ORDER BY

LIMIT - Pagination | பக்க பிரிவு

```sql
```

```
-- First 5 records
SELECT * FROM employees LIMIT 5;

-- Skip first 2, get next 5
SELECT * FROM employees LIMIT 2, 5;  -- or OFFSET
SELECT * FROM employees LIMIT 5 OFFSET 2;

-- Top salaries
SELECT * FROM employees ORDER BY salary DESC LIMIT 3;
```

Tasks - LIMIT | பயிற்சி

1. Display first 10 employees
2. Skip first 5 records, show next 5
3. Show top 3 highest paid employees
4. Pagination: Get records 11-20

GROUP BY

```
sql

-- Count employees by department
SELECT department, COUNT(*) AS emp_count FROM employees GROUP BY department;

-- Average salary by department
SELECT department, AVG(salary) AS avg_salary FROM employees GROUP BY department;

-- Sum salary by department
SELECT department, SUM(salary) AS total_salary FROM employees GROUP BY department;

-- Multiple grouping
SELECT department, job_title, COUNT(*) FROM employees GROUP BY department, job_title;
```

Aggregate Functions | ஒருங்கிணைப்பு செயல்பாடுகள்

- **COUNT()** - எண்ணிக்கை | Number of records
- **SUM()** - மொத்தம் | Total sum
- **AVG()** - சராசரி | Average value
- **MAX()** - அதிகபட்சம் | Maximum value
- **MIN()** - குறைந்தபட்சம் | Minimum value

Tasks - GROUP BY | பயிற்சி

1. Count employees per department

2. Average salary by department
3. Total salary expense per department
4. Maximum and minimum salary by department
5. Count employees by job_title

HAVING Clause

```sql
-- Groups with more than 5 employees
SELECT department, COUNT(*) AS emp_count
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;

-- Departments with avg salary > 50000
SELECT department, AVG(salary) AS avg_sal
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;

-- Multiple HAVING conditions
SELECT department, COUNT(*) AS count, AVG(salary) AS avg_sal
FROM employees
GROUP BY department
HAVING COUNT(*) > 3 AND AVG(salary) > 45000;
```

Difference: WHERE filters rows before GROUP BY | HAVING filters groups after GROUP BY

Tasks - HAVING | பயிற்சி

1. Find departments with more than 10 employees
2. Find departments with average salary > 60000
3. Find job titles where max salary > 80000
4. Groups with total salary > 500000

DQL Complete Tasks - 30 Total | 30 பயிற்சிகள்

SELECT (7 tasks):

1. All columns from employees
2. Specific columns only
3. With column aliases

4. Where salary > 55000

5. Where department = 'IT'

6. Multiple conditions (AND)

7. With LIKE pattern

ORDER BY (2 tasks): 8. Sort by salary DESC 9. Sort by department ASC, salary DESC

LIMIT (2 tasks): 10. First 10 records 11. Pagination: records 11-20

DISTINCT (2 tasks): 12. Unique departments 13. Count distinct departments

GROUP BY (5 tasks): 14. Count per department 15. Average salary by department 16. Total salary by department 17. Max salary by department 18. Multiple grouping

HAVING (5 tasks): 19. Departments > 5 employees 20. Average salary > 50000 21. Total salary > 500000 22. Multiple HAVING conditions 23. Maximum salary groups

Complex (7 tasks): 24. SELECT with all clauses 25. GROUP BY with ORDER BY 26. WHERE with GROUP BY HAVING 27. SELECT DISTINCT with WHERE 28. Multiple aggregates per group 29. Nested conditions 30. Report-style query with all elements

---

4. Normalization - தயாரிப்பு மாறுபாடு

Theory | கோட்பாடு

Normalization is organizing data to minimize redundancy and improve data integrity.

