INDEX

S.No.	Date	Topic	Page No	Sign
1		**Week 1: (Store all your data in a file named `db.sql` and email it to yourself for future reference and use in subsequent labs.)** • Create a Scenario based ER-Models with the entities. (Hospital Details like: Wards, Patients, Doctor, Bills etc) • Convert this ER-model into table with all the entities. (Minimum five Entities). • Insert random data in each column of all the tables. • Update the table by applying some conditions.(For example: using alter command) Apply the `DELETE` and `DROP` command, and then review the results.		
2		**Week 2:** • Create a user and provide the GRANT privileges to the user on the database then REVOKE the given privileges. • Insert any five records in the previous schema and apply the rollback. Also check the results. • Add default, check, unique and not null constraints to the schema. Insert NULL values and check the results.Add duplicate value and try to make a column as primary key, check what happen to the table.		
3		**Week 3: (If the Employee table is not present in the `db.sql` file, please create it.)** • Create an Employee table with the following attributes and constraints: Employee Table - (Employee Id. (Primary key), Name, Department, Age (check >18), Salary, City). • Display the total number of employees. • Retrieve all information of employees whose age is 22. • Fetch the employee id, name, and department, whose salary >= 50000. • Print the name of the employees and label the column as "Full Name" for those employees whose department name is 'Finance' and age is 22. • Print the department names from the employee table without having the duplicates.		
4		**Week 4:** • Find out the maximum and minimum salary from the employee table. • Show the total salary and average salary of all the employees. • Show all the details of the employees who have the same salary. Display the employees name from lowest salary to the highest salary.Display the employee name and salary (department-wise) for employees, whose salary is greater than or equal to 10,000 and age is greater than 25.		

5		**Week 5:** • Fetch the information of employees who belong to the city "Delhi" or "Pune." • Print the name and department of employees whose ID is in the range from 2001 to 2005. • Show the names of employees who belong to the same city (use the IN operator). • Check whether the all employee is belongs to the same city or not. (use ALL operator) • Check whether the all employee is belongs to the same city or not. (use ANY operator)Check whether the all employee is belongs to the same city or not. (use Exists operator)		
6		**Week 6:** • Show the record of employees who working in the 'CSE' department. • Fetch the names of employees whose names start with the letters 'ay'. • Fetch the information of employees, including their names and departments, whose names end with the letters 'sh'. • Display the employee names and their departments of employees, whose city name starts with 'D' or ends with 'h'. • Print all records of employees whose salary is greater than 15,000 and whose name starts with 'h'. Print the names of employees whose names consist of exactly three letters.Print the names of employees along with their city for those whose names have at least five letters.		
7		**Week 7:** • Create two tables named as employee and department with the given constraints and attributes: Employee table - (Employee Id.(Primary key), Department ID, Name, Age (check >18), Salary, City) Department table - (Department Id, and Department name) • Display the details of employees along with their corresponding department names. • Print the names of employees who are not assigned to any department. • Print the employee names and department names for employees whose salary is greater than 25,000. (Using left join). • Display the names of employees along with their department names for those who are not assigned to any department. • Print the employee names and their corresponding department names for employees with a salary greater than 25,000. (Using right join). Display the names of departments along with the names of employees who are older than 30 years.		
8		**Week 8:** • Create the table to keep track of customer records and their order. Customer table - (Name as Not null, Customer_id as primary key, Age, Address)		

		Order table - (Customer_id, order_id, date). • Apply the full join and the full outer join to the schema and review the results. • Display the name of the city as "destination" for customers who have placed orders. Apply the cross join and check the results.Display the customer names and order IDs for customers who have placed orders from the same city.		
9		**Week 9:** • Create the Student table, Register table and Program table. Student table - (Roll no. as primary key, Name as not null, city) Program table - (Program ID as primary key, Program Name as not null, Program Fee not less than 10000, Department) Register table - (Program ID and Roll no. as primary composite key) • Display the details of students who are registered in the "MCA" program. • Display the list of all students, who are registered in at least one program. • Display the details of programs that have fees greater than the average fee. • Display the names of students who are registered in a program having fees less than 30000. • Display the details of students who have not registered in any course. Display the names of programs in which a maximum number of students are registered.Display the names of programs in which a minimum number of students are registered.		
10		**Week 10:** Find out the second minimum salary of an employee. • Find out the second minimum salary of an employee without using limit, dense range, and order by clause. • Find out the third maximum salary of an employee. • Find out the third maximum salary of an employee without using limit, dense range, and order by clause. • Display the names and salaries of employees who earn more than the average salary of their department. Fetch the list of the employee who belongs to the same department but earns less than the second employee. Display the names of employees who are older than their colleagues in the same department		
11		**Week 11:** • Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the EMPLOYEE table. This trigger will display the salary difference between the old values and new values. • Add a new employee with the salary value inserted and check the result. • Try to update the existing employee salary and see what happens. • Delete a record of employees and check what happens. • Convert employee name into uppercase whenever an employee record is inserted or updated.		

12		**Week 12:** Case study 1: (General Hospital) A hospital relies on a database to manage its operations effectively. This database helps keep track of various aspects, including different wards like the General Ward, Emergency Ward, and Specific Ward. Each ward contains patients who are admitted based on their General Practitioner's(GP) recommendation and the approval of a consultant from the hospital. When a patient is admitted, the hospital records essential personal details such as their name, age, gender, address, and contact information. This information is crucial for medical and administrative purposes. Additionally,the hospital maintains a separate register to record all medical tests and treatments for each patient, ensuring that their medical history is thoroughly documented. Patients may undergo multiple tests during their stay, and the database is designed to link each patient with these test records. Each patient is assigned a leading consultant who oversees their treatment, but they may also be examined by other doctors if needed.The database also tracks the connections between patients, consultants, and doctors. Consultants and doctors might specialize in different medical fields and can treat patients from various wards, adding flexibility to the care provided. Overall, this database ensures that patient information, medical records, and hospital operations are managed efficiently. It supports the hospital in delivering high-quality care, streamlining administrative tasks, and addressing the specialized needs of patients and medical staff. Based on the details provided in the case study, address the following requirements: Create an ER diagram based on the hospital's database system case study. Include entities like patients, wards, consultants, and doctors with relevant attributes such as Patient ID, Ward ID, and Consultant ID. Also, none of the entities in the template are marked as weak; if you wish to change that, you may. You will need to specify two things: a. Specify all attributes and keys for each entity. Clearly define relationships, such as patients being associated with wards, consultants, and doctors, and include connections between patients and their medical tests. b. Define all relationships and constraints, including primary keys, cardinality, and participation constraints. Show how a patient can undergo multiple tests and be treated by various doctors. Note: Model most constraints from the description. If some constraints can't be represented, provide comments explaining the limitations.		
13		**Week 13:** Case Study 2: (Tracking the Employee Record) An organization has implemented a detailed database system to manage and track its employees and departmental activities. The organization is divided into various departments, each with a unique identification number and name. Each department is managed by a designated manager, who is responsible for overseeing the operations within that department. Additionally, some departments may be located in Employees are assigned to specific departments, and the system tracks the date on which a manager was appointed to each department, ensuring that managerial changes are recorded accurately. Beyond departmental assignments,. Moreover, every project undertaken by the organization is managed by a specific department, although employees from various departments may be assigned to these projects based on their expertise and the project's requirements. For each project, the database includes details such as the project name, project number, and location. This comprehensive approach ensures that all aspects of employee and project management are wellcoordinated and effectively managed. Based on the details provided in the case study, address the following requirements: Create an ER diagram representing the organization's database system. Include key entities such as Departments, Employees, and Projects. For each Department, capture attributes like Department ID, Department Name, Manager ID, and Location. For Employees, include attributes such as Employee ID, Name, Birth Date, Address, Gender, Salary, and Department ID. Projects should have attributes like Project ID, Project Name, and Project Location. a. Specify all attributes and primary keys for each entity. Clearly define relationships between entities, such as employees being assigned to departments, departments managing projects, and supervisory relationships among employees. b. Define the relationships and constraints, including primary keys, cardinality, and participation constraints. For example, a department can manage multiple projects, and employees can work on multiple projects while reporting to one or more supervisors. Note: Include comments to address any constraints from the case study that cannot be fully represented in the ER diagram. Ensure the diagram accurately reflects the management of employee information, departmental structures, and project assignments as described. Based on the provided case study, perform the following queries: a. Write an SQL query to identify the department(s) with the highest average salary among its employees.		

