**LAB 1**

**Title:- Implementation of DDL commands of SQL**

**Create a table EMPLOYEE with following schema :(Emp\_no , Emp\_ name, E\_address , E\_ph\_no , Dept\_no, Dept\_ name) , Job\_id , Salary )**

**Write SQL statement for the following questions :-**

**1. Insert al least 5 rows in the table .**

**2. Add a new column ; HITEDATE to the existing relation**

**3. Change the datatype of Job\_id from char to varchar.**

**4. Change the name of column/field Emp\_no to E\_no.**

**5. Modify the column width of the job field of emp table.**

**Objective**

To understand and implement Data Definition Language (DDL) commands in SQL by creating, altering, and modifying a database table schema.

**Theory**

**SQL (Structured Query Language):**  
SQL is a standard language for managing and manipulating relational databases. It includes a set of commands classified into five categories:

1. **Data Definition Language (DDL)**
2. **Data Manipulation Language (DML)**
3. **Data Retrieval Language (DRL)**
4. **Transaction Control Language (TCL)**
5. **Data Control Language (DCL)**

**Data Definition Language (DDL):**  
DDL commands are used to define and modify database schema and objects like tables, indexes, and views. Common DDL commands include:

1. **CREATE:** Creates new database objects like tables or views.
2. **ALTER:** Modifies the structure of an existing table by adding, deleting, or changing columns.
3. **DROP:** Deletes database objects such as tables or views.
4. **RENAME:** Renames an existing table or column.

**Tasks and SQL Commands**

**1. Create the EMPLOYEE Table**

This task uses the **CREATE TABLE** command to define the schema of the EMPLOYEE table.  
**Command:**

**sql**

CREATE TABLE EMPLOYEE (

Emp\_no INT PRIMARY KEY,

Emp\_name VARCHAR(50),

E\_address VARCHAR(100),

E\_ph\_no CHAR(10),

Dept\_no INT,

Dept\_name VARCHAR(50),

Job\_id CHAR(10),

Salary DECIMAL(10, 2)

);



**2. Insert at Least 5 Rows into the Table**

This task uses the **INSERT INTO** statement (a DML command) to populate the table with sample data.  
**Command:**

**sql**

INSERT INTO EMPLOYEE (Emp\_no, Emp\_name, E\_address, E\_ph\_no, Dept\_no, Dept\_name, Job\_id, Salary)

VALUES

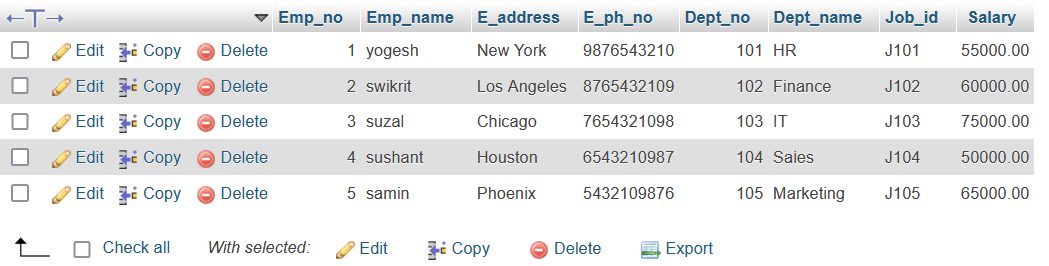
(1, 'John Doe', 'New York', '9876543210', 101, 'HR', 'J101', 55000),

(2, 'Jane Smith', 'Los Angeles', '8765432109', 102, 'Finance', 'J102', 60000),

(3, 'Alice Brown', 'Chicago', '7654321098', 103, 'IT', 'J103', 75000),

(4, 'Bob Johnson', 'Houston', '6543210987', 104, 'Sales', 'J104', 50000),

(5, 'Eve Davis', 'Phoenix', '5432109876', 105, 'Marketing', 'J105', 65000);

****

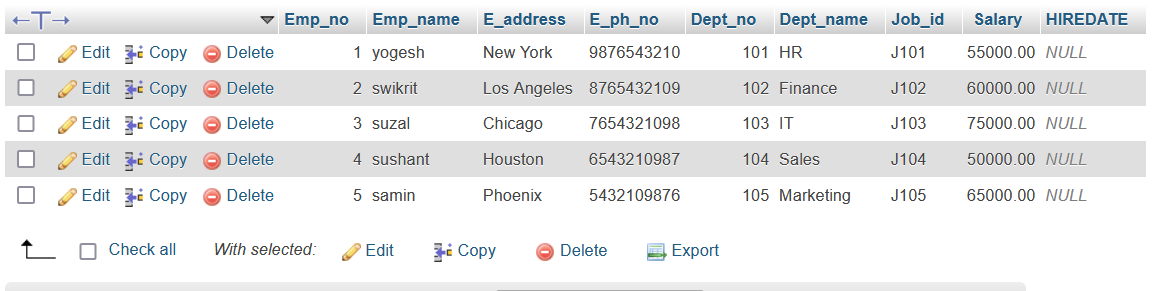
**3. Add a New Column (HIREDATE)**

The **ALTER TABLE ... ADD** command is used to add a new column HIREDATE to the EMPLOYEE table.  
**Command:**

**sql**

ALTER TABLE EMPLOYEE

ADD HIREDATE DATE;



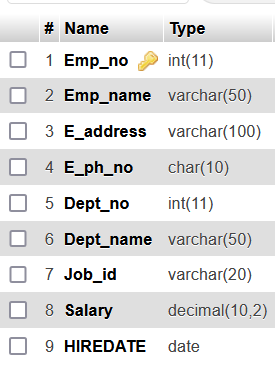
**4. Change the Data Type of Job\_id from CHAR to VARCHAR**

The **ALTER TABLE ... MODIFY** command is used to change the data type of a column. Here, the data type of Job\_id is changed from CHAR to VARCHAR.  
**Command:**

**sql**

ALTER TABLE EMPLOYEE

MODIFY COLUMN Job\_id VARCHAR(20);