1NF - First Normal Form - முதல் சாधாரண வடிவம்

Rules:

- All columns contain atomic (indivisible) values
- No repeating groups
- Each row is unique

sql

```sql
```

```
-- BAD (Not 1NF)
CREATE TABLE StudentCourses (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    courses VARCHAR(200)  -- Multiple courses in one field
);

-- GOOD (1NF)
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50)
);

CREATE TABLE StudentCourses (
    student_id INT,
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    PRIMARY KEY (student_id, course_id)
);
```

2NF - Second Normal Form - இரண்டாம் சாதாரண வடிவம்

Rules:

- Must be in 1NF
- Remove partial dependencies (non-key attributes depend on entire primary key)

```
sql
```

```
-- BAD (Not 2NF)
CREATE TABLE OrderDetails (
    order_id INT,
    product_id INT,
    product_name VARCHAR(50),  -- Depends on product_id only
    quantity INT,
    PRIMARY KEY (order_id, product_id)
);

-- GOOD (2NF)
CREATE TABLE Products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(50)
);

CREATE TABLE OrderDetails (
    order_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (order_id, product_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

3NF - Third Normal Form - மூன்றாம் சாதாரண வடிவம்

Rules:

- Must be in 2NF
- Remove transitive dependencies (non-key attributes depend on other non-key attributes)

sql

```
-- BAD (Not 3NF)
CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    department_id INT,
    department_name VARCHAR(50),  -- Depends on department_id
    manager_id INT
);

-- GOOD (3NF)
CREATE TABLE Departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50)
);

CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    department_id INT,
    manager_id INT,
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)
);
```

Tasks - Normalization | பயிற்சி

1. Create 1NF tables for School system
2. Create 2NF tables for Hospital
3. Create 3NF tables for E-commerce
4. Identify non-normalized issues
5. Normalize a bad database design
6. Create normalized Bank system
7. Normalize Student-Course relationship
8. Normalize Employee-Project system
9. Identify 2NF violations
10. Design fully normalized database

5. JOINS - அட்டவணை இணைப்புகள்

Theory | கோட்பாடு

JOINS combine rows from two or more tables based on a related column.

INNER JOIN - Inner Linking

```
sql

-- Syntax
SELECT e.emp_name, d.department_name, e.salary
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id;

-- Displays only matching records from both tables
-- Result: Only employees with valid departments
```

LEFT JOIN - Left Outer Join

```
sql

-- Syntax
SELECT e.emp_name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id;

-- All records from left table (employees)
-- Matching records from right table (departments)
-- NULL where no match
```

RIGHT JOIN - Right Outer Join

```
sql

-- Syntax
SELECT e.emp_name, d.department_name
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.department_id;

-- All records from right table (departments)
-- Matching records from left table (employees)
-- NULL where no match
```

FULL OUTER JOIN - Complete Outer Join

```
sql
```

```
-- Syntax (MySQL uses UNION)
SELECT e.emp_name, d.department_name
FROM employees e
FULL OUTER JOIN departments d ON e.department_id = d.department_id;

-- Or in MySQL:
SELECT e.emp_name, d.department_name FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id
UNION
SELECT e.emp_name, d.department_name FROM employees e
RIGHT JOIN departments d ON e.department_id = d.department_id;
```

CROSS JOIN - Cartesian Product

```
sql

-- Syntax
SELECT e.emp_name, p.project_name
FROM employees e
CROSS JOIN projects p;

-- All combinations: 5 employees × 10 projects = 50 rows
```

SELF JOIN - Same Table Join

```
sql

-- Manager-Employee relationship
SELECT e1.emp_name AS Employee, e2.emp_name AS Manager
FROM employees e1
LEFT JOIN employees e2 ON e1.manager_id = e2.emp_id;
```

Multiple JOINs

```
sql

SELECT
    e.emp_name,
    d.department_name,
    p.project_name,
    s.salary_amount
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
INNER JOIN projects p ON e.project_id = p.project_id
INNER JOIN salaries s ON e.emp_id = s.emp_id;
```

Tasks - JOINS | பயிற்சி

1. INNER JOIN employees and departments
2. LEFT JOIN to find unassigned employees
3. Find employees without departments (RIGHT JOIN with NULL)
4. FULL OUTER JOIN employees and projects
5. CROSS JOIN departments and locations
6. SELF JOIN for manager relationships
7. Multiple JOINs (3+ tables)
8. Join with aggregate functions
9. Join with WHERE and ORDER BY
10. Complex join conditions

---

6. SUB QUERIES - உள் வினாக்கள்

Theory | கோட்பாடு

A sub-query (inner query) executes first, returns value/table, then outer query uses result.

Single Row Sub Query - ஒற்றை வரிசை விளைவு

```sql
-- Find employee earning more than average salary
SELECT * FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);