		b. Write an SQL query to list all employees who are directly supervised by more than one manager. c. Write an SQL query to find the project(s) with the highest total hours spent by employees. d. Write an SQL query to find all employees who have never been assigned to any project. e. Write an SQL query to list each department along with the total number of projects managed by the department and the total number of employees assigned to it.		
14		**Week 14:** Case Study 2: (Trainee Record in a Institution) A training institute requires a sophisticated database to effectively track the progress of trainees in their various training programs. Trainees enroll in different programs, such as Java Developer, Full Stack Developer, and Data Scientist. Each of these programs consists of several courses offered by the institute, each with its own unique code, title, and number of credit hours. Courses are overseen by an instructor, who may also be involved in teaching the course. In addition to the main instructor, each course has one or more teaching assistants who help facilitate the course. An instructor can manage and teach multiple courses across different programs, reflecting their expertise and versatility. The database captures detailed information about each course, including its unique code, title, credit hours, the instructor responsible for managing and teaching the course, the teaching assistants assigned, and the department to which the course belongs. Each course may have prerequisites, which means that some courses must be completed before others can be taken. For trainees, the database records essential details such as their ID numbers, names, addresses, the training program they are enrolled in, and their academic performance. This includes information on the courses they have taken and the grades they received. Based on the details provided in the case study, address the following requirements: Develop an ER diagram to model the training institute's database system, incorporating all key entities and their attributes. The primary entities to include are Trainees, Programs, Courses, Instructors, and Teaching Assistants. For each entity, specify attributes and primary keys: Trainees should have Trainee_ID, Name, Address, Program_ID, and Performance details. Courses should include Course_Code, Title, Credit_Hours, Prerequisites, and links to Instructors and Teaching Assistants. a. Define the relationships between entities, such as Trainees being enrolled in Programs, Courses assigned to Programs, and Instructors and Teaching Assistants managing and teaching Courses. Include relationships for course prerequisites and mandatory courses within Programs. b. Clearly specify key constraints like primary keys for each entity and cardinality and participation constraints for relationships. For instance, a Course may have multiple Teaching Assistants and prerequisites, while a Trainee can be enrolled in multiple Courses. Note: Add comments to highlight any constraints from the case study that are challenging to represent in the ER diagram. Ensure the diagram effectively captures the management of trainees' progress, course requirements, and instructor roles as outlined. Based on the provided case study, perform the following queries: a. Write an SQL query to show the names and IDs of trainees who have completed all the required courses for their program. b. Write an SQL query to get the names of instructors and the courses they teach, including any courses that have prerequisites. c. Write an SQL query to find the average grade for each course and list the courses where the average grade is below than 70%. d. Write an SQL query to find the trainees who have signed up for the most courses, no matter which program they are in. e. Write an SQL query to list all courses and the total hours all trainees have spent on each course.		

3. Insert random data in each column of all the tables.

Solution Query:

```
mysql> INSERT INTO Hospital (Hospital_ID, Name, Location) VALUES
    -> (1, 'Sunrise Hospital', 'Bangalore'),
    -> (2, 'Green Valley Hospital', 'Mumbai');
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert data into Ward
mysql> INSERT INTO Ward (Ward_ID, Hospital_ID, Ward_Name, Capacity) VALUES
    -> (1, 1, 'General Ward', 20),
    -> (2, 1, 'ICU', 5),
    -> (3, 2, 'Pediatric Ward', 15);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert data into Patient
mysql> INSERT INTO Patient (Patient_ID, Ward_ID, Name, Age, Gender, Admission_Date) VALUES
    -> (1, 1, 'Aarav', 30, 'Male', '2024-10-01'),
    -> (2, 2, 'Diya', 25, 'Female', '2024-10-03'),
    -> (3, 3, 'Kiran', 12, 'Male', '2024-10-04');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert data into Doctor
mysql> INSERT INTO Doctor (Doctor_ID, Name, Specialty, Phone_Number) VALUES
    -> (1, 'Dr. Anjali', 'Cardiologist', '9876543210'),
    -> (2, 'Dr. Ramesh', 'Pediatrician', '9876543211');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert data into Bill
mysql> INSERT INTO Bill (Bill_ID, Patient_ID, Total_Amount, Payment_Status) VALUES
    -> (1, 1, 5000.00, 'Paid'),
    -> (2, 2, 15000.00, 'Pending'),
    -> (3, 3, 8000.00, 'Paid');
Query OK, 3 rows affected (0.01 sec)
```

4. Update the table by applying some conditions (for example: using ALTER command).

Solution Query:

```
mysql> ALTER TABLE Patient ADD Contact_Number VARCHAR(15);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> UPDATE Patient
    -> SET Contact_Number = '9998887776'
    -> WHERE Patient_ID = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

5. Apply the DELETE and DROP commands, and then review the results.

Solution Query:

```
mysql> DELETE FROM Bill WHERE Patient_ID = 2;
Query OK, 1 row affected (0.01 sec)


mysql> DROP TABLE Doctor_Patient;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from doctor_patient;
ERROR 1146 (42S02): Table 'gb.doctor_patient' doesn't exist
mysql>
```

WEEK 2

1. Create a user and provide the GRANT privileges to the user on the database, then REVOKE the given privileges.
Solution Query:

```
mysql> CREATE USER 'hospital_user'@'localhost' IDENTIFIED BY 'password123';
Query OK, 0 rows affected (0.04 sec)

mysql> GRANT ALL PRIVILEGES ON hospital_db.* TO 'hospital_user'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> REVOKE ALL PRIVILEGES ON hospital_db.* FROM 'hospital_user'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

2. Insert any five records in the previous schema and apply the ROLLBACK. Check the results.
Solution Query:

```
mysql>
mysql> INSERT INTO Patient (Patient_ID, Ward_ID, Name, Age, Gender, Admission_Date, Contact_Number) VALUES
    -> (4, 1, 'Vikas', 45, 'Male', '2024-10-06', '9876543212'),
    -> (5, 3, 'Meera', 28, 'Female', '2024-10-06', '9876543213'),
    -> (6, 2, 'Raj', 32, 'Male', '2024-10-06', '9876543214'),
    -> (7, 1, 'Pooja', 23, 'Female', '2024-10-06', '9876543215'),
    -> (8, 3, 'Kabir', 36, 'Male', '2024-10-06', '9876543216');
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)
```

3. Add DEFAULT, CHECK, UNIQUE, and NOT NULL constraints to the schema.
Solution Query:

```
mysql> ALTER TABLE Patient
    -> MODIFY Age INT NOT NULL,                          -- Not Null constraint
    -> ADD CONSTRAINT chk_age CHECK (Age > 0),           -- Check constraint for positive age
    -> ADD CONSTRAINT unique_contact UNIQUE (Contact_Number),  -- Unique constraint on Contact Number
    -> MODIFY Gender VARCHAR(10) DEFAULT 'Unknown';      -- Default constraint for Gender
Query OK, 3 rows affected (0.17 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from patient;
+------------+---------+-------+-----+--------+----------------+----------------+
| Patient_ID | Ward_ID | Name  | Age | Gender | Admission_Date | Contact_Number |
+------------+---------+-------+-----+--------+----------------+----------------+
|          1 |       1 | Aarav |  30 | Male   | 2024-10-01     | 9998887776     |
|          2 |       2 | Diya  |  25 | Female | 2024-10-03     | NULL           |
|          3 |       3 | Kiran |  12 | Male   | 2024-10-04     | NULL           |
+------------+---------+-------+-----+--------+----------------+----------------+
3 rows in set (0.00 sec)
```

4. **Insert NULL values and check the results.**
Solution Query:

```
mysql> INSERT INTO Patient (Patient_ID, Ward_ID, Name, Age, Gender, Admission_Date, Contact_Number)
    -> VALUES (9, 1, 'Priya', NULL, NULL, '2024-10-06', NULL);
ERROR 1048 (23000): Column 'Age' cannot be null
```

5. **Add duplicate value and try to make a column as Primary Key, then check what happens to the table.**
Solution Query:

```
mysql> INSERT INTO Patient (Patient_ID, Ward_ID, Name, Age, Gender, Admission_Date, Contact_Number)
    -> VALUES (10, 2, 'Sohail', 29, 'Male', '2024-10-06', '9876543215');
Query OK, 1 row affected (0.01 sec)

mysql> ALTER TABLE Patient ADD PRIMARY KEY (Contact_Number);
ERROR 1068 (42000): Multiple primary key defined
mysql>
```

WEEK 3

1. Create an Employee table with the following attributes and constraints:
Employee Table - ***(Employee_Id (Primary Key), Name, Department, Age (CHECK >18), Salary, City)***
Solution Query:

```
+--------+------+------------+------+----------+-----------+
| Emp_ID | Name | Department | Age  | Salary   | City      |
+--------+------+------------+------+----------+-----------+
|   2001 | Amit | Finance    |   22 | 52000.00 | Delhi     |
|   2002 | Riya | HR         |   25 | 48000.00 | Pune      |
|   2003 | John | Finance    |   22 | 50000.00 | Mumbai    |
|   2004 | Sara | IT         |   30 | 70000.00 | Delhi     |
|   2005 | Raj  | CSE        |   26 | 55000.00 | Bangalore |
+--------+------+------------+------+----------+-----------+
5 rows in set (0.01 sec)
```