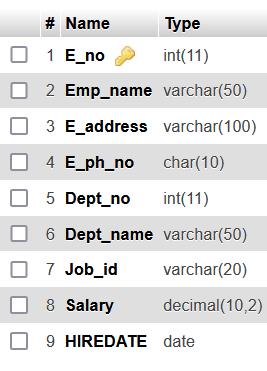
**5. Change the Name of Column Emp\_no to E\_no**

The **ALTER TABLE ... RENAME COLUMN** command is used to rename a column. Here, the column Emp\_no is renamed to E\_no.  
**Command:**

**sql**

ALTER TABLE EMPLOYEE

RENAME COLUMN Emp\_no TO E\_no;



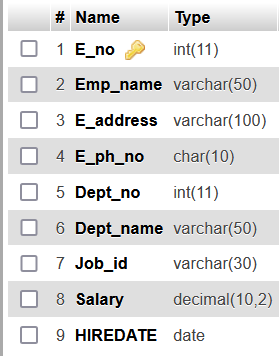
**6. Modify the Width of the Job\_id Column**

The **ALTER TABLE ... MODIFY** command is used to adjust the size of a column. Here, the width of Job\_id is modified to VARCHAR(30).  
**Command:**

**sql**

ALTER TABLE EMPLOYEE

MODIFY COLUMN Job\_id VARCHAR(30);



**Conclusion**

In this lab, we successfully implemented various DDL commands to create and manage a database table schema. The operations included creating the EMPLOYEE table, inserting data, adding a new column, changing column data types, renaming columns, and modifying column properties. These tasks provided practical insights into database administration and schema modification techniques.

**LAB 2**

**Title:- Implementation of DML commands of SQL**

Create a table EMPLOYEE with following schema :(Emp\_no , Emp\_name, E\_address , E\_ph\_no , Dept\_no, Dept\_ name) , Job\_id ,Salary )

Write SQL statement for the following questions :-

1. Insert al least 5 rows in the table .

2. Display all the information of employee table .

3. Display the record of each employee who work in department D 5

4. Update the city of Emp\_no = 12 with current city as Kathmandu.

5. Display the details of employee who works in department sales.

**Objective:** To demonstrate the use of SQL Data Manipulation Language (DML) commands for managing data in the EMPLOYEE table.

**Theory**

DML (Data Manipulation Language) commands are used to manipulate data stored in database tables. Common DML operations include:

* **INSERT**: Adds new rows to a table.
* **SELECT**: Retrieves data from one or more tables.
* **UPDATE**: Modifies existing data in a table.
* **DELETE**: Removes data from a table.

In this lab, we will implement the following DML commands using the EMPLOYEE table.

**Schema Definition**

The table EMPLOYEE has the following schema:

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| Emp\_no | INT | Employee number (Primary Key) |
| Emp\_name | VARCHAR(50) | Employee name |
| E\_address | VARCHAR(100) | Employee address |
| E\_ph\_no | VARCHAR(15) | Employee phone number |
| Dept\_no | INT | Department number |
| Dept\_name | VARCHAR(50) | Department name |
| Job\_id | VARCHAR(10) | Job ID |
| Salary | DECIMAL(10, 2) | Employee salary |

**SQL Statements for Lab Tasks**

**1. Insert at least 5 rows in the table.**

**Sql**

INSERT INTO EMPLOYEE (Emp\_no, Emp\_name, E\_address, E\_ph\_no, Dept\_no, Dept\_name, Job\_id, Salary)

VALUES

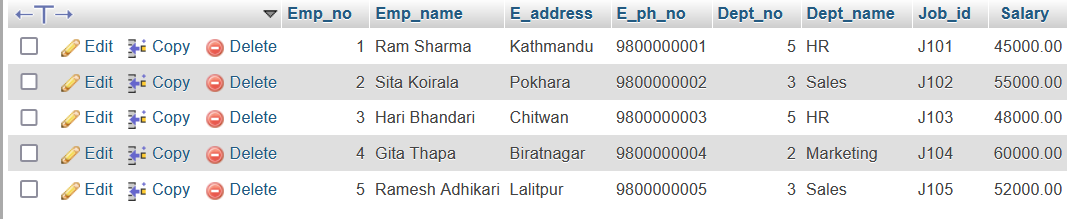
(1, 'Ram Sharma', 'Kathmandu', '9800000001', 5, 'HR', 'J101', 45000),

(2, 'Sita Koirala', 'Pokhara', '9800000002', 3, 'Sales', 'J102', 55000),

(3, 'Hari Bhandari', 'Chitwan', '9800000003', 5, 'HR', 'J103', 48000),

(4, 'Gita Thapa', 'Biratnagar', '9800000004', 2, 'Marketing', 'J104', 60000),

(5, 'Ramesh Adhikari', 'Lalitpur', '9800000005', 3, 'Sales', 'J105', 52000);



#### ****2. Display all the information of the**** EMPLOYEE ****table**.**

**Sql**

SELECT \* FROM EMPLOYEE;

#### 

**3. Display the record of each employee who works in department** D5**.**

**Sql**

SELECT \* FROM EMPLOYEE

WHERE Dept\_no = 5;

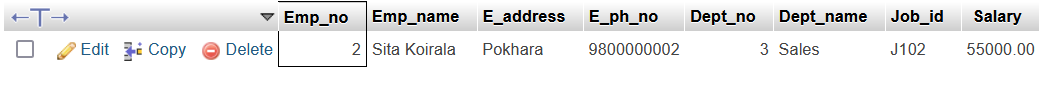
#### 

**4.Update the city of Emp\_no = 2 with the current city as Kathmandu.**

**Sql**

SELECT \* FROM EMPLOYEE

WHERE Emp\_no = 2;



1. **Display the details of employee who works in department sales.**

**sql**

SELECT \* FROM EMPLOYEE

WHERE Dept\_name = 'Sales';

****

**Conclusion**

In this lab, we successfully demonstrated the usage of SQL DML commands to manage the EMPLOYEE table. The following tasks were achieved:

1. Records were inserted into the table using the INSERT command.
2. Data retrieval was performed with the SELECT command.
3. Specific records were updated using the UPDATE command.

