LAB 5 QUERIES

**Start MySQL Command Line**

MySQL -u root -p

* **Guidance**: Start the MySQL shell with the root user. You'll be prompted to enter the root password.

**Create a Database**

CREATE DATABASE shop\_db;

* **Guidance**: Creates a new database named shop\_db.

**Show Available Databases**

show databases;

* **Guidance**: Lists all databases available in the MySQL instance.

**Select the Database**

use shop\_db;

* **Guidance**: Selects the shop\_db database for subsequent operations.

**Create a Table**

CREATE TABLE Data(name VARCHAR(50), age INT);

* **Guidance**: Creates a table named Data with two columns:
  + name: A string up to 50 characters.
  + age: An integer.

**Describe the Table**

DESCRIBE data;

* **Guidance**: Displays the structure of the Data table, showing column names, data types, and additional details.

**Rename the Table**

ALTER TABLE data RENAME TO information;

* **Guidance**: Changes the table name from Data to information.

**Change Column Name**

ALTER TABLE data CHANGE age AGE INT;

* **Guidance**: Changes the column age to AGE while keeping the same data type (INT).

**Add a New Column**

ALTER TABLE data ADD address VARCHAR;

* **Guidance**: Adds a new column named address with a data type of VARCHAR to the Data table.

**Drop a Column**

ALTER TABLE data DROP name;

* **Guidance**: Removes the name column from the Data table.

**Drop a Table**

DROP TABLE products;

* **Guidance**: Deletes the products table completely from the database.

**Drop the Database**

DROP DATABASE shop\_db;

* **Guidance**: Deletes the entire shop\_db database and all its tables permanently.  
  *Caution*: This action cannot be undone.

**LAB 6 QUERIES**

CREATE TABLE jobs (job\_id VARCHAR(50), job\_title VARCHAR(50) , min\_salary INT ,max\_salary INT);

1. **INSERTION :**

* **Inserting a Single Record:**

INSERT INTO jobs VALUES ('SE\_PROG', 'Senior Programmer', 1000, 2000);

* **Inserting Multiple Records:**

INSERT INTO jobs VALUES ('Web\_PROG', 'Web Programmer', 1000, 2000), ('AND\_PROG', 'Android Programmer', 2000, 3000);

* **Inserting a Single Record with Specific Columns:**

INSERT INTO jobs (job\_id, job\_title, min\_salary) VALUES ('IOS\_PROG', 'IOS Developer', 2000);

1. **UPDATE:**

Modifying Data using UPDATE Statement:

* **Example 1: Update max\_salary of all jobs to 25000:**UPDATE jobs SET max\_salary = 25000;
* **Example 2: Update min\_salary to 5000 and max\_salary to 10000 for job\_id = 'IT\_PROG':**

UPDATE jobs SET min\_salary = 5000, max\_salary = 10000 WHERE job\_id = 'IT\_PROG';

1. **DELETE:**

Deleting Rows using DELETE FROM Statement:

* **Example 1: Delete all rows from the jobs table:**

DELETE FROM jobs;

* **Example 2: Delete a specific row with job\_id = 'IT\_PROG':**

DELETE FROM jobs WHERE job\_id = 'IT\_PROG';

1. **READ:**

* **Example 1: Select all rows and specific columns (job\_id, job\_title):**

SELECT job\_id, job\_title FROM jobs;

* **Example 2: Select all columns using \*:**

SELECT \* FROM jobs;

**LAB 8 QUERIES**

Below are the SQL queries for all the described cases in the lab document:

**Basic SELECT Queries**

1. **Select all columns and rows from the departments table:**

SELECT department\_id, department\_name, manager\_id, location\_id FROM departments;

1. **Select employees whose salary is greater than 10,000:**

SELECT \* FROM employees WHERE salary > 10000;

**Using Comparison Operators**

1. **Select employees whose salary is less than 10,000**

SELECT \* FROM employees WHERE salary < 10000;