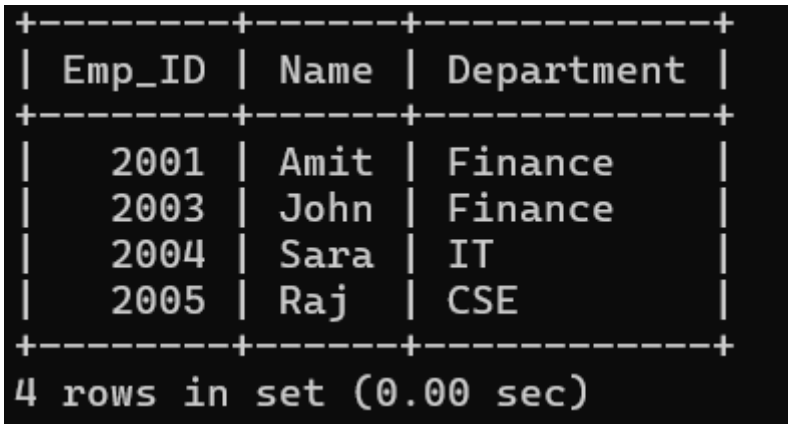
2. Display the total number of employees.
Solution Query:

```
mysql> SELECT COUNT(*) AS Total_Employees FROM Employee;
+-----------------+
| Total_Employees |
+-----------------+
|               5 |
+-----------------+
1 row in set (0.01 sec)
```

3. Retrieve all information of employees whose Age = 22.
Solution Query:

```
+--------+------+------------+------+----------+--------+
| Emp_ID | Name | Department | Age  | Salary   | City   |
+--------+------+------------+------+----------+--------+
|   2001 | Amit | Finance    |   22 | 52000.00 | Delhi  |
|   2003 | John | Finance    |   22 | 50000.00 | Mumbai |
+--------+------+------------+------+----------+--------+
2 rows in set (0.01 sec)
```

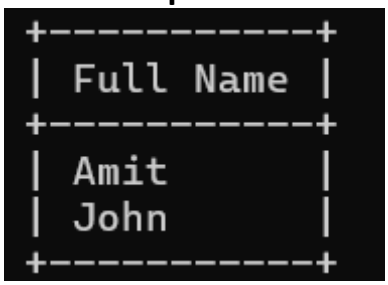
4. Fetch the Employee_Id, Name, and Department of employees whose Salary >= 50,000.
Solution Query:



```
+--------+------+------------+
| Emp_ID | Name | Department |
+--------+------+------------+
|   2001 | Amit | Finance    |
|   2003 | John | Finance    |
|   2004 | Sara | IT         |
|   2005 | Raj  | CSE        |
+--------+------+------------+
4 rows in set (0.00 sec)
```

5. Print the Name of employees and label the column as "Full Name" for those whose Department = 'Finance' and Age = 22.
Solution Query

```
SELECT Name AS "Full Name"
FROM Employee
WHERE Department = 'Finance' AND Age = 22;
```



```
+-----------+
| Full Name |
+-----------+
| Amit      |
| John      |
+-----------+
```

6. Print the Department names from the employee table without duplicates.
Solution Query:

```
SELECT DISTINCT Department FROM Employee;
```

```
+------------+
| Department |
+------------+
| Finance    |
| HR         |
| IT         |
| CSE        |
+------------+
```

WEEK 4

1. Find out the maximum and minimum salary from the employee table.
Solution Query:

```
mysql> SELECT MAX(Salary) AS Maximum_Salary, MIN(Salary) AS Minimum_Salary
    -> FROM Employee;
+----------------+----------------+
| Maximum_Salary | Minimum_Salary |
+----------------+----------------+
|       75000.00 |        9500.00 |
+----------------+----------------+
1 row in set (0.00 sec)
```

2. Show the total salary and average salary of all the employees.
Solution Query:

```
mysql> SELECT SUM(Salary) AS Total_Salary, AVG(Salary) AS Average_Salary
    -> FROM Employee;
+--------------+----------------+
| Total_Salary | Average_Salary |
+--------------+----------------+
|    194500.00 |   27785.714286 |
+--------------+----------------+
1 row in set (0.00 sec)
```

3. Show all the details of the employees who have the same salary.
Solution Query:

```
mysql> SELECT *
    -> FROM Employee
    -> WHERE Salary IN (
    ->     SELECT Salary
    ->     FROM Employee
    ->     GROUP BY Salary
    ->     HAVING COUNT(*) > 1
    -> );
Empty set (0.01 sec)
```

4. Display the employee names from lowest salary to highest salary.
Solution Query:

```
mysql> SELECT Name, Salary
    -> FROM Employee
    -> ORDER BY Salary ASC;
+--------------+----------+
| Name         | Salary   |
+--------------+----------+
| Neha Gupta   |  9500.00 |
| Vikram Singh | 10000.00 |
| Ayesha Khan  | 12000.00 |
| Suman Verma  | 15000.00 |
| Rina Patel   | 18000.00 |
| Rahul Sharma | 55000.00 |
| Hari Joshi   | 75000.00 |
+--------------+----------+
7 rows in set (0.00 sec)
```

5. Display the employee name and salary (department-wise) for employees whose salary >= 10,000 and age > 25.

Solution Query:

```
mysql> SELECT Department, Name, Salary
    -> FROM Employee
    -> WHERE Salary >= 10000
    ->   AND Age > 25
    -> ORDER BY Department;
+------------+-------------+----------+
| Department | Name        | Salary   |
+------------+-------------+----------+
| CSE        | Ayesha Khan | 12000.00 |
| CSE        | Rina Patel  | 18000.00 |
| HR         | Suman Verma | 15000.00 |
| Marketing  | Vikram Singh| 10000.00 |
+------------+-------------+----------+
4 rows in set (0.00 sec)
```

WEEK 5

1. Fetch the information of employees who belong to the city "Delhi" or "Pune".
Solution Query:

```
mysql> SELECT * FROM Employee WHERE City IN ('Delhi', 'Pune');
+--------+------+------------+------+----------+-------+
| Emp_ID | Name | Department | Age  | Salary   | City  |
+--------+------+------------+------+----------+-------+
|   2001 | Amit | Finance    |   22 | 52000.00 | Delhi |
|   2002 | Riya | HR         |   25 | 48000.00 | Pune  |
|   2004 | Sara | IT         |   30 | 70000.00 | Delhi |
+--------+------+------------+------+----------+-------+
3 rows in set (0.00 sec)
```

2. Print the Name and Department of employees whose Employee_ID is in the range from 2001 to 2005.
Solution Query:

3. Show the names of employees who belong to the same city (use the IN operator).

Solution Query:

```
mysql> SELECT Name FROM Employee WHERE City IN (
    ->     SELECT City FROM Employee GROUP BY City HAVING COUNT(*) > 1
    -> );
+------+
| Name |
+------+
| Amit |
| Sara |
+------+
2 rows in set (0.00 sec)
```

4. Check whether all employees belong to the same city or not (use the ALL operator).
Solution Query:

```
mysql> SELECT * FROM Employee WHERE City = ALL (SELECT City FROM Employee);
Empty set (0.00 sec)
```

5. Check whether all employees belong to the same city or not (use the ANY operator).
Solution Query:

```
mysql> SELECT * FROM Employee WHERE City = ANY (SELECT City FROM Employee);
+--------+------+------------+------+----------+-----------+
| Emp_ID | Name | Department | Age  | Salary   | City      |
+--------+------+------------+------+----------+-----------+
|   2001 | Amit | Finance    |   22 | 52000.00 | Delhi     |
|   2002 | Riya | HR         |   25 | 48000.00 | Pune      |
|   2003 | John | Finance    |   22 | 50000.00 | Mumbai    |
|   2004 | Sara | IT         |   30 | 70000.00 | Delhi     |
|   2005 | Raj  | CSE        |   26 | 55000.00 | Bangalore |
+--------+------+------------+------+----------+-----------+
5 rows in set (0.00 sec)
```

6. Check whether all employees belong to the same city or not (use the EXISTS operator).
Solution Query:

```
mysql>  SELECT * FROM Employee e WHERE EXISTS (
    ->     SELECT 1 FROM Employee e2 WHERE e.City = e2.City
    -> );
+--------+------+------------+------+----------+-----------+
| Emp_ID | Name | Department | Age  | Salary   | City      |
+--------+------+------------+------+----------+-----------+
|   2001 | Amit | Finance    |   22 | 52000.00 | Delhi     |
|   2002 | Riya | HR         |   25 | 48000.00 | Pune      |
|   2003 | John | Finance    |   22 | 50000.00 | Mumbai    |
|   2004 | Sara | IT         |   30 | 70000.00 | Delhi     |
|   2005 | Raj  | CSE        |   26 | 55000.00 | Bangalore |
+--------+------+------------+------+----------+-----------+
5 rows in set (0.00 sec)
```

WEEK 6

1. Show the records of employees who are working in the 'CSE' department.

Solution Query:

```
mysql> SELECT * FROM Employee WHERE Department = 'CSE';
+--------+------+------------+------+----------+-----------+
| Emp_ID | Name | Department | Age  | Salary   | City      |
+--------+------+------------+------+----------+-----------+
|   2005 | Raj  | CSE        |   26 | 55000.00 | Bangalore |
+--------+------+------------+------+----------+-----------+
1 row in set (0.00 sec)
```