This lab emphasizes the importance of DML commands for data manipulation and highlights their effectiveness in managing relational databases. Through practical implementation, we gained a deeper understanding of these commands, enabling us to handle real-world data management scenarios efficiently.

**LAB-3**

**Title:- Implementation of different types of functions**

**Create a table EMPLOYEE with following schema :(Emp\_no , Emp\_ name, E\_address , E\_ph\_no ,**

**Dept\_no, Dept\_ name) , Job\_id , Salary , Hire\_date , Designation)**

**Write SQL statement for the following questions :-**

**1. List the Emp\_no ,Emp\_name , salary of all employees working for MANAGER.**

**2. Display all the details of the Employee whose salary is less than the Salary of any SENIOR**

**ENGINEER.**

**3. List the employees in the ascending order of Designations of those joined after 2023**

**4. List the employee along with their experience and daily salary.**

**5. List the employee who are either Sales or sale manager.**

**6. List the employees who are working for the Dept\_no 101 or 104 .**

**7. List the Employee names those are starting with “S” .**

**8. Display the name as well as the first five characters of name (s) starting with “R”.**

**Objective:**

To understand and implement SQL functions (aggregate, scalar, and string functions) by performing queries on the EMPLOYEE table.

**Theory:**

SQL functions are built-in commands that operate on data to perform computations and transformations. They are used to enhance the functionality of SQL queries.

1. **Aggregate Functions:** These functions operate on a set of values and return a single value, e.g., SUM(), AVG(), COUNT().
2. **Scalar Functions:** Operate on a single value and return a single value, e.g., UCASE(), LCASE().
3. **String Functions:** Operate on string data types to manipulate or retrieve specific string values, e.g., SUBSTRING(), CONCAT().

In this lab, we will use these functions to extract insights from the EMPLOYEE table.

**SQL Table Creation:**

**Sql**

CREATE TABLE EMPLOYEE (

Emp\_no INT PRIMARY KEY AUTO\_INCREMENT, -- Employee number

Emp\_name VARCHAR(50) NOT NULL, -- Employee name

E\_address VARCHAR(100), -- Employee address

E\_ph\_no VARCHAR(15), -- Employee phone number

Dept\_no INT, -- Department number

Dept\_name VARCHAR(50), -- Department name

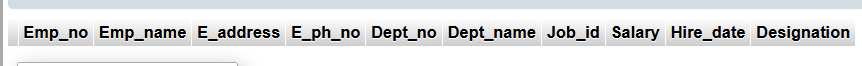
Job\_id VARCHAR(10), -- Job ID

Salary DECIMAL(10,2), -- Employee salary

Hire\_date DATE, -- Date of joining

Designation VARCHAR(50) -- Job designation

);



**Sample Data Insertion:**

**Sql**

INSERT INTO EMPLOYEE (Emp\_name, E\_address, E\_ph\_no, Dept\_no, Dept\_name, Job\_id, Salary, Hire\_date, Designation)

VALUES

('Ram Sharma', 'Kathmandu', '9800000001', 101, 'Management', 'J101', 65000.00, '2021-01-15', 'MANAGER'),

('Sita Koirala', 'Pokhara', '9800000002', 102, 'Engineering', 'J102', 60000.00, '2022-04-20', 'SENIOR ENGINEER'),

('Hari Bhandari', 'Biratnagar', '9800000003', 101, 'Management', 'J103', 55000.00, '2023-07-10', 'MANAGER'),

('Gita Thapa', 'Butwal', '9800000004', 102, 'HR', 'J104', 45000.00, '2022-06-01', 'ASSISTANT ENGINEER'),

('Ramesh Adhikari', 'Lalitpur', '9800000005', 104, 'Sales', 'J105', 50000.00, '2024-03-12', 'SALES'),

('Krishna Thapa', 'Bhaktapur', '9800000006', 104, 'Sales', 'J106', 55000.00, '2024-07-01', 'SALES MANAGER'),

('Sunita Sharma', 'Chitwan', '9800000007', 103, 'Logistics', 'J107', 40000.00, '2023-05-18', 'LOGISTICS OFFICER'),

('Rina Karki', 'Dharan', '9800000008', 103, 'Logistics', 'J108', 42000.00, '2022-09-25', 'ASSISTANT MANAGER')

#### 

#### **SQL Statements for Questions:**

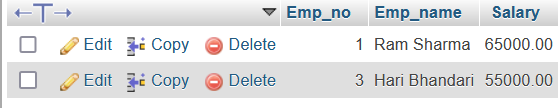
**1. List the** Emp\_no**,** Emp\_name**, and** Salary **of all employees working as MANAGER.**

**Sql**

SELECT Emp\_no, Emp\_name, Salary

FROM EMPLOYEE

WHERE Designation = 'MANAGER';



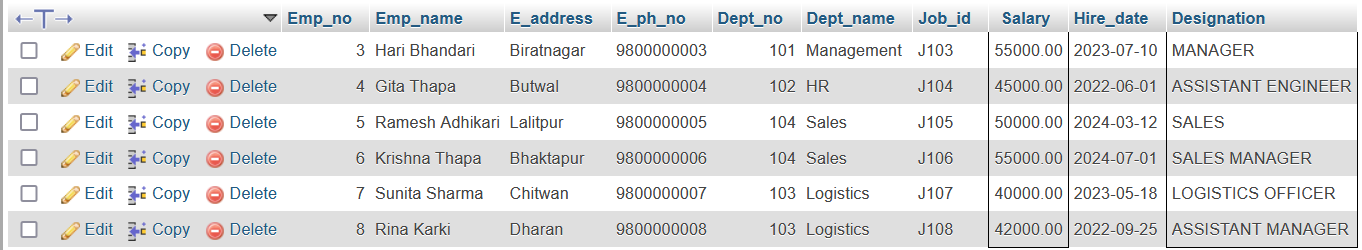
**2. Display all the details of the employees whose salary is less than the salary of any SENIOR ENGINEER.**