-- Find employee in same department as specific employee
SELECT * FROM employees
WHERE department_id = (SELECT department_id FROM employees WHERE emp_id = 5);
```

Multiple Row Sub Query - பல வரிசை விளைவு

```sql
-- Find all employees in IT or HR
SELECT * FROM employees
WHERE department_id IN (SELECT department_id FROM departments WHERE dept_name IN ('IT', 'HR'));

-- Find employees earning more than any manager
SELECT * FROM employees
WHERE salary > (SELECT salary FROM employees WHERE job_title = 'Manager');
```

Single Column Sub Query - ஒற்றை நிரல் விளைவு

sql

```
SELECT emp_name FROM employees
WHERE department_id IN (SELECT department_id FROM departments WHERE location = 'Chennai');
```

Multiple Column Sub Query - பல நிரல் விளைவு

sql

```
-- Find employees with same department and salary range as emp_id=5
SELECT * FROM employees e1
WHERE (e1.department_id, e1.salary) IN
(SELECT department_id, salary FROM employees WHERE emp_id = 5);
```

Table Sub Query (Derived Table) - அட்டவணை விளைவு

sql

```
-- Create temporary result set
SELECT dept_name, emp_count FROM
(SELECT d.dept_name, COUNT(*) AS emp_count FROM departments d
 JOIN employees e ON d.department_id = e.department_id
 GROUP BY d.department_id) AS dept_summary
WHERE emp_count > 5;
```

Correlated Sub Query - সম্পর্কিত উপ-প্রশ্ন

sql

```
-- Find employees earning more than their department average
SELECT * FROM employees e1
WHERE salary > (SELECT AVG(salary) FROM employees e2
               WHERE e2.department_id = e1.department_id);
```

Scalar Sub Query - স্কেলার উপ-প্রশ্ন

sql

```
-- Add sub-query result to SELECT list
SELECT emp_name, salary,
       (SELECT AVG(salary) FROM employees) AS overall_avg,
       salary - (SELECT AVG(salary) FROM employees) AS diff
FROM employees;
```

EXISTS / NOT EXISTS

```sql
-- Find employees with assigned projects
SELECT * FROM employees e
WHERE EXISTS (SELECT 1 FROM projects p WHERE p.emp_id = e.emp_id);

-- Find employees without projects
SELECT * FROM employees e
WHERE NOT EXISTS (SELECT 1 FROM projects p WHERE p.emp_id = e.emp_id);
```

Tasks - SUB QUERIES | பயிற்சி

1. Single row: Find salary above average
2. Multiple row: Find departments with >5 employees
3. Single column: Get emp names from IT dept
4. Multiple column: Match department and salary
5. Derived table: Create temporary result set
6. Correlated: Compare with department average
7. Scalar: Add sub-query to SELECT list
8. EXISTS: Find employees with projects
9. NOT EXISTS: Find departments without employees
10. Nested sub-query: Sub-query within sub-query

---

7. String Functions - சரம் செயல்பாடுகள்

Common Functions | பொதுவான செயல்பாடுகள்

```sql
```

```
-- UPPER / LOWER - Change case
SELECT UPPER(emp_name) FROM employees;
SELECT LOWER(emp_name) FROM employees;

-- LENGTH / LEN - String length
SELECT emp_name, LENGTH(emp_name) FROM employees;

-- SUBSTR / SUBSTRING - Extract part
SELECT SUBSTR(emp_name, 1, 3) FROM employees;  -- First 3 chars
SELECT SUBSTRING(emp_name, 1, 3) FROM employees;

-- TRIM / LTRIM / RTRIM - Remove spaces
SELECT TRIM(emp_name) FROM employees;
SELECT LTRIM(emp_name) FROM employees;  -- Left trim
SELECT RTRIM(emp_name) FROM employees;  -- Right trim

-- REPLACE - Replace characters
SELECT REPLACE(emp_name, 'Kumar', 'K') FROM employees;

-- CONCAT / CONCATENATE - Join strings
SELECT CONCAT(emp_name, ' - ', department) FROM employees;
SELECT emp_name || ' - ' || department FROM employees;

-- INSTR / POSITION - Find position
SELECT INSTR(emp_name, 'a') FROM employees;

-- REVERSE - Reverse string
SELECT REVERSE(emp_name) FROM employees;