2. Fetch the names of employees whose names start with the letters 'ay'.

Solution Query:

```
mysql> SELECT Name FROM Employee WHERE Name LIKE 'ay%';
Empty set (0.00 sec)
```

3. Fetch the information of employees (including their Name and Department) whose names end with the letters 'sh'.

Solution Query:

```
mysql> SELECT Name, Department FROM Employee WHERE Name LIKE '%sh';
Empty set (0.00 sec)
```

4. Display the Employee Names and Departments of employees whose City name starts with 'D' or ends with 'h'.

Solution Query:

```
mysql> SELECT Name, Department FROM Employee WHERE City LIKE 'D%' OR City LIKE '%h';
+------+------------+
| Name | Department |
+------+------------+
| Amit | Finance    |
| Sara | IT         |
+------+------------+
2 rows in set (0.00 sec)
```

5. Print all records of employees whose Salary > 15,000 and whose Name starts with 'h'.

Solution Query:

```
mysql> SELECT * FROM Employee WHERE Salary > 15000 AND Name LIKE 'h%';
Empty set (0.00 sec)
```

6. Print the names of employees whose names consist of exactly three letters.

Solution Query:

```
mysql> SELECT Name FROM Employee WHERE LENGTH(Name) = 3;
+------+
| Name |
+------+
| Raj  |
+------+
1 row in set (0.00 sec)
```

WEEK 7

1. Create two tables named Employee and Department with the given attributes and constraints: Employee Table -

- **Employee_Id (Primary Key) ,Department_Id (Foreign Key)**
- **Name Age (Check > 18)**
- **Salary**
- **City**

Department Table -

- **Department_Id (Primary Key) Department_Name**

Solution Query:

```
mysql> CREATE TABLE Department (
    ->     Department_ID INT PRIMARY KEY,
    ->     Department_Name VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> CREATE TABLE Employee (
    ->     Employee_ID INT PRIMARY KEY,
    ->     Department_ID INT,
    ->     Emp_Name VARCHAR(50),
    ->     Age INT CHECK (Age > 18),
    ->     Salary DECIMAL(10,2),
    ->     City VARCHAR(50),
    ->     FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID)
    -> );
```

2. Display the details of employees along with their corresponding department names.

Solution Query:

```
mysql> SELECT e.Employee_ID, e.Emp_Name, e.Age, e.Salary, e.City, d.Department_Name
    -> FROM Employee e
    -> JOIN Department d
    -> ON e.Department_ID = d.Department_ID;
+-------------+--------------+------+----------+-----------+-----------------------+
| Employee_ID | Emp_Name     | Age  | Salary   | City      | Department_Name       |
+-------------+--------------+------+----------+-----------+-----------------------+
|         101 | Aarav Sharma |   28 | 32000.00 | Delhi     | Computer Science      |
|         106 | Simran Kaur  |   33 | 41000.00 | Kolkata   | Computer Science      |
|         102 | Priya Mehta  |   31 | 45000.00 | Mumbai    | Information Technology |
|         105 | Arjun Singh  |   29 | 27000.00 | Delhi     | Information Technology |
|         103 | Rohan Kumar  |   26 | 23000.00 | Pune      | Electronics           |
|         104 | Sneha Nair   |   35 | 38000.00 | Chennai   | Human Resources       |
|         107 | Aditya Rao   |   41 | 52000.00 | Hyderabad | Finance               |
+-------------+--------------+------+----------+-----------+-----------------------+
7 rows in set (0.00 sec)
```

3. Print the names of employees who are not assigned to any department.

Solution Query:

```
mysql> SELECT Emp_Name
    -> FROM Employee
    -> WHERE Department_ID IS NULL;
Empty set (0.00 sec)
```

4. Print the employee names and department names for employees whose salary is greater than 25,000. (Using LEFT JOIN)

Solution Query:

```
mysql> SELECT e.Emp_Name, d.Department_Name
    -> FROM Employee e
    -> LEFT JOIN Department d
    -> ON e.Department_ID = d.Department_ID
    -> WHERE e.Salary > 25000;
+--------------+------------------------+
| Emp_Name     | Department_Name        |
+--------------+------------------------+
| Aarav Sharma | Computer Science       |
| Priya Mehta  | Information Technology |
| Sneha Nair   | Human Resources        |
| Arjun Singh  | Information Technology |
| Simran Kaur  | Computer Science       |
| Aditya Rao   | Finance                |
+--------------+------------------------+
```

5. Display the names of employees along with their department names for those who are not assigned to any department.

Solution Query:

```
mysql> SELECT e.Emp_Name, d.Department_Name
    -> FROM Employee e
    -> LEFT JOIN Department d
    -> ON e.Department_ID = d.Department_ID
    -> WHERE e.Department_ID IS NULL;
```

6. Print the employee names and their corresponding department names for employees with a salary greater than 25,000. (Using RIGHT JOIN)

Solution Query:

```
mysql> SELECT d.Department_Name, e.Emp_Name
    -> FROM Department d
    -> JOIN Employee e
    -> ON e.Department_ID = d.Department_ID
    -> WHERE e.Age > 30;
+------------------------+-------------+
| Department_Name        | Emp_Name    |
+------------------------+-------------+
| Information Technology | Priya Mehta |
| Human Resources        | Sneha Nair  |
| Computer Science       | Simran Kaur |
| Finance                | Aditya Rao  |
+------------------------+-------------+
```

7. Display the names of departments along with the names of employees who are older than 30 years.

Solution Query:

```
mysql> SELECT d.Department_Name, e.Emp_Name
    -> FROM Department d
    -> JOIN Employee e
    -> ON e.Department_ID = d.Department_ID
    -> WHERE e.Age > 30;
+------------------------+-------------+
| Department_Name        | Emp_Name    |
+------------------------+-------------+
| Information Technology | Priya Mehta |
| Human Resources        | Sneha Nair  |
| Computer Science       | Simran Kaur |
| Finance                | Aditya Rao  |
+------------------------+-------------+
```

WEEK 8

1. Create the tables to keep track of customer records and their orders.

Customer Table -

- **Name (NOT NULL)**
- **Customer_Id (Primary Key)**
- **Age**
- **Address**

Order Table -

- **Customer_Id**
- **Order_Id**
- **Date**

Solution Query:

```
mysql> CREATE TABLE Customer (
    ->         Customer_Id INT PRIMARY KEY,
    ->         Name VARCHAR(50) NOT NULL,
    ->         Age INT,
    ->         Address VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE Orders (
    ->     Order_Id INT PRIMARY KEY,
    ->     Customer_Id INT,
    ->     Date DATE,
    ->     FOREIGN KEY (Customer_Id) REFERENCES Customer(Customer_Id)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

2. Apply the FULL JOIN and the FULL OUTER JOIN to the schema and review the results.

Solution Query:

```
ysql> SELECT c.Customer_Id, c.Name, o.Order_Id, o.Date
    -> FROM Customer c
    -> LEFT JOIN Orders o ON c.Customer_Id = o.Customer_Id
    -> UNION
    -> SELECT c.Customer_Id, c.Name, o.Order_Id, o.Date
    -> FROM Customer c
    -> RIGHT JOIN Orders o ON c.Customer_Id = o.Customer_Id;
+-------------+-------+----------+------------+
| Customer_Id | Name  | Order_Id | Date       |
+-------------+-------+----------+------------+
|           1 | Ravi  |      101 | 2024-11-01 |
|           2 | Aman  |      103 | 2024-11-05 |
|           3 | Neha  |      102 | 2024-11-03 |
|           4 | Priya |     NULL | NULL       |
+-------------+-------+----------+------------+
4 rows in set (0.01 sec)
```

3. Display the name of the city as "destination" for customers who have placed orders.
Solution Query:

```
mysql> SELECT DISTINCT c.Address AS Destination
    -> FROM Customer c
    -> JOIN Orders o ON c.Customer_Id = o.Customer_Id;
+-------------+
| Destination |
+-------------+
| Delhi       |
| Mumbai      |
+-------------+
```

4. Apply the CROSS JOIN and check the results.
Solution Query:

```
mysql> SELECT c.Name, o.Order_Id, o.Date
    -> FROM Customer c
    -> CROSS JOIN Orders o;
+-------+----------+------------+
| Name  | Order_Id | Date       |
+-------+----------+------------+
| Ravi  |      103 | 2024-11-05 |
| Ravi  |      102 | 2024-11-03 |
| Ravi  |      101 | 2024-11-01 |
| Aman  |      103 | 2024-11-05 |
| Aman  |      102 | 2024-11-03 |
| Aman  |      101 | 2024-11-01 |
| Neha  |      103 | 2024-11-05 |
| Neha  |      102 | 2024-11-03 |
| Neha  |      101 | 2024-11-01 |
```

5. Display the customer names and order IDs for customers who have placed orders from the same city.
Solution Query:

```
mysql> SELECT c.Name, o.Order_Id
    -> FROM Customer c
    -> JOIN Orders o ON c.Customer_Id = o.Customer_Id
    -> WHERE c.Address IN (
    ->     SELECT Address
    ->     FROM Customer
    ->     GROUP BY Address
    ->     HAVING COUNT(Address) > 1
    -> );
+------+----------+
| Name | Order_Id |
+------+----------+
| Ravi |      101 |
| Neha |      102 |
+------+----------+
```

WEEK 9

1. Create the Student, Register, and Program tables.

Student Table -

- **Roll_No (Primary Key)**
- **Name (NOT NULL)**
- **City**

Program Table -

- **Program_ID (Primary Key)**
- **Program_Name (NOT NULL)**
- **Program_Fee (Check ≥ 10000)**
- **Department**

Register Table -

- **Program_ID**
- **Roll_No**

(Primary Composite Key: Program_ID + Roll_No)

Solution Query:

```
mysql> CREATE TABLE Student (
    ->     Roll_No INT PRIMARY KEY,
    ->     Name VARCHAR(50) NOT NULL,
    ->     City VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE Program (
    ->     Program_ID INT PRIMARY KEY,
    ->     Program_Name VARCHAR(50) NOT NULL,
    ->     Program_Fee DECIMAL(10,2) CHECK (Program_Fee >= 10000),
    ->     Department VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE Register (
    ->     Program_ID INT,
    ->     Roll_No INT,
    ->     PRIMARY KEY (Program_ID, Roll_No),
    ->     FOREIGN KEY (Program_ID) REFERENCES Program(Program_ID),
    ->     FOREIGN KEY (Roll_No) REFERENCES Student(Roll_No)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

2. Display the details of students who are registered in the "MCA" program.

Solution Query:

```
mysql> SELECT s.*
    -> FROM Student s
    -> JOIN Register r ON s.Roll_No = r.Roll_No
    -> JOIN Program p ON p.Program_ID = r.Program_ID
    -> WHERE p.Program_Name = 'MCA';
+---------+-------+-------+
| Roll_No | Name  | City  |
+---------+-------+-------+
|       1 | Aarav | Delhi |
+---------+-------+-------+
```

3. Display the list of all students who are registered in at least one program.

Solution Query:

```
mysql> SELECT DISTINCT s.*
    -> FROM Student s
    -> JOIN Register r ON s.Roll_No = r.Roll_No;
+---------+-------+---------+
| Roll_No | Name  | City    |
+---------+-------+---------+
|       1 | Aarav | Delhi   |
|       2 | Priya | Mumbai  |
|       3 | Rohan | Pune    |
|       4 | Isha  | Chennai |
|       5 | Arjun | Delhi   |
+---------+-------+---------+
5 rows in set (0.00 sec)
```

4. Display the details of programs that have fees greater than the average fee.
Solution Query:

```
mysql> SELECT *
    -> FROM Program
    -> WHERE Program_Fee > (SELECT AVG(Program_Fee) FROM Program);
+------------+--------------+-------------+------------------+
| Program_ID | Program_Name | Program_Fee | Department       |
+------------+--------------+-------------+------------------+
|        101 | MCA          |    35000.00 | Computer Science |
|        102 | MBA          |    45000.00 | Management       |
|        105 | MSC          |    30000.00 | Science          |
+------------+--------------+-------------+------------------+
3 rows in set (0.01 sec)
```

5. Display the names of students who are registered in a program having fees less than 30000.
Solution Query:

```
mysql> SELECT s.Name, p.Program_Name, p.Program_Fee
    -> FROM Student s
    -> JOIN Register r ON s.Roll_No = r.Roll_No
    -> JOIN Program p ON r.Program_ID = p.Program_ID
    -> WHERE p.Program_Fee < 30000;
+-------+--------------+-------------+
| Name  | Program_Name | Program_Fee |
+-------+--------------+-------------+
| Rohan | BCA          |    20000.00 |
| Isha  | BCA          |    20000.00 |
+-------+--------------+-------------+
2 rows in set (0.00 sec)
```

6. Display the details of students who have not registered in any course.
Solution Query:

```
mysql> SELECT s.*
    -> FROM Student s
    -> LEFT JOIN Register r ON s.Roll_No = r.Roll_No
    -> WHERE r.Program_ID IS NULL;
Empty set (0.00 sec)
```

7. Display the names of programs in which a maximum number of students are registered.

Solution Query:

```
mysql> SELECT p.Program_Name, COUNT(r.Roll_No) AS Total_Students
    -> FROM Program p
    -> JOIN Register r ON p.Program_ID = r.Program_ID
    -> GROUP BY p.Program_Name
    -> HAVING COUNT(r.Roll_No) = (
    ->     SELECT MAX(StudentCount)
    ->     FROM (
    ->         SELECT COUNT(Roll_No) AS StudentCount
    ->         FROM Register
    ->         GROUP BY Program_ID
    ->     ) AS sub
    -> );
+--------------+----------------+
| Program_Name | Total_Students |
+--------------+----------------+
| BCA          |              2 |
+--------------+----------------+
1 row in set (0.01 sec)
```

8. Display the names of programs in which a minimum number of students are registered.

Solution Query:

```
mysql> SELECT p.Program_Name, COUNT(r.Roll_No) AS Total_Students
    -> FROM Program p
    -> JOIN Register r ON p.Program_ID = r.Program_ID
    -> GROUP BY p.Program_Name
    -> HAVING COUNT(r.Roll_No) = (
    ->     SELECT MIN(StudentCount)
    ->     FROM (
    ->         SELECT COUNT(Roll_No) AS StudentCount
    ->         FROM Register
    ->         GROUP BY Program_ID
    ->     ) AS sub
    -> );
+--------------+----------------+
| Program_Name | Total_Students |
+--------------+----------------+
| MCA          |              1 |
| MBA          |              1 |
| MSC          |              1 |
+--------------+----------------+
3 rows in set (0.00 sec)
```

WEEK 10

1. Find out the second minimum salary of an employee.
Solution Query:

```
mysql> SELECT MIN(Salary) AS Second_Min_Salary
    -> FROM Employee
    -> WHERE Salary > (SELECT MIN(Salary) FROM Employee);
+-------------------+
| Second_Min_Salary |
+-------------------+
|          28000.00 |
+-------------------+
```

2. Find out the second minimum salary of an employee without using LIMIT, DENSE_RANK, or ORDER BY clause.
Solution Query:

```
mysql> SELECT MIN(e1.Salary) AS Second_Min_Salary
    -> FROM Employee e1
    -> WHERE e1.Salary > (SELECT MIN(e2.Salary) FROM Employee e2);
+-------------------+
| Second_Min_Salary |
+-------------------+
|          28000.00 |
+-------------------+
```

3. Find out the third maximum salary of an employee.
Solution Query:

```
mysql> SELECT Salary AS Third_Max_Salary
    -> FROM Employee
    -> ORDER BY Salary DESC
    -> LIMIT 1 OFFSET 2;
+-------------------+
| Third_Max_Salary  |
+-------------------+
|          37000.00 |
+-------------------+
```

4. Find out the third maximum salary of an employee without using LIMIT, DENSE_RANK, or ORDER BY clause.
Solution Query:

```
mysql> SELECT MIN(e1.Salary) AS Third_Max_Salary
    -> FROM Employee e1
    -> WHERE 2 = (
    ->     SELECT COUNT(DISTINCT e2.Salary)
    ->     FROM Employee e2
    ->     WHERE e2.Salary > e1.Salary
    -> );
+-------------------+
| Third_Max_Salary  |
+-------------------+
|          37000.00 |
+-------------------+
```

5. Display the names and salaries of employees who earn more than the average salary of their department.

Solution Query:

```
mysql> SELECT e.Emp_Name, e.Department, e.Salary
    -> FROM Employee e
    -> WHERE e.Salary > (
    ->     SELECT AVG(Salary)
    ->     FROM Employee
    ->     WHERE Department = e.Department
    -> );
+-------------+------------+----------+
| Emp_Name    | Department | Salary   |
+-------------+------------+----------+
| Sneha Nair  | IT         | 40000.00 |
| Simran Kaur | CSE        | 45000.00 |
+-------------+------------+----------+
2 rows in set (0.00 sec)
```

6. Fetch the list of employees who belong to the same department but earn less than the second employee.

Solution Query:

```
mysql> SELECT e1.Emp_Name AS Employee, e1.Department, e1.Salary
    -> FROM Employee e1
    -> WHERE e1.Salary < (
    ->     SELECT MAX(e2.Salary)
    ->     FROM Employee e2
    ->     WHERE e1.Department = e2.Department
    -> );
+-------------+------------+----------+
| Employee    | Department | Salary   |
+-------------+------------+----------+
| Aarav Sharma | CSE        | 32000.00 |
| Rohit Kumar  | CSE        | 25000.00 |
| Aditya Rao   | IT         | 29000.00 |
+-------------+------------+----------+
3 rows in set (0.00 sec)
```

7. Display the names of employees who are older than their colleagues in the same department.

Solution Query:

```
mysql> SELECT e1.Emp_Name, e1.Department, e1.Age
    -> FROM Employee e1
    -> WHERE e1.Age > (
    ->     SELECT AVG(e2.Age)
    ->     FROM Employee e2
    ->     WHERE e1.Department = e2.Department
    -> );
+------------+------------+------+
| Emp_Name   | Department | Age  |
+------------+------------+------+
| Simran Kaur | CSE        |   24 |
| Aditya Rao  | IT         |   27 |
+------------+------------+------+
2 rows in set (0.00 sec)
```

WEEK 11

1. Create a row-level trigger for the Employee table that fires for INSERT, UPDATE, or DELETE operations.
This trigger should display the salary difference between the old and new values.