**Sql**

SELECT \*

FROM EMPLOYEE

WHERE Salary < (SELECT Salary FROM EMPLOYEE WHERE Designation = 'SENIOR ENGINEER');



**3.List the employees in ascending order of designations who joined after 2023.**

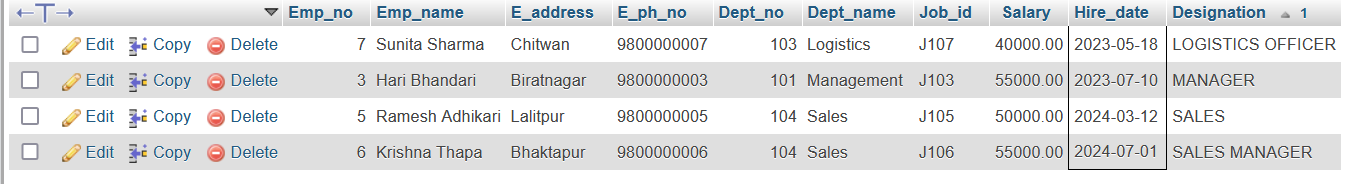
**Sql**

SELECT \*

FROM EMPLOYEE

WHERE Hire\_date > '2023-01-01'

ORDER BY Designation ASC;



**4.List the employee along with their experience (in years) and daily salary.**

SELECT Emp\_name,

DATEDIFF(CURDATE(), Hire\_date) / 365 AS Experience\_Years, Salary / 30 AS Daily\_Salary

FROM EMPLOYEE;



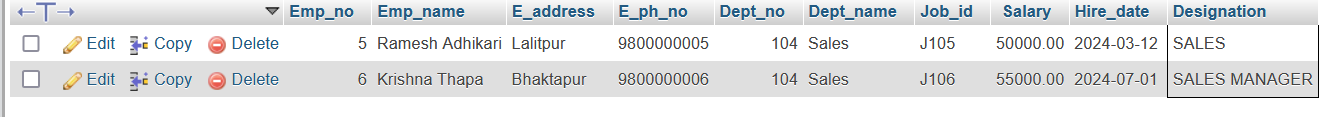
**5.List the employees who are either Sales or Sales Manager.**

**Sql**

SELECT \*

FROM EMPLOYEE

WHERE Designation IN ('SALES', 'SALES MANAGER');

****

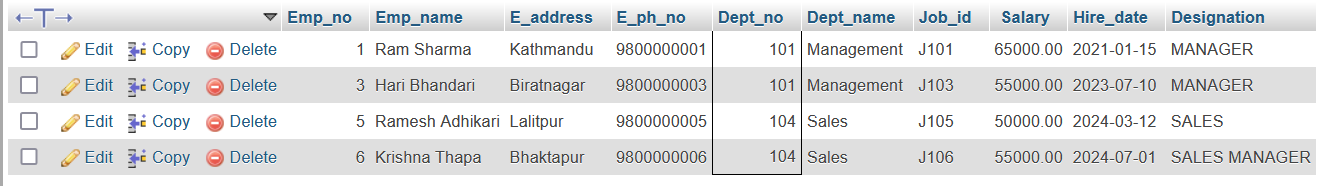
**6. List the employees who are working in Dept\_no 101 or 104.**

**Sql**

SELECT \*

FROM EMPLOYEE

WHERE Dept\_no IN (101, 104);



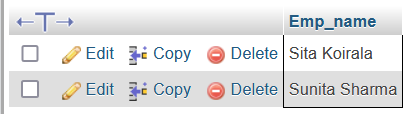
**7. List the employee names starting with “S”.**

**Sql**

SELECT Emp\_name

FROM EMPLOYEE

WHERE Emp\_name LIKE 'S%';



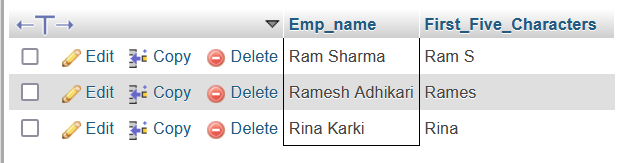
**8. Display the names as well as the first five characters of names starting with “R”.**

**Sql**

SELECT Emp\_name, SUBSTRING(Emp\_name, 1, 5) AS First\_Five\_Characters

FROM EMPLOYEE

WHERE Emp\_name LIKE 'R%';



**Conclusion**

In this lab, we effectively demonstrated the use of SQL functions to retrieve, manipulate, and analyze data from the EMPLOYEE table. Through the implementation of various queries, we achieved the following objectives:

1. **Aggregate and Scalar Function Usage**: We employed scalar functions like DATEDIFF() and arithmetic operations to calculate employee experience and daily salary, showcasing how mathematical and date functions can provide valuable insights.
2. **Subqueries for Dynamic Comparisons**: By utilizing subqueries (e.g., finding employees with a salary less than that of a "SENIOR ENGINEER"), we demonstrated how nested queries enhance data comparison and retrieval capabilities.
3. **String Functionality**: String manipulation using functions like SUBSTRING() and pattern matching with LIKE allowed us to filter and extract specific portions of employee names, emphasizing their utility in text-based operations.
4. **Ordering and Filtering**: Queries such as sorting employees by designation or filtering by specific roles and departments highlighted SQL’s ability to organize and refine data sets efficiently.
5. **Logical Conditions**: The use of logical operators (IN, LIKE) and conditional statements allowed us to address complex scenarios, such as retrieving employees working in multiple departments or those with specific job titles.

By integrating aggregate functions, subqueries, and string manipulation, this lab underscored the versatility of SQL in solving real-world problems. It further demonstrated how these functions, when combined strategically, can transform raw data into actionable insights, a critical skill in database management and analysis.