1. **Select departments with location\_id equal to 1700:**

SELECT \* FROM departments WHERE location\_id = 1700;

1. **Select departments whose manager\_id is not 200:**

SELECT \* FROM departments WHERE manager\_id != 200;

1. **Select countries with region\_id less than or equal to 2:**

SELECT \* FROM countries WHERE region\_id <= 2;

**Using Pattern Matching (LIKE)**

1. **Select employees whose first\_name starts with 'a':**

SELECT \* FROM employees WHERE first\_name LIKE 'a%';

1. **Select employees whose first\_name ends with 'a':**

SELECT \* FROM employees WHERE first\_name LIKE '%a';

**Using NULL Values**

1. **Select departments where manager\_id is NULL:**

SELECT \* FROM departments WHERE manager\_id IS NULL;

**Using Logical Operators (AND, OR)**

1. **Select employees with salary > 10,000 and first\_name starting with 'N':**

SELECT \* FROM employees WHERE salary > 10000 AND first\_name LIKE 'N%';

1. **Select employees with salary > 5,000 or first\_name ending with 'a':**

SELECT \* FROM employees WHERE salary > 5000 OR first\_name LIKE '%a';

**Using BETWEEN**

1. **Select employees with salary between 500 and 20,000:**

SELECT \* FROM employees WHERE salary BETWEEN 500 AND 20000;

**Advanced SELECT Queries**

1. **Sort countries by region\_id in ascending order:**

SELECT \* FROM countries ORDER BY region\_id;

1. **Limit query to show the first 10 employees:**

SELECT \* FROM employees LIMIT 10;

1. **Retrieve first\_name and last\_name of the 3rd and 4th records from employees table:**

SELECT first\_name, last\_name FROM employees LIMIT 2, 2;

1. **Retrieve unique salaries from the employees table:**

SELECT DISTINCT salary AS Salaries FROM employees;

**LAB 9 QUERIES**

**1. Date/Time Functions**

* **Purpose**: To retrieve the current date and time.

SELECT NOW();

SELECT CURDATE(),

SELECT CURTIME();

**2. String Functions**

* **2.1. Adding Text to an Existing Value**
  + **Purpose**: To concatenate first name and job ID of employees.

SELECT CONCAT(first\_name, '-', job\_id) AS 'Employee Detail' FROM employees;

* **2.2. Changing Part of a String**
  + **Purpose**: To replace the name "Steven" with "Steven hock" in the employees table.

SELECT REPLACE(employees.first\_name, 'Steven', 'Steven hock') FROM employees WHERE employee\_id=100;

* **2.3. Finding a Piece of Text in a String**
  + **Purpose**: To locate the string "King" in the last name attribute of employees.

SELECT LOCATE('King', employees.last\_name) FROM employees;

**3. Numeric Functions**

* **Purpose**: To perform basic arithmetic operations.

SELECT 6/2, 6\*8, 6+3, 6-5;

**4. Aggregate/Summarizing Functions**

* **4.1. COUNT()**
  + **Purpose**: To count all rows in the employees table.

SELECT COUNT(\*) FROM employees;

* **4.2. AVG()**
  + **Purpose**: To show the average salary paid by department number 90.

SELECT AVG(salary) FROM employees WHERE department\_id=90;

* **4.3. MIN()**
  + **Purpose**: To show the minimum salary paid by department number 90.