Solution Query:

```
mysql> CREATE TABLE Employee (
    ->     Emp_ID INT PRIMARY KEY,  Emp_Name VARCHAR(50),
    ->     Department VARCHAR(50),  Salary DECIMAL(10,2),    Age INT);
Query OK, 0 rows affected (0.03 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER Salary_Change_Trigger
    -> BEFORE UPDATE ON Employee
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE diff DECIMAL(10,2);
    ->     SET diff = NEW.Salary - OLD.Salary;
    ->     INSERT INTO Salary_Log (Emp_ID, Emp_Name, Old_Salary, New_Salary, Difference)
    ->     VALUES (OLD.Emp_ID, OLD.Emp_Name, OLD.Salary, NEW.Salary, diff);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

2. Add a new employee with a salary value inserted and check the result.
Solution Query:

```
mysql> INSERT INTO Employee VALUES (6, 'Simran Kaur', 'EEE', 37000, 24);
Query OK, 1 row affected (0.01 sec)
```

3. Try to update the existing employee salary and observe what happens.

Solution Query:

```
mysql> UPDATE Employee
    -> SET Salary = 48000
    -> WHERE Emp_ID = 2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> SELECT * FROM Salary_Log;
+--------+-------------+------------+------------+------------+---------------------+
| Emp_ID | Emp_Name    | Old_Salary | New_Salary | Difference | Change_Time         |
+--------+-------------+------------+------------+------------+---------------------+
|      2 | Priya Mehta |   42000.00 |   48000.00 |    6000.00 | 2025-11-12 20:12:42 |
+--------+-------------+------------+------------+------------+---------------------+
1 row in set (0.00 sec)
```

4. Delete a record of an employee and check what happens.

Solution Query:

```
mysql> CREATE TRIGGER Employee_Delete_Trigger
    -> BEFORE DELETE ON Employee
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO Salary_Log (Emp_ID, Emp_Name, Old_Salary, New_Salary, Difference)
    ->     VALUES (OLD.Emp_ID, OLD.Emp_Name, OLD.Salary, NULL, NULL);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
mysql> DELETE FROM Employee WHERE Emp_ID = 3;
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM Salary_Log;
+--------+------------+------------+------------+------------+---------------------+
| Emp_ID | Emp_Name   | Old_Salary | New_Salary | Difference | Change_Time         |
+--------+------------+------------+------------+------------+---------------------+
|      2 | Priya Mehta |   42000.00 |   48000.00 |    6000.00 | 2025-11-12 20:12:42 |
|      3 | Rohit Kumar |   31000.00 |       NULL |       NULL | 2025-11-12 20:13:34 |
+--------+------------+------------+------------+------------+---------------------+
2 rows in set (0.00 sec)
```

5. Convert the employee name into uppercase whenever an employee record is inserted or updated

Solution Query:

```
mysql> CREATE TRIGGER Uppercase_Name_Insert
    -> BEFORE INSERT ON Employee
    -> FOR EACH ROW
    -> BEGIN
    ->     SET NEW.Emp_Name = UPPER(NEW.Emp_Name);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
mysql> DELIMITER //
mysql> CREATE TRIGGER Uppercase_Name_Update
    -> BEFORE UPDATE ON Employee
    -> FOR EACH ROW
    -> BEGIN
    ->     SET NEW.Emp_Name = UPPER(NEW.Emp_Name);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
mysql> INSERT INTO Employee VALUES (7, 'Isha Varma', 'CSE', 40000, 23);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM Employee;
+--------+--------------+------------+----------+------+
| Emp_ID | Emp_Name     | Department | Salary   | Age  |
+--------+--------------+------------+----------+------+
|      1 | Aarav Sharma | CSE        | 35000.00 |   23 |
|      2 | Priya Mehta  | IT         | 48000.00 |   25 |
|      4 | Sneha Nair   | CSE        | 39000.00 |   26 |
|      5 | Aditya Rao   | IT         | 45000.00 |   28 |
|      6 | Simran Kaur  | EEE        | 37000.00 |   24 |
|      7 | ISHA VARMA   | CSE        | 40000.00 |   23 |
+--------+--------------+------------+----------+------+
6 rows in set (0.00 sec)
```

WEEK 12

Case study 1: (General Hospital)

A hospital relies on a database to manage its operations effectively. This database helps keep track of various aspects, including different wards like the General Ward, Emergency Ward, and Specific Ward. Each ward contains patients who are admitted based on their General Practitioner's(GP) recommendation and the approval of a consultant from the hospital. When a patient is admitted, the hospital records essential personal details such as their name, age, gender, address, and contact information. This information is crucial for medical and administrative purposes. Additionally,the hospital maintains a separate register to record all medical tests and treatments for each patient, ensuring that their medical history is thoroughly documented. Patients may undergo multiple tests during their stay, and the database is designed to link each patient with these test records. Each patient is assigned a leading consultant who oversees their treatment, but they may also be examined by other doctors if needed.The database also tracks the connections between patients, consultants, and doctors. Consultants and doctors might specialize in different medical fields and can treat patients from various wards, adding flexibility to the care provided. Overall, this database ensures that patient information, medical records, and hospital operations are managed efficiently. It supports the hospital in delivering high-quality care, treamlining administrative tasks, and addressing the specialized needs of patients and medical staff. Based on the details provided in the case study, address the following requirements:

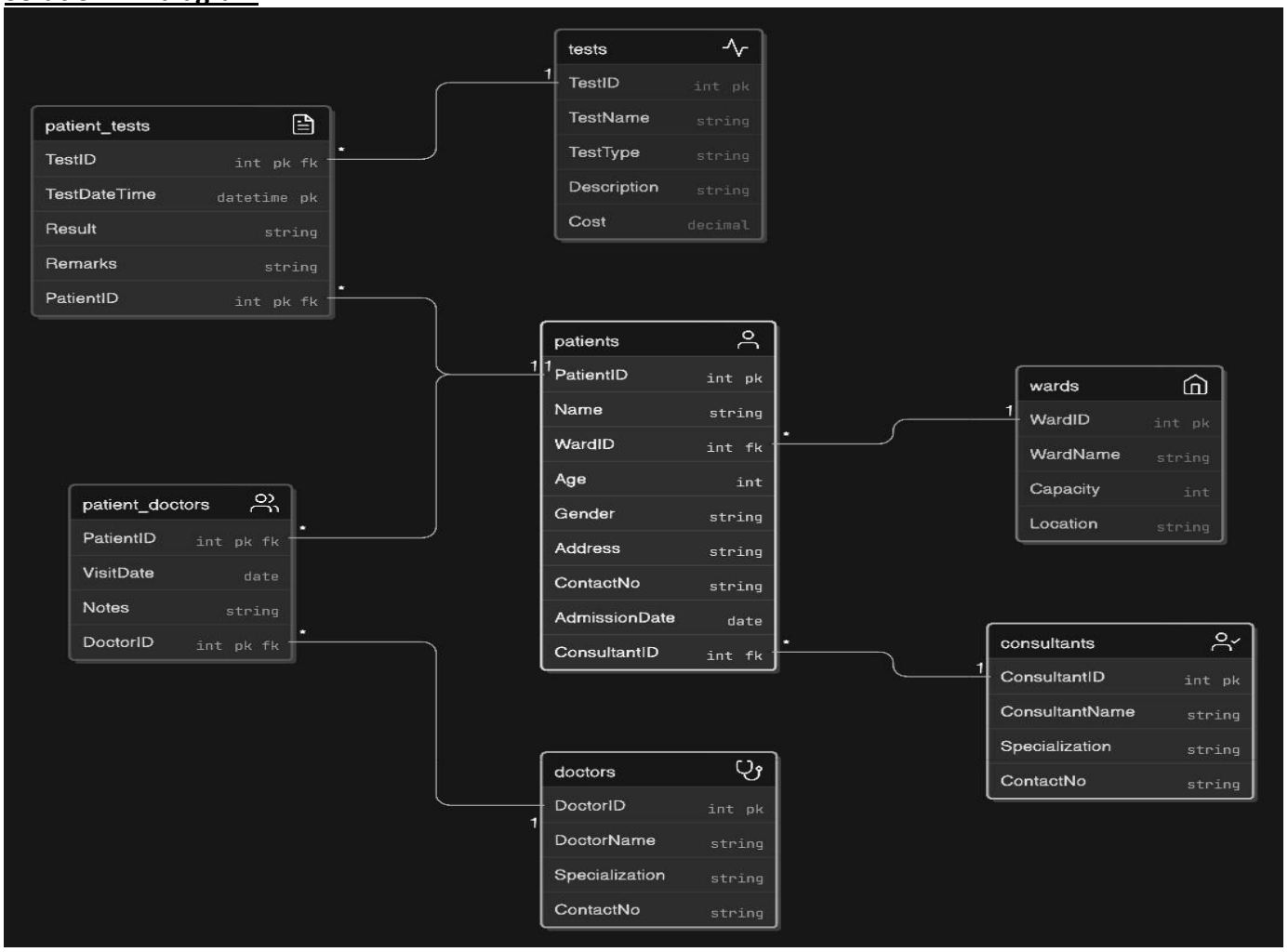
Create an ER diagram based on the hospital's database system case study. Include entities like patients, wards, consultants, and doctors with relevant attributes such as Patient ID, Ward ID, and Consultant ID. Also, none of the entities in the template are marked as weak; if you wish to change that, you may. You will need to specify two things:

a. Specify all attributes and keys for each entity. Clearly define relationships, such as patients being associated with wards, consultants, and doctors, and include connections between patients and their medical tests.

b. Define all relationships and constraints, including primary keys, cardinality, and participation constraints. Show how a patient can undergo multiple tests and be treated by various doctors.

Note: Model most constraints from the description. If some constraints can't be represented, provide comments explaining the limitations.

Solution: ER diagram



WEEK 13

Case Study 2: (Tracking the Employee Record)

An organization has implemented a detailed database system to manage and track its employees and departmental activities. The organization is divided into various departments, each with a unique identification number and name. Each department is managed by a designated manager, who is responsible for overseeing the operations within that department. Additionally, some departments may be located in different geographic locations, reflecting the organization's diverse operational reach. The database maintains comprehensive records for each employee, including their name, identification number, birth date, address, gender, and salary.

Employees are assigned to specific departments, and the system tracks the date on which a manager was appointed to each department, ensuring that managerial changes are recorded accurately. Beyond departmental assignments, the database captures supervisory relationships where employees may be directly supervised by others. This helps in understanding the hierarchical structure within the organization. Moreover, every project undertaken by the organization is managed by a specific department, although employees from various departments may be assigned to these projects based on their expertise and the project's requirements.

For each project, the database includes details such as the project name, project number, and location. Additionally, it records the hours spent by employees on each project, providing insights into individual contributions and project progress. This tracking helps in managing project resources effectively and ensures that employees' time and efforts are properly accounted for. Overall, this database system facilitates efficient management of employee information, departmental structures, and project assignments. It allows the organization to monitor departmental performance, manage employee roles and responsibilities, and track the progress of various projects with precision. This comprehensive approach ensures that all aspects of employee and project management are wel lcoordinated and effectively managed.

Based on the details provided in the case study, address the following requirements:

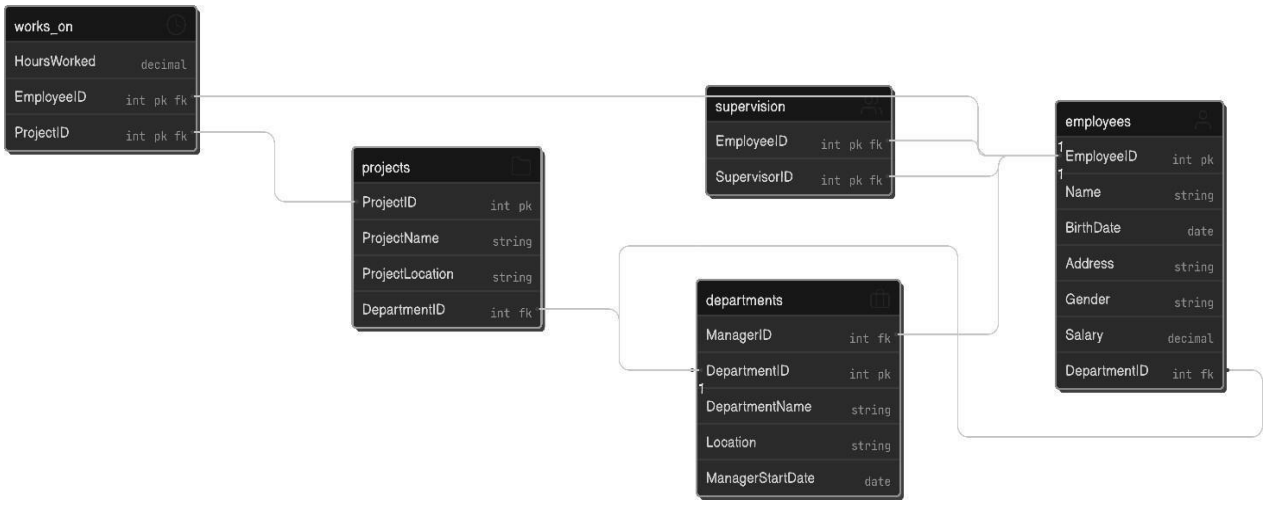
Create an ER diagram representing the organization's database system. Include key entities such as Departments, Employees, and Projects. For each Department, capture attributes like Department ID, Department Name, Manager ID, and Location. For Employees, include attributes such as Employee ID, Name, Birth Date, Address, Gender, Salary, and Department ID. Projects should have attributes like Project ID, Project Name, and Project Location.

a. Specify all attributes and primary keys for each entity. Clearly define relationships between entities, such as employees being assigned to departments, departments managing projects, and supervisory relationships among employees.

b. Define the relationships and constraints, including primary keys, cardinality, and participation constraints. For example, a department can manage multiple projects, and employees can work on multiple projects while reporting to one or more supervisors. Note: Include comments to address any constraints from the case study that cannot be fully represented in the ER diagram. Ensure the diagram accurately reflects the management of employee information, departmental structures, and project assignments as described.

Based on the provided case study, perform the following queries:

Solution: ER Diagram



a) Write an SQL query to identify the department(s) with the highest average salary among its employees.