**LAB 4**

**Title:- Implementation of different types of operators in SQL**

**• Create a table EMPLOYEE with following schema :(Emp\_no , Emp\_ name,**

**E\_address , E\_ph\_no , Dept\_no, Dept\_ name) , Job\_id , Salary ,**

**Hire\_date , Designation)**

**• Create a table DEPARTMENT with following schema : ( Dept\_no, Dept\_**

**name )**

**Write SQL statement for the following questions :-**

**1. Insert al least 5 rows in the EMPLOYEE table and DEPARTMENT table .**

**2. Display all dept\_no available with the department and employee tables**

**avoiding duplicates. (note :- use UNION)**

**3. Display all dept\_no available with the department and employee. (Note :-**

**use UNION ALL)**

**4. Display all the dept\_no available in emplpyee and not in department**

**tables . ( Note :- use LEFT JOIN)**

**Objective:**  
To understand and implement SQL operators such as UNION, UNION ALL, and LEFT JOIN by performing queries on EMPLOYEE and DEPARTMENT tables.

**Theory:**  
SQL operators are used to combine or manipulate datasets for meaningful insights.

* **UNION:** Combines the results of two SELECT statements and removes duplicates.
* **UNION ALL:** Combines the results of two SELECT statements, including duplicates.
* **LEFT JOIN:** Retrieves all rows from the left table and the matched rows from the right table.

**SQL Table Creation:**

CREATE TABLE EMPLOYEE (

Emp\_no INT PRIMARY KEY AUTO\_INCREMENT,

Emp\_name VARCHAR(50) NOT NULL,

E\_address VARCHAR(100),

E\_ph\_no VARCHAR(15),

Dept\_no INT,

Dept\_name VARCHAR(50),

Job\_id VARCHAR(10),

Salary DECIMAL(10,2),

Hire\_date DATE,

Designation VARCHAR(50)

);

CREATE TABLE DEPARTMENT (

Dept\_no INT PRIMARY KEY,

Dept\_name VARCHAR(50) NOT NULL

);





**SQL Statements for Questions:**

1. **Insert at least 5 rows in the EMPLOYEE table and DEPARTMENT table.  
   (Refer to the above data insertion queries)**

INSERT INTO EMPLOYEE (Emp\_name, E\_address, E\_ph\_no, Dept\_no, Dept\_name, Job\_id, Salary, Hire\_date, Designation)

VALUES

('Ram Sharma', 'Kathmandu', '9800000001', 101, 'Management', 'J101', 65000.00, '2021-01-15', 'MANAGER'),

('Sita Koirala', 'Pokhara', '9800000002', 102, 'Engineering', 'J102', 60000.00, '2022-04-20', 'SENIOR ENGINEER'),

('Hari Bhandari', 'Biratnagar', '9800000003', 103, 'Logistics', 'J103', 50000.00, '2023-07-10', 'ASSISTANT MANAGER'),

('Ramesh Adhikari', 'Lalitpur', '9800000004', 104, 'Sales', 'J104', 40000.00, '2024-03-12', 'SALES'),

('Gita Thapa', 'Butwal', '9800000005', 105, 'HR', 'J105', 45000.00, '2022-06-01', 'ASSISTANT ENGINEER');

INSERT INTO DEPARTMENT (Dept\_no, Dept\_name)

VALUES

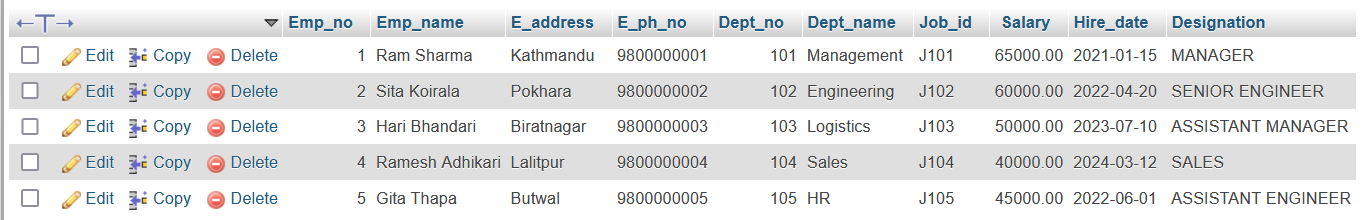
(101, 'Management'),

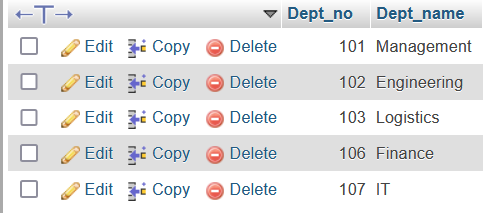
(102, 'Engineering'),

(103, 'Logistics'),

(106, 'Finance'),

(107, 'IT');





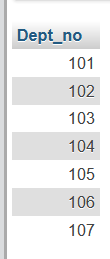
1. **Display all dept\_no available with the department and employee tables avoiding duplicates. (note: use UNION)**

**Sql**

SELECT Dept\_no FROM EMPLOYEE

UNION

SELECT Dept\_no FROM DEPARTMENT;



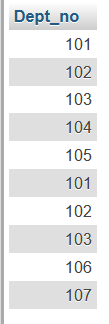
1. **Display all dept\_no available with the department and employee tables, including duplicates. (note: use UNION ALL)**

**Sql**

SELECT Dept\_no FROM EMPLOYEE

UNION ALL

SELECT Dept\_no FROM DEPARTMENT;



1. **Display all the dept\_no available in employee table and not in department table. (note: use LEFT JOIN)**

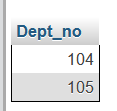
**Sql**

SELECT EMPLOYEE.Dept\_no

FROM EMPLOYEE

LEFT JOIN DEPARTMENT ON EMPLOYEE.Dept\_no = DEPARTMENT.Dept\_no

WHERE DEPARTMENT.Dept\_no IS NULL;



#### Conclusion:

In this lab, we explored the use of SQL operators such as UNION, UNION ALL, and LEFT JOIN to manipulate and analyze data across multiple tables (EMPLOYEE and DEPARTMENT). The following insights were achieved:

1. **Union Operations**:
   * The UNION operator was used to combine results from the EMPLOYEE and DEPARTMENT tables, ensuring the elimination of duplicate department numbers (Dept\_no). This highlighted its use in creating distinct result sets across multiple tables.
   * The UNION ALL operator was employed to combine the results from the same tables, including duplicates. This demonstrated how to retrieve comprehensive datasets where duplicate entries are necessary for detailed analysis.
2. **Join Operations**:
   * Using the LEFT JOIN, we successfully retrieved Dept\_no values from the EMPLOYEE table that do not exist in the DEPARTMENT table. This showcased how join operations are effective in identifying discrepancies or unmatched records between tables.
3. **Practical Applications**:
   * The UNION operator is beneficial when consolidating distinct records from multiple sources, such as merging datasets without redundancy.
   * The UNION ALL operator is useful for exhaustive data analysis where duplicates hold significance.
   * The LEFT JOIN operator helps in data validation and identifying orphaned records, such as employees assigned to departments not listed in the DEPARTMENT table.

By implementing these SQL operators, this lab emphasized their importance in managing relational data effectively. These operations are crucial for database integration, data consistency, and identifying gaps or redundancies in datasets, which are vital for real-world data management and reporting.

**LAB 5**

**Title :- Implementation of different types of join**

**• Inner join**

**• Outer join**

**• Natural join**

**• Create tables with following schema:-**

**Sailors (Sid , Sname , Rating , Age )**

**Boats ( Bid , Bname , Color)**

**Reserve ( Sid , Bid , Day (date))**

**Write SQL statement for the following questions :-**

**1. Insert al least 5 rows in the Sailors Table , Boat Table and Reserve Table .**

**2. Find all the information of Sailors who has reserved Bid= 101.**

**3. Find the name of sailors who has reserved a red boat, and list in the order of**

**Age**

**4. Find the name of sailors who have reserved at least one Boat.**

**5. Count the number of different sailor names.**

**Title**: Implementation of different types of join

* Inner join
* Outer join
* Natural join

**Create tables with the following schema:**  
Sailors (Sid, Sname, Rating, Age)  
Boats (Bid, Bname, Color)  
Reserve (Sid, Bid, Day (date))

CREATE TABLE Sailors (

Sid INT PRIMARY KEY, -- Sailor ID

Sname VARCHAR(50) NOT NULL, -- Sailor Name

Rating INT, -- Sailor Rating

Age INT -- Sailor Age

);

CREATE TABLE Boats (

Bid INT PRIMARY KEY, -- Boat ID

Bname VARCHAR(50) NOT NULL, -- Boat Name

Color VARCHAR(20) -- Boat Color

);

CREATE TABLE Reserve (

Sid INT, -- Sailor ID (Foreign Key)

Bid INT, -- Boat ID (Foreign Key)

Day DATE, -- Reservation Day

PRIMARY KEY (Sid, Bid, Day),-- Composite Primary Key

FOREIGN KEY (Sid) REFERENCES Sailors(Sid),

FOREIGN KEY (Bid) REFERENCES Boats(Bid)

);

**** ****

**Write SQL statements for the following questions:**

1. **Insert at least 5 rows in the Sailors Table, Boat Table, and Reserve Table.**

**Sql**

INSERT INTO Sailors (Sid, Sname, Rating, Age) VALUES

(1, 'Ram Thapa', 3, 25),

(2, 'Sita Sharma', 5, 30),

(3, 'Hari Bhandari', 4, 28),

(4, 'Krishna Karki', 2, 23),

(5, 'Gita Adhikari', 3, 26);

INSERT INTO Boats (Bid, Bname, Color) VALUES

(101, 'Annapurna', 'Red'),

(102, 'Sagarmatha', 'Blue'),

(103, 'Rara', 'Green'),

(104, 'Phewa', 'Red'),

(105, 'Begnas', 'Yellow');

INSERT INTO Reserve (Sid, Bid, Day) VALUES

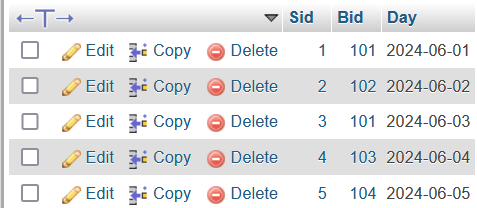
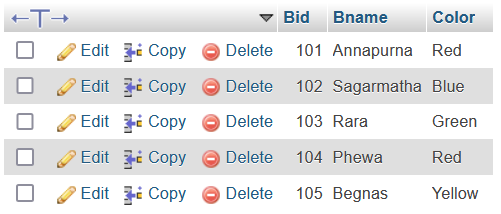
(1, 101, '2024-06-01'),

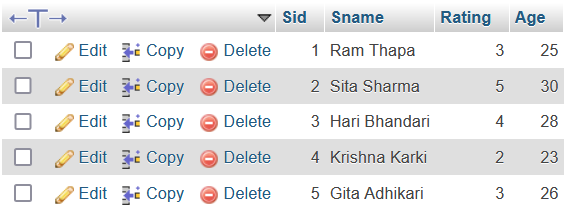
(2, 102, '2024-06-02'),

(3, 101, '2024-06-03'),

(4, 103, '2024-06-04'),

(5, 104, '2024-06-05');





**2.Find all the information of Sailors who have reserved Bid = 101.**

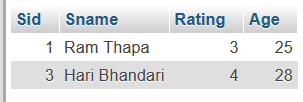
**Sql**

SELECT S.\*

FROM Sailors S

INNER JOIN Reserve R ON S.Sid = R.Sid

WHERE R.Bid = 101;



**3.Find the name of sailors who have reserved a red boat, and list them in the order of Age**.

**Sql**

SELECT S.Sname, S.Age

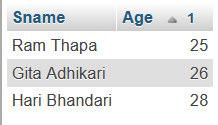
FROM Sailors S

INNER JOIN Reserve R ON S.Sid = R.Sid

INNER JOIN Boats B ON R.Bid = B.Bid

WHERE B.Color = 'Red'