SELECT MIN(salary) FROM employees WHERE department\_id=90;

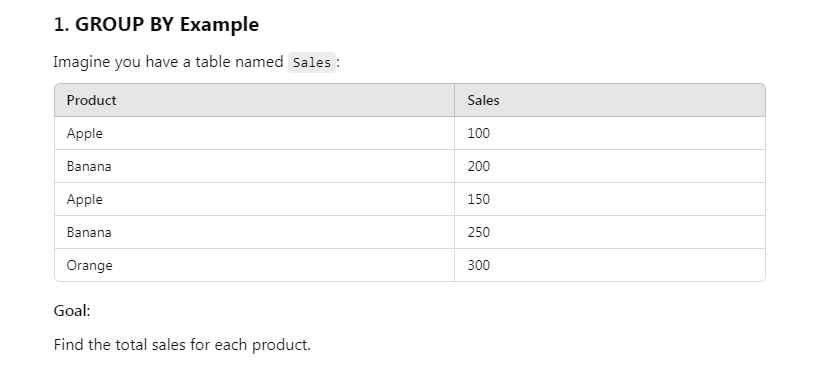
* **4.4. MAX()**
  + **Purpose**: To show the maximum salary paid by department number 90.

SELECT MAX(salary) FROM employees WHERE department\_id=90;

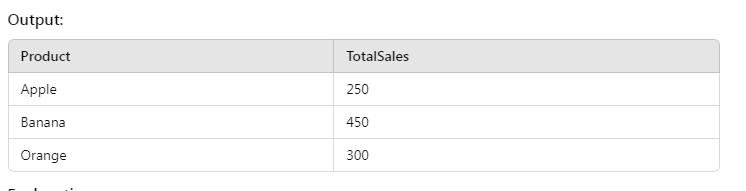
* **4.5. SUM()**
  + **Purpose**: To show the total salary paid by department number 90.

SELECT SUM(salary) FROM employees WHERE department\_id=90;

**LAB 10 QUERIES**



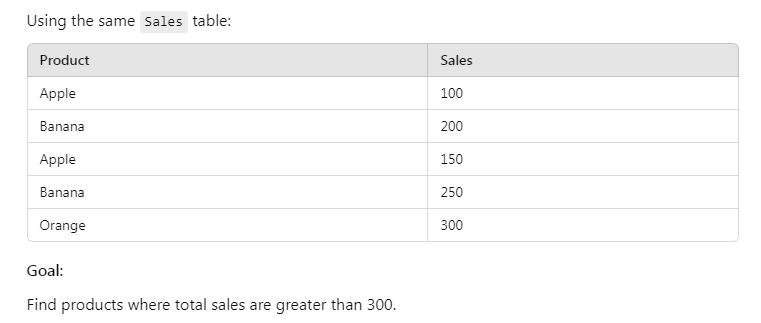
**SELECT Product, SUM(Sales) AS TotalSales FROM Sales GROUP BY Product;**



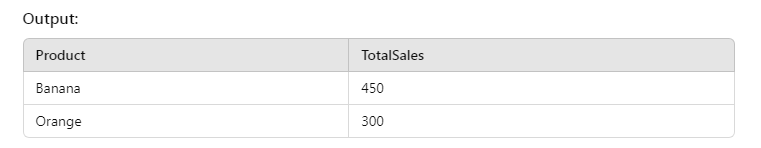
**The GROUP BY Product groups all rows with the same product name.**