```
mysql> SELECT DepartmentID
    -> FROM Employee
    -> GROUP BY DepartmentID
    -> HAVING AVG(Salary) = (
    ->     SELECT MAX(AvgSalary)
    ->     FROM (
    ->         SELECT AVG(Salary) AS AvgSalary
    ->         FROM Employee
    ->         GROUP BY DepartmentID
    ->     ) AS DeptAvg
    -> );
+--------------+
| DepartmentID |
+--------------+
|           30 |
+--------------+
1 row in set (0.00 sec)
```

Write an SQL query to list all employees who are directly supervised by more than one manager.

Solution Query :

```
mysql> SELECT EmployeeID
    -> FROM Supervision
    -> GROUP BY EmployeeID
    -> HAVING COUNT(SupervisorID) > 1;
+------------+
| EmployeeID |
+------------+
|        202 |
+------------+
```

b) Write an SQL query to find the project(s) with the highest total hours spent by employees.

Solution Query :

```
mysql> SELECT ProjectID
    -> FROM Works_On
    -> GROUP BY ProjectID
    -> HAVING SUM(HoursWorked) = (
    ->     SELECT MAX(TotalHours)
    ->     FROM (
    ->         SELECT SUM(HoursWorked) AS TotalHours
    ->         FROM Works_On
    ->         GROUP BY ProjectID
    ->     ) AS P
    -> );
+-----------+
| ProjectID |
+-----------+
|      2001 |
+-----------+
1 row in set (0.00 sec)
```

c) Write an SQL query to find all employees who have never been assigned to any project.

Solution Query :

```
mysql> SELECT E.EmployeeID, E.Name
    -> FROM Employee E
    -> LEFT JOIN Works_On W ON E.EmployeeID = W.EmployeeID
    -> WHERE W.ProjectID IS NULL;
Empty set (0.00 sec)
```

d) Write an SQL query to list each department along with the total number of projects managed by the department and the total number of employees assigned to it.

Solution Query :