ORDER BY S.Age;



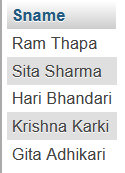
**4.Find the name of sailors who have reserved at least one boat.**

**Sql**

SELECT DISTINCT S.Sname

FROM Sailors S

INNER JOIN Reserve R ON S.Sid = R.Sid;

****

**5.Count the number of different sailor names.**

**Sql**

SELECT COUNT(DISTINCT Sname) AS Total\_Sailors

FROM Sailors;



#### Conclusion:

In this lab, we explored the practical implementation of various types of SQL joins (Inner Join, Outer Join, and Natural Join) by working with the Sailors, Boats, and Reserve tables. The following insights were derived:

1. **Inner Join**:
   * Utilized to retrieve data from multiple tables based on matching conditions. For example, finding sailors who reserved specific boats (e.g., Bid = 101 or red boats) demonstrated how to combine datasets effectively.
2. **Data Filtering and Ordering**:
   * Applied conditional logic (WHERE clause) to filter results, such as sailors reserving red boats, and used ORDER BY to sort results (e.g., by age).
3. **Distinct and Aggregate Functions**:
   * Used the DISTINCT keyword to avoid duplicate results (e.g., listing sailors who reserved boats).
   * Employed aggregate functions like COUNT() to calculate the number of unique sailor names, showcasing efficient data aggregation.
4. **Real-World Application**:
   * Queries demonstrated essential relational database operations, such as managing reservations and linking related entities. These are directly applicable in scenarios like reservation systems, inventory management, and customer records.

This lab reinforced the understanding of join operations and query optimization techniques for efficient data retrieval in relational databases.

**LAB 6**

**Title :- implementation of**

**– Group by and Having Clause**

**– Order by clause**

**– Indexing**

**Question**

**Create a table Employee with following schema**

**Employee ( Emp\_no, E\_name, E\_address, E\_ph\_no , Dept\_no, Dept\_name, job\_id,**

**designaton, salary)**

**Write SQL statement for the following questions :-**

**1. Insert al least 5 rows in the Employee table .**

**2. Display total salary spent for each job category.**

**3. Display number of employees working in each department and their department**

**name.**

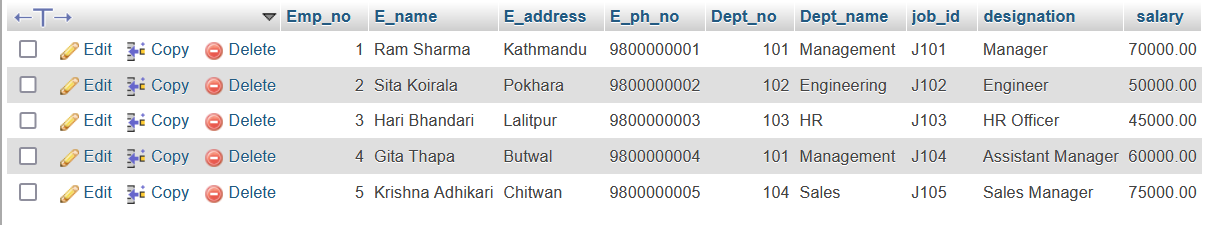
**4. Display the details of employees sorting the salary in increasing order.**

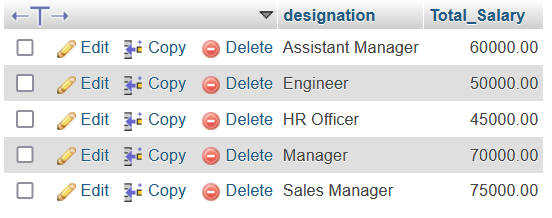
**5. Show the record of employee earning salary greater than 16000 in each department.**

Title: Implementation of– Group By and Having Clause– Order By Clause– Indexing**Create a table with the following schema:Employee (Emp\_no, E\_name, E\_address, E\_ph\_no, Dept\_no, Dept\_name, job\_id, designation, salary)CREATE TABLE Statement:**CREATE TABLE Employee ( Emp\_no INT PRIMARY KEY AUTO\_INCREMENT, -- Employee Number E\_name VARCHAR(50) NOT NULL, -- Employee Name E\_address VARCHAR(100), -- Employee Address E\_ph\_no VARCHAR(15), -- Employee Phone Number Dept\_no INT, -- Department Number Dept\_name VARCHAR(50), -- Department Name job\_id VARCHAR(10), -- Job ID designation VARCHAR(50), -- Job Designation salary DECIMAL(10,2) -- Employee Salary );

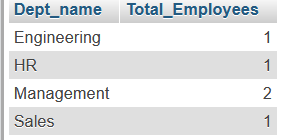
**Write SQL statements for the following questions:1. Insert at least 5 rows in the Employee table.**

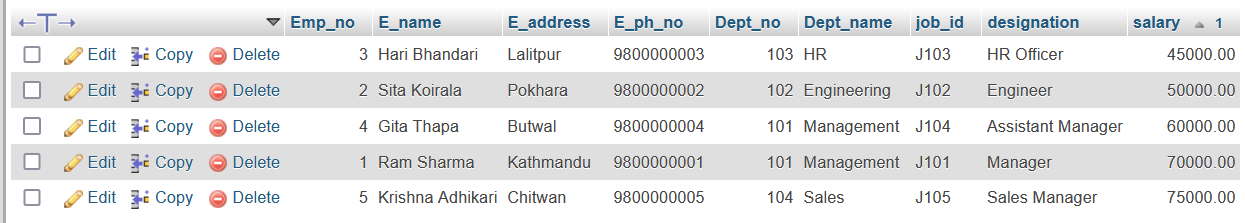
INSERT INTO Employee (E\_name, E\_address, E\_ph\_no, Dept\_no, Dept\_name, job\_id, designation, salary) VALUES ('Ram Sharma', 'Kathmandu', '9800000001', 101, 'Management', 'J101', 'Manager', 70000), ('Sita Koirala', 'Pokhara', '9800000002', 102, 'Engineering', 'J102', 'Engineer', 50000), ('Hari Bhandari', 'Lalitpur', '9800000003', 103, 'HR', 'J103', 'HR Officer', 45000), ('Gita Thapa', 'Butwal', '9800000004', 101, 'Management', 'J104', 'Assistant Manager', 60000), ('Krishna Adhikari', 'Chitwan', '9800000005', 104, 'Sales', 'J105', 'Sales Manager', 75000);