-- REPEAT - Repeat string
SELECT REPEAT(emp_name, 2) FROM employees;
```

8. Number Functions - எண் செயல்பாடுகள்

sql

```
-- ABS - Absolute value
SELECT ABS(salary - 50000) FROM employees;

-- ROUND - Round number
SELECT ROUND(salary, -3) FROM employees;  -- Round to nearest thousand
SELECT ROUND(45678.546, 2) FROM employees;  -- Round to 2 decimals

-- CEIL / FLOOR - Round up/down
SELECT CEIL(45.3) FROM employees;  -- Returns 46
SELECT FLOOR(45.9) FROM employees;  -- Returns 45

-- POWER - Exponentiation
SELECT POWER(2, 3) FROM employees;  -- 2^3 = 8

-- SQRT - Square root
SELECT SQRT(16) FROM employees;  -- 4

-- MOD - Modulus (remainder)
SELECT MOD(10, 3) FROM employees;  -- 1

-- RAND - Random number
SELECT RAND() FROM employees;
SELECT FLOOR(RAND() * 100) FROM employees;  -- 0-99

-- SIGN - Return sign
SELECT SIGN(-50), SIGN(0), SIGN(50);  -- -1, 0, 1

-- TRUNCATE - Remove decimals
SELECT TRUNCATE(45.789, 2) FROM employees;  -- 45.78
```

9. Date Functions - தேதி செயல்பாடுகள்

sql

```
-- CURRENT_DATE / NOW / CURDATE - Today's date
SELECT CURRENT_DATE() FROM employees;
SELECT NOW() FROM employees;

-- CURRENT_TIME - Current time
SELECT CURRENT_TIME() FROM employees;

-- DATE_ADD / DATE_SUB - Add/subtract days
SELECT DATE_ADD(hire_date, INTERVAL 1 DAY) FROM employees;
SELECT DATE_SUB(hire_date, INTERVAL 30 DAY) FROM employees;
SELECT DATE_ADD(hire_date, INTERVAL 1 YEAR) FROM employees;

-- DATEDIFF - Days between dates
SELECT DATEDIFF(CURRENT_DATE(), hire_date) AS days_worked FROM employees;

-- DAY / MONTH / YEAR - Extract parts
SELECT DAY(hire_date), MONTH(hire_date), YEAR(hire_date) FROM employees;

-- DATE_FORMAT - Format date
SELECT DATE_FORMAT(hire_date, '%d-%m-%Y') FROM employees;  -- 25-12-2023
SELECT DATE_FORMAT(hire_date, '%Y-%m-%d') FROM employees;  -- 2023-12-25

-- DAYNAME / MONTHNAME - Names
SELECT DAYNAME(hire_date) FROM employees;  -- Monday, Tuesday...
SELECT MONTHNAME(hire_date) FROM employees;  -- January, February...

-- LAST_DAY - Last day of month
SELECT LAST_DAY(hire_date) FROM employees;

-- STR_TO_DATE - String to date
SELECT STR_TO_DATE('25-12-2023', '%d-%m-%Y') FROM employees;
```

10. TCL & DCL - Transaction & Permission

TCL - Transaction Control Language - பரிவர்த்தன கட்டுப்பாடு

sql

```
-- BEGIN - Start transaction
BEGIN;
-- OR START TRANSACTION;

-- INSERT statement
INSERT INTO employees VALUES (100, 'New Employee', 50000, 'IT');

-- UPDATE statement
UPDATE employees SET salary = 55000 WHERE emp_id = 100;

-- COMMIT - Save changes
COMMIT;

-- Example with ROLLBACK - Cancel changes
BEGIN;
DELETE FROM employees WHERE emp_id = 100;
ROLLBACK;  -- Undo delete

-- SAVEPOINT - Partial rollback
BEGIN;
INSERT INTO employees VALUES (101, 'Emp1', 50000, 'IT');
SAVEPOINT sp1;
INSERT INTO employees VALUES (102, 'Emp2', 55000, 'HR');
ROLLBACK TO sp1;  -- Only undo second insert
COMMIT;
```

DCL - Data Control Language - தரவு கட்டுப்பாடு

sql

```
-- GRANT - Give permissions
GRANT SELECT ON company_db.* TO 'user1'@'localhost';
GRANT SELECT, INSERT ON company_db.employees TO 'user1'@'localhost';
GRANT ALL PRIVILEGES ON company_db.* TO 'admin'@'localhost';

