MySQL — Week 1: Detailed Notes & Examples

Table of Contents

1. Overview of MySQL Workbench

What is MySQL Workbench?
MySQL Workbench is a GUI tool for designing, developing, and administering MySQL servers. It has features such as connection management, SQL editor, visual schema design (EER), data export/import, and query result visualization.

2. DDL — CREATE, ALTER, DROP (Tables)

What is DDL?
Data Definition Language — statements to create or change table structures and schema objects.

CREATE TABLE (example)

```
CREATE TABLE departments (
  dept_id INT AUTO_INCREMENT PRIMARY KEY,
  dept_name VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE employees (
  emp_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  dept_id INT,
  salary DECIMAL(10,2) DEFAULT 0,
  hired_on DATE,
  CONSTRAINT fk_dept FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
);
```

ALTER TABLE (examples)

```
-- Add a new column
ALTER TABLE employees ADD COLUMN email VARCHAR(255);

-- Modify column datatype
ALTER TABLE employees MODIFY COLUMN salary DECIMAL(12,2);

-- Rename column
ALTER TABLE employees RENAME COLUMN name TO full_name;

-- Add a constraint
ALTER TABLE employees ADD CONSTRAINT chk_salary CHECK (salary >= 0);

-- Drop a column
ALTER TABLE employees DROP COLUMN email;
```

DROP TABLE

```
DROP TABLE IF EXISTS employees;
DROP TABLE IF EXISTS departments;
```

---

3. DML — INSERT, UPDATE, DELETE (Rows)

What is DML?
Data Manipulation Language — statements that change data inside tables.

INSERT examples

```
-- Insert a single row
INSERT INTO departments (dept_name) VALUES ('Engineering');

INSERT INTO employees (name, dept_id, salary, hired_on)
VALUES ('Alice', 1, 60000, '2023-08-01');

-- Multi-row insert
INSERT INTO departments (dept_name) VALUES
('Human Resources'),
('Sales'),
('Marketing');

-- Insert from SELECT
INSERT INTO employees (name, dept_id, salary)
SELECT full_name, d.dept_id, 40000
FROM other_table t
JOIN departments d ON t.dept_name = d.dept_name;
```

UPDATE examples

```
-- Give a raise to all Engineering employees
UPDATE employees
SET salary = salary * 1.10
WHERE dept_id = (SELECT dept_id FROM departments WHERE
dept_name='Engineering');

-- Update by id
UPDATE employees SET salary = 70000 WHERE emp_id = 2;
```

DELETE examples

```
-- Delete a single row
DELETE FROM employees WHERE emp_id = 5;

-- Delete all rows (but keep table)
TRUNCATE TABLE employees;
```

---

4. DQL — SELECT, WHERE, ORDER BY, GROUP BY, HAVING

What is DQL?

Data Query Language — primarily `SELECT` used to read and filter data.

Basic SELECT

```
-- Select all columns
SELECT * FROM employees;

-- Select certain columns with alias
SELECT emp_id, full_name AS name, salary FROM employees;
```

Filtering with WHERE

```
SELECT * FROM employees WHERE salary > 50000;

-- Multiple conditions
SELECT * FROM employees WHERE dept_id = 1 AND salary BETWEEN 40000 AND 80000;

-- Pattern match
SELECT * FROM employees WHERE full_name LIKE 'A%';

-- NULL checks
SELECT * FROM employees WHERE dept_id IS NULL;
```

ORDER BY and LIMIT

```
-- Order by salary descending and limit
SELECT full_name, salary FROM employees ORDER BY salary DESC LIMIT 10;

-- Use OFFSET
SELECT full_name, salary FROM employees ORDER BY salary DESC LIMIT 5 OFFSET 5;
```

GROUP BY and HAVING (group-level filters)

```
-- Count employees per department
SELECT d.dept_name, COUNT(e.emp_id) AS emp_count
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_name;

-- Departments with average salary > 50k
SELECT d.dept_name, AVG(e.salary) AS avg_salary
FROM departments d
JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_name
HAVING AVG(e.salary) > 50000;
```

5. JOINs — INNER, LEFT, RIGHT (with examples)

Join concept: Joins combine rows from two (or more) tables using related columns.

Sample tables (recap)

- `departments(dept_id, dept_name)`
- `employees(emp_id, full_name, dept_id, salary)`

INNER JOIN (only matching rows)

```
SELECT e.emp_id, e.full_name, d.dept_name
FROM employees e
INNER JOIN departments d ON e.dept_id = d.dept_id;
```

Result: Only employees who have a matching department.

LEFT JOIN (all from left, matched from right)

```
SELECT e.emp_id, e.full_name, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id;
```

Result: All employees. If an employee has no department (`dept_id` NULL or pointing to nonexistent), `dept_name` is NULL.

RIGHT JOIN (all from right, matched from left)

```
SELECT e.emp_id, e.full_name, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.dept_id;
```

Result: All departments shown; employees included where match exists.

Other join types

- `CROSS JOIN`: Cartesian product (use seldomly).

- `SELF JOIN`: join a table to itself (e.g., manager -> employee relationship).

Example: find departments with no employees

```
SELECT d.dept_name
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
WHERE e.emp_id IS NULL;
```

---

6. Aggregate Functions — SUM, COUNT, AVG, MIN, MAX

What are aggregates?
Functions that summarize multiple rows into a single value.

Examples

```
-- Total salary paid
SELECT SUM(salary) AS total_payroll FROM employees;

-- Count employees
SELECT COUNT(*) AS total_employees FROM employees;
SELECT COUNT(dept_id) AS employees_with_dept FROM employees; (counts non-NULL)

-- Average salary
SELECT AVG(salary) FROM employees;

-- Min and Max
SELECT MIN(salary) AS min_salary, MAX(salary) AS max_salary FROM employees;

-- Combined per department
SELECT d.dept_name,
       COUNT(e.emp_id) AS emp_count,
       SUM(e.salary) AS total_salary,
       AVG(e.salary) AS avg_salary
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_name;
```