**SUM(Sales) adds the sales for each group (e.g., Apple: 100 + 150 = 250).**

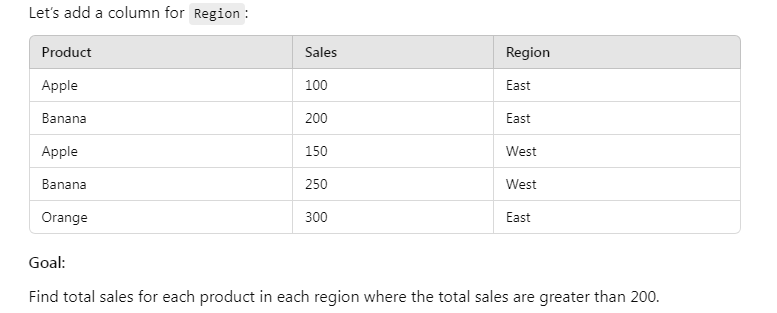
2. **HAVING Example:**



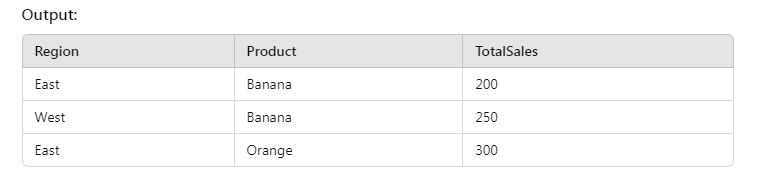
**SELECT Product, SUM(Sales) AS TotalSales FROM Sales GROUP BY Product HAVING SUM(Sales) > 300;**



3. **GROUP BY + HAVING Example:**



**SELECT Region, Product, SUM(Sales) AS TotalSales FROM Sales GROUP BY Region, Product HAVING SUM(Sales) > 200;**



**1. GROUP BY Clause**

* **Purpose**: To show the total number of employees in each department.

SELECT department\_id, COUNT(employee\_id)

FROM employees

GROUP BY department\_id;

**2. HAVING Clause**

* **Purpose**: To show the department IDs that contain more than 10 employees.

SELECT department\_id, COUNT(employee\_id)

FROM employees

GROUP BY department\_id

HAVING COUNT(employee\_id) > 10;

**3. Grouping Using a Single Column**

* **Purpose**: To get unique values for genders from the members table.

SELECT gender

FROM members

GROUP BY gender;

**4. Grouping by Multiple Columns**

* **Purpose**: To get unique combinations of category IDs and release years from the movies table.

SELECT category\_id, year\_released

FROM movies

GROUP BY category\_id, year\_released;

**5. Counting Gender Entries**

* **Purpose**: To count the total number of males and females in the members table.

SELECT gender, COUNT(membership\_number)

FROM members

GROUP BY gender;

**6. Filtering with HAVING Clause**

* **Purpose**: To show all release years for movies with category ID 8.

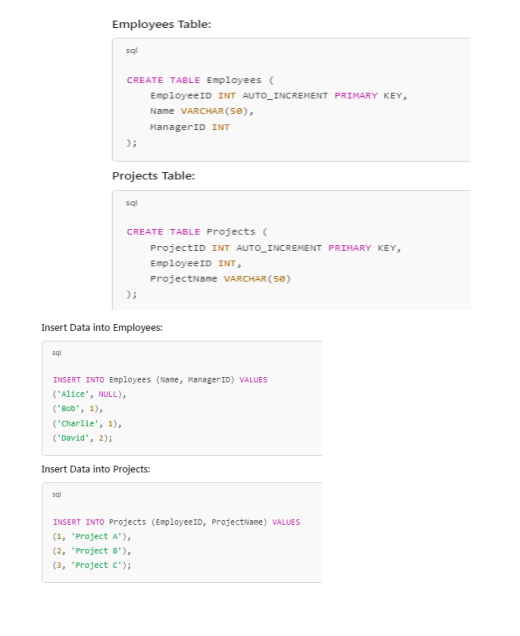
SELECT \*

FROM movies

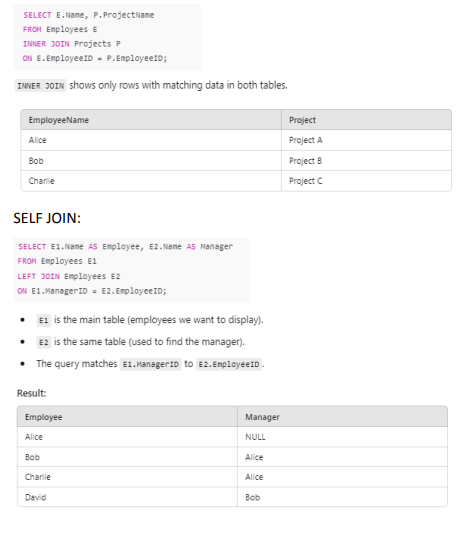
GROUP BY category\_id, year\_released

HAVING category\_id = 8;

**LAB 11 QUERIES**







**1. Basic JOIN Query**

* **Purpose**: To list the employee names from the **employees** table and their corresponding job titles from the **jobs** table by joining the two tables on the common **job\_id**.