-- REVOKE - Remove permissions
REVOKE SELECT ON company_db.employees FROM 'user1'@'localhost';
REVOKE ALL PRIVILEGES ON company_db.* FROM 'user1'@'localhost';

-- Create user
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password123';

-- Drop user
DROP USER 'newuser'@'localhost';
```

---

11. TRIGGERS - தூண்டிகள்

Theory | கோட்பாடு

Trigger is an automatic action executed in response to specific database events.

Single Statement Trigger - ஒற்றை அறிக்கை

sql

```
-- AFTER INSERT trigger
CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_log VALUES (NEW.emp_id, 'Employee Added', NOW());
END;

-- Before UPDATE trigger
CREATE TRIGGER before_employee_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    IF NEW.salary < OLD.salary THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Salary cannot decrease!';
    END IF;
END;
```

Multiple Statement Trigger - பல அறிக்கை

sql

```
-- Complex trigger with multiple statements
CREATE TRIGGER salary_audit_trigger
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO salary_history (emp_id, old_salary, new_salary, change_date)
    VALUES (OLD.emp_id, OLD.salary, NEW.salary, NOW());

    IF NEW.salary > OLD.salary THEN
        UPDATE employees_stats SET salary_increases = salary_increases + 1
        WHERE emp_id = NEW.emp_id;
    END IF;

    INSERT INTO audit_log VALUES (NEW.emp_id, 'Salary Updated', NOW());
END;
```

INSERT Trigger

```
sql

-- Automatically create related record
CREATE TRIGGER create_employee_profile
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_profile (emp_id, created_date)
  VALUES (NEW.emp_id, NOW());
END;
```

UPDATE Trigger

```
sql

-- Track changes
CREATE TRIGGER track_department_change
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
  IF NEW.department_id != OLD.department_id THEN
    INSERT INTO change_log
    VALUES (NEW.emp_id, 'Department Changed', OLD.department_id, NEW.department_id, NOW());
  END IF;
END;
```

DELETE Trigger

```
sql

-- Archive before delete
CREATE TRIGGER archive_employee
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO archived_employees
  SELECT * FROM employees WHERE emp_id = OLD.emp_id;
END;
```

Multi-Table Trigger - பல அட்டவணை

```
sql
```

```
-- Update multiple tables
CREATE TRIGGER update_department_stats
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    UPDATE department_stats SET employee_count = employee_count + 1
    WHERE department_id = NEW.department_id;

    UPDATE company_totals SET total_salary = total_salary + NEW.salary;
END;
```

Tasks - TRIGGERS | பயிற்சி

1. Create INSERT trigger for new employees
2. Create UPDATE trigger to prevent salary decrease
3. Create DELETE trigger to archive employees
4. Create audit trigger for all changes
5. Multi-table trigger for department updates
6. Trigger with IF conditions
7. Trigger with SIGNAL (error handling)
8. Trigger to auto-fill columns
9. Trigger to maintain summary tables
10. Trigger with date tracking

---

12. Complete Database Design Example | முழு வடிவமைப்பு

sql

```
-- Database
CREATE DATABASE company_db;
USE company_db;

-- 1. Departments Table
CREATE TABLE departments (
    dept_id INT PRIMARY KEY AUTO_INCREMENT,
    dept_name VARCHAR(50) NOT NULL UNIQUE,
    location VARCHAR(50),
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. Employees Table
CREATE TABLE employees (
    emp_id INT PRIMARY KEY AUTO_INCREMENT,
    emp_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(10),
    hire_date DATE,
    salary DECIMAL(10,2),
    manager_id INT,
    dept_id INT NOT NULL,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id),
    FOREIGN KEY (manager_id) REFERENCES employees(emp_id),
    CHECK (salary > 0)
);

-- 3. Projects Table
CREATE TABLE projects (
    project_id INT PRIMARY KEY AUTO_INCREMENT,
    project_name VARCHAR(50) NOT NULL,
    start_date DATE,
    end_date DATE,
    budget DECIMAL(10,2)
);

-- 4. Employee-Project Assignment (Many-to-Many)
CREATE TABLE emp_projects (
    emp_id INT,
    project_id INT,
    role VARCHAR
```