```
mysql> SELECT D.DepartmentID,
    ->        D.DepartmentName,
    ->        COUNT(DISTINCT P.ProjectID) AS TotalProjects,
    ->        COUNT(DISTINCT E.EmployeeID) AS TotalEmployees
    -> FROM Department D
    -> LEFT JOIN Project P ON D.DepartmentID = P.DepartmentID
    -> LEFT JOIN Employee E ON D.DepartmentID = E.DepartmentID
    -> GROUP BY D.DepartmentID, D.DepartmentName;
+--------------+----------------+---------------+-----------------+
| DepartmentID | DepartmentName | TotalProjects | TotalEmployees |
+--------------+----------------+---------------+-----------------+
|           10 | HR             |             1 |               2 |
|           20 | IT             |             2 |               2 |
|           30 | Finance        |             1 |               2 |
+--------------+----------------+---------------+-----------------+
3 rows in set (0.00 sec)
```

WEEK 14

Case Study 2: (Trainee Record in a Institution)

A training institute requires a sophisticated database to effectively track the progress of trainees in their various training programs. Trainees enroll in different programs, such as Java Developer, Full Stack Developer, and Data Scientist. Each of these programs consists of several courses offered by the institute, each with its own unique code, title, and number of credit hours. Courses are overseen by an instructor, who may also be involved in teaching the course. In addition to the main instructor, each course has one or more teaching assistants who help facilitate the course. An instructor can manage and teach multiple courses across different programs, reflecting their expertise and versatility. The database captures detailed information about each course, including its unique code, title, credit hours, the instructor responsible for managing and teaching the course, the teaching assistants assigned, and the department to which the course belongs. Each course may have prerequisites, which means that some courses must be completed before others can be taken. This requirement ensures that trainees follow a structured learning path. Additionally, some training programs have mandatory courses that all trainees must complete to successfully finish the program.

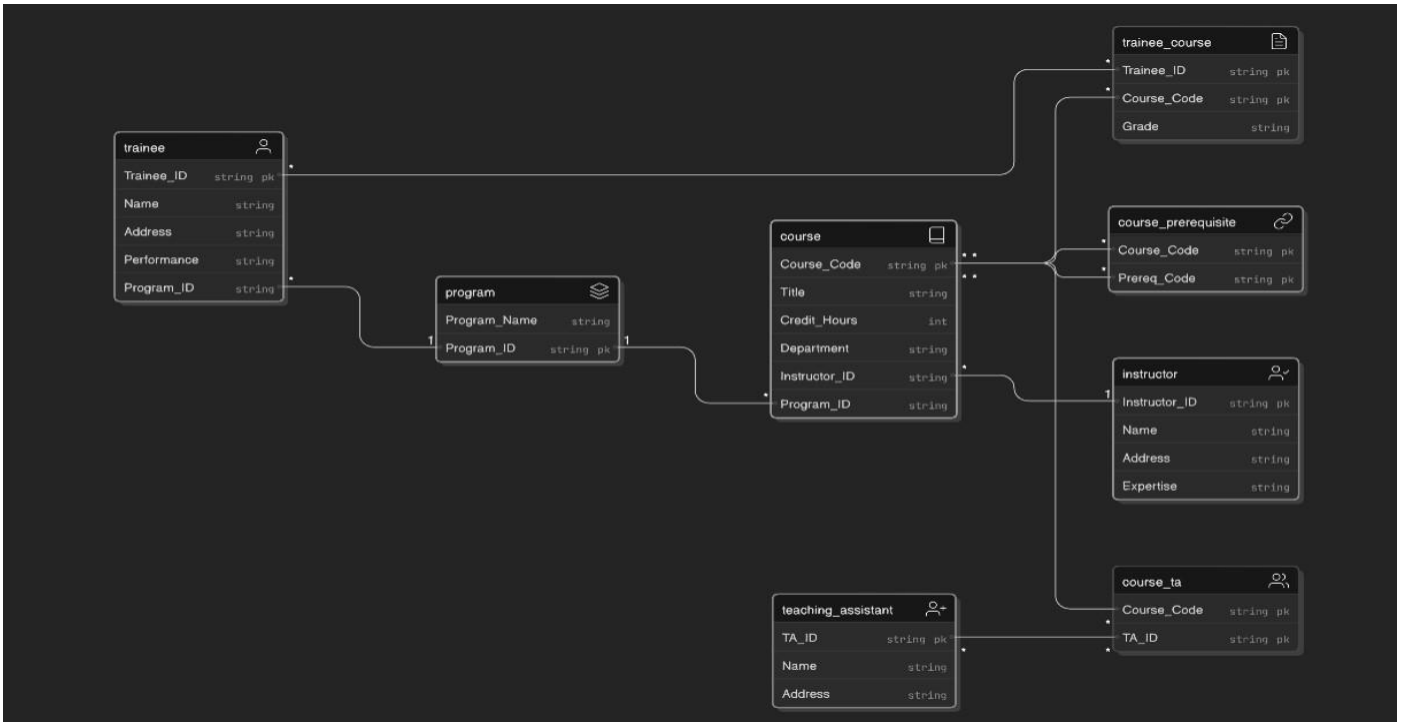
For trainees, the database records essential details such as their ID numbers, names, addresses, the training program they are enrolled in, and their academic performance. This includes information on the courses they have taken and the grades they received. By maintaining these records, the database helps monitor each trainee's progress and ensures they meet the necessary requirements for their chosen program. This database system plays a crucial role in managing the training institute's operations. It provides a clear view of course offerings, tracks the performance of trainees, and ensures that the educational requirements and prerequisites are properly enforced. This comprehensive approach allows the institute to deliver a well-organized training experience and supports both the instructors and trainees in achieving their educational goals.

Based on the details provided in the case study, address the following requirements:

Develop an ER diagram to model the training institute's database system, incorporating all key entities and their attributes. The primary entities to include are Trainees, Programs, Courses, Instructors, and Teaching Assistants. For each entity, specify attributes and primary keys: Trainees should have Trainee_ID, Name, Address, Program_ID, and Performance details. Courses should include Course_Code, Title, Credit_Hours, Prerequisites, and links to Instructors and Teaching Assistants.

a. Define the relationships between entities, such as Trainees being enrolled in Programs, Courses assigned to Programs, and Instructors and Teaching Assistants managing and teaching Courses. Include relationships for course prerequisites and mandatory courses within Programs.

b. Clearly specify key constraints like primary keys for each entity and cardinality and participation constraints for relationships. For instance, a Course may have multiple Teaching Assistants and prerequisites, while a Trainee can be enrolled in multiple Courses. Note: Add comments to highlight any constraints from the case study that are challenging to represent in the ER diagram. Ensure the diagram effectively captures the management of trainees' progress, course requirements, and instructor roles as outlined. Based on the provided case study, perform the following queries:



Solution: ER Diagram

a) Write an SQL query to show the names and IDs of trainees who have completed all the required courses for their program

Solution Query :

```
mysql> SELECT t.Trainee_ID, t.Name
    -> FROM Trainee t
    -> WHERE NOT EXISTS (
    ->     SELECT c.Course_Code
    ->     FROM Course c
    ->     WHERE c.Program_ID = t.Program_ID
    ->     AND c.Course_Code NOT IN (
    ->         SELECT Course_Code FROM Trainee_Course WHERE Trainee_ID = t.Trainee_ID
    ->     )
    -> );
+------------+--------+
| Trainee_ID | Name   |
+------------+--------+
|          1 | Rahul  |
|          2 | Sneha  |
|          3 | Aditya |
+------------+--------+
3 rows in set (0.01 sec)
```

b) Write an SQL query to get the names of instructors and the courses they teach, including any courses that have prerequisites

Solution Query :

```
mysql> SELECT i.Name AS Instructor, c.Course_Code, c.Title, cp.Prereq_Code
    -> FROM Instructor i
    -> JOIN Course c ON i.Instructor_ID = c.Instructor_ID
    -> LEFT JOIN Course_Prerequisite cp ON c.Course_Code = cp.Course_Code;
+--------------+-------------+------------------+-------------+
| Instructor   | Course_Code | Title            | Prereq_Code |
+--------------+-------------+------------------+-------------+
| Dr. Mehta    | J101        | Java Basics      | NULL        |
| Dr. Mehta    | J201        | Advanced Java    | J101        |
| Prof. Sharma | FS101       | HTML & CSS       | NULL        |
| Dr. Kapoor   | DS101       | Statistics       | NULL        |
| Dr. Kapoor   | DS201       | Machine Learning | DS101       |
+--------------+-------------+------------------+-------------+
5 rows in set (0.01 sec)
```

c) Write an SQL query to find the average grade for each course and list the courses where the average grade is below than 70%.

Solution Query :

```
mysql> SELECT c.Course_Code, c.Title, AVG(tc.Grade) AS Avg_Grade
    -> FROM Course c
    -> JOIN Trainee_Course tc ON c.Course_Code = tc.Course_Code
    -> GROUP BY c.Course_Code, c.Title
    -> HAVING AVG(tc.Grade) < 70;
Empty set (0.01 sec)
```

d) Write an SQL query to find the trainees who have signed up for the most courses, no matter which program they are in.

Solution Query :

```
mysql> SELECT Trainee_ID, COUNT(*) AS Course_Count
    -> FROM Trainee_Course
    -> GROUP BY Trainee_ID
    -> ORDER BY Course_Count DESC
    -> LIMIT 1;
+------------+--------------+
| Trainee_ID | Course_Count |
+------------+--------------+
|          1 |            2 |
+------------+--------------+
1 row in set (0.00 sec)
```

e) Write an SQL query to list all courses and the total hours all trainees have spent on each course.

Solution Query :

```
mysql> SELECT c.Course_Code, c.Title,
    ->        SUM(c.Credit_Hours) AS Total_Hours_Spent
    -> FROM Course c
    -> JOIN Trainee_Course tc ON c.Course_Code = tc.Course_Code
    -> GROUP BY c.Course_Code, c.Title;
+-------------+------------------+-------------------+
| Course_Code | Title            | Total_Hours_Spent |
+-------------+------------------+-------------------+
| DS101       | Statistics       |                 4 |
| DS201       | Machine Learning |                 4 |
| FS101       | HTML & CSS       |                 3 |
| J101        | Java Basics      |                 3 |
| J201        | Advanced Java    |                 4 |
+-------------+------------------+-------------------+
5 rows in set (0.00 sec)
```