SELECT employees.first\_name, jobs.job\_title

FROM employees

JOIN jobs ON employees.job\_id = jobs.job\_id;

**2. JOIN Using WHERE Clause (Not Recommended)**

* **Purpose**: To achieve the same result as the previous query but using the WHERE clause for joining, which is not recommended.

SELECT employees.first\_name, jobs.job\_title

FROM employees, jobs

WHERE employees.job\_id = jobs.job\_id;

**3. Using Aliases for Column Names**

* **Purpose**: To create aliases for the **job\_id** columns from both tables to avoid confusion in the result set.

SELECT employees.first\_name, jobs.job\_title,

jobs.job\_id AS 'jobs job\_id',

employees.job\_id AS 'employees job\_id'

FROM employees

JOIN jobs ON employees.job\_id = jobs.job\_id;

**4. Using Aliases for Table Names**

* **Purpose**: To use aliases for table names to simplify the query and improve readability.

SELECT e.first\_name, j.job\_title,

j.job\_id AS 'jobs job\_id',

e.job\_id AS 'employees job\_id'

FROM employees AS e

JOIN jobs AS j ON e.job\_id = j.job\_id;

**5. Creating Tables for Example**

* **Purpose**: To create two tables (**t1** and **t2**) with a common column for demonstration of joins.

CREATE TABLE t1 (

id INT PRIMARY KEY,

pattern VARCHAR(50) NOT NULL

);

CREATE TABLE t2 (

id VARCHAR(50) PRIMARY KEY,

pattern VARCHAR(50) NOT NULL

);

**6. Inserting Data into Tables**

* **Purpose**: To insert sample data into both **t1** and **t2** tables for join demonstration.

INSERT INTO t1 (id, pattern)

VALUES (1, 'Divot'),

(2, 'Brick'),

(3, 'Grid');

INSERT INTO t2 (id, pattern)

VALUES ('A', 'Brick'),

('B', 'Grid'),

('C', 'Diamond');

**7. INNER JOIN Example**

* **Purpose**: To demonstrate an INNER JOIN between **t1** and **t2**, producing a Cartesian product of the two tables.

SELECT t1.id, t1.pattern, t2.id, t2.pattern

FROM t1

INNER JOIN t2 ON t1.pattern = t2.pattern;

**LAB 12 QUERIES**

**1. LEFT JOIN Example**

* **Purpose**: To join the **employees** table with the **jobs** table, returning all rows from the **employees** table and matching rows from the **jobs** table. If there is no match, NULLs will appear in the columns of the **jobs** table.

SELECT \*

FROM employees

LEFT JOIN jobs ON employees.job\_id = jobs.job\_id;

**2. LEFT JOIN with Job History**

* **Purpose**: To join the **employees** table with the **job\_history** table, returning all rows from the **employees** table and matching rows from the **job\_history** table. If there is no match, NULLs will appear in the columns of the **job\_history** table.

SELECT employees.employee\_id, end\_date

FROM employees

LEFT JOIN job\_history ON employees.job\_id = job\_history.job\_id;

**3. Error Example with LEFT JOIN**

* **Purpose**: To illustrate that a WHERE clause cannot be used directly on an OUTER JOIN. This query will produce a syntax error.

SELECT \*

FROM t1

LEFT JOIN t2

WHERE t1.id = t2.id;

**4. RIGHT JOIN Example**

* **Purpose**: To join the **t1** table with the **t2** table, returning all rows from the **t2** table and matching rows from the **t1** table. If there is no match, NULLs will appear in the columns of the **t1** table.

SELECT \*

FROM t1

RIGHT JOIN t2 ON t1.id = t2.id;

**5. RIGHT JOIN Using USING Clause**

* **Purpose**: To join the **t1** table with the **t2** table using the **USING** clause, which simplifies the syntax when the column names are the same in both tables.

SELECT \*

FROM t1

RIGHT JOIN t2 USING (id);

**6. RIGHT JOIN with Locations and Countries**

* **Purpose**: To join the **locations** table with the **countries** table, returning all rows from the **countries** table and matching rows from the **locations** table. If there is no match, NULLs will appear in the columns of the **locations** table.

SELECT country\_name, location\_id

FROM locations

RIGHT JOIN countries ON locations.country\_id = countries.country\_id;

Table 1:

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | Ali |
| 2 | Ahmed |

Table2:

|  |  |
| --- | --- |
| ID | Age |
| 1 | 20 |
| 2 | 25 |
| 3 | 30 |

**SELECT Table1.ID, Table1.Name, Table2.Age**

**FROM Table1**

**RIGHT JOIN Table2 ON Table1.ID = Table2.ID;**

**After right join:**

|  |  |  |
| --- | --- | --- |
| ID | NAME | AGE |
| 1 | Ali | 20 |
| 2 | Ahmed | 25 |
| 3 | **Null** | 30 |

Table 1:

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | Ali |
| 2 | Ahmed |
| 3 | Hassan |

Table 2:

|  |  |
| --- | --- |
| **ID** | **Age** |
| 1 | 20 |
| 2 | 25 |

**SELECT Table1.ID, Table1.Name, Table2.Age**

**FROM Table1**

**LEFT JOIN Table2 ON Table1.ID = Table2.ID;**

**After Left join:**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Age** |
| 1 | Ali | 20 |
| 2 | Ahmed | 25 |
| 3 | Hassan | **null** |

**LAB 13 QUERIES**

**1. Basic Subquery Example**

* **Purpose**: To find all employees who earn a salary greater than Donald's salary.

SELECT \*

FROM employees

WHERE SALARY > (SELECT SALARY FROM employees WHERE first\_name = 'Donald');

**2. Single-row Subquery Example**

* **Purpose**: To find the names of employees that are paid more than 'Asad Khan'.

SELECT employee\_id, first\_name, salary, department\_id

FROM employees

WHERE SALARY IN (SELECT MIN(salary) FROM employees GROUP BY department\_id);

**3. Multiple-row Subquery Example**

* **Purpose**: To find employees who earn the same salary as the minimum salary for each department.

SELECT \*

FROM employees

WHERE SALARY > (SELECT SALARY FROM employees WHERE employee\_name = 'Shelley');

**4. Using ANY Operator**

* **Purpose**: To find employees whose salary is less than any salary of employees with the job ID 'SA\_REP' and whose job is not 'SA\_REP'.

SELECT employee\_id, first\_name, salary, job\_id

FROM employees

WHERE SALARY < ANY (SELECT salary FROM employees WHERE job\_id = 'SA\_REP')

AND job\_id <> 'SA\_REP';

**5. Multiple-column Subquery Example**

* **Purpose**: To display employee details for those whose salary is greater than the average salary in their department.

SELECT a.employee\_id, a.first\_name, a.salary, a.department\_id, b.salavg

FROM employees AS a,

(SELECT department\_id, AVG(salary) AS 'salavg' FROM employees GROUP BY department\_id) AS b

WHERE a.department\_id = b.department\_id AND a.salary > b.salavg;

**6. Correlated Subquery Example**

* **Purpose**: To find all employees in each department who earn more than the average salary of their respective department.

SELECT employee\_id, first\_name, salary

FROM employees AS e

WHERE salary > (SELECT AVG(salary) FROM employees WHERE department\_id = e.department\_id);