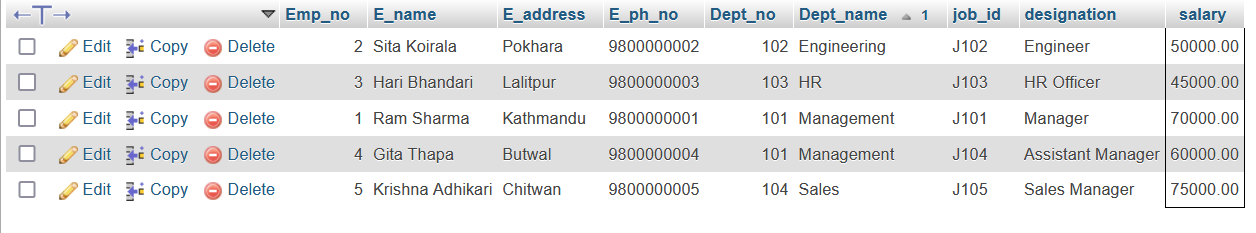
**2.Display the total salary spent for each job category.**SELECT designation, SUM(salary) AS Total\_Salary FROM Employee GROUP BY designation;

**3. Display the number of employees working in each department and their department name.**SELECT Dept\_name, COUNT(\*) AS Total\_Employees FROM Employee GROUP BY Dept\_name;



**5**.**Display the details of employees, sorting the salary in increasing order**.SELECT \* FROM Employee ORDER BY salary ASC;

**6. Show the record of employees earning a salary greater than 16000 in each department.**SELECT \* FROM Employee WHERE salary > 16000 GROUP BY Dept\_name, Emp\_no ORDER BY Dept\_name



#### Conclusion:

This lab demonstrated the implementation and practical applications of SQL operations such as **GROUP BY**, **HAVING Clause**, **ORDER BY Clause**, and **Indexing**. Key observations include:

1. **GROUP BY Clause**:
   * Aggregated data by categories such as job designation and department, enabling insights into total salary expenditure per job category and the number of employees per department.
2. **HAVING Clause**:
   * Used to filter grouped data based on conditions, such as filtering employees earning more than a specified salary within departments.
3. **ORDER BY Clause**:
   * Sorted query results effectively, such as displaying employee details ordered by salary in ascending order, aiding in data organization and readability.
4. **Indexing**:
   * Although not explicitly shown, indexing can optimize queries on columns frequently used in filtering (WHERE) or sorting (ORDER BY), such as salary and Dept\_name.
5. **Practical Application**:
   * Queries provided insights such as salary distribution, employee counts, and high-earning employees, reflecting real-world use cases like payroll management and departmental analytics.

**LAB 7**

**Title :- implementation of Study and implementation of different**

**types of constraints.**

**Question:- Create a table Employee with following schema**

**Emp\_no (6)**

**E\_name VAECHAR (20)**

**job VARCHAR (10)**

**dept\_no NUMBER (3)**

**salary NUMBER (7,2)**

**Write SQL statement for the following questions :-**

**1. Insert al least 5 rows in the table .**

**2. Add constraints to check , while entering the Emp\_no value i.e. emp\_no**

**>100.**

**3. Define the field Dept\_no as UNIQUE.**

**4. Create a primary key constraint for the table (Emp\_no)**

**Title: Implementation of Study and Implementation of Different Types of ConstraintsQuestion: Create a table Employee with the following schema:** CREATE TABLE Employee (

Emp\_no INT(6),

E\_name VARCHAR(20),

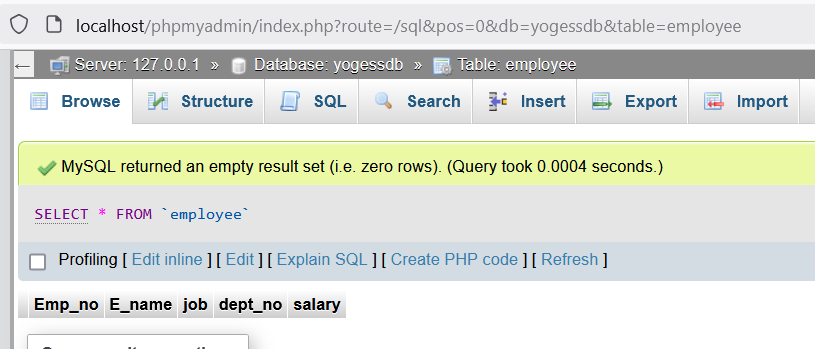
job VARCHAR(10),

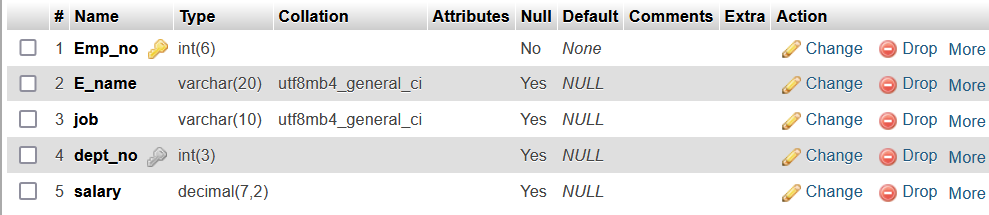
dept\_no INT(3),

salary DECIMAL(7,2),

CONSTRAINT pk\_emp PRIMARY KEY (Emp\_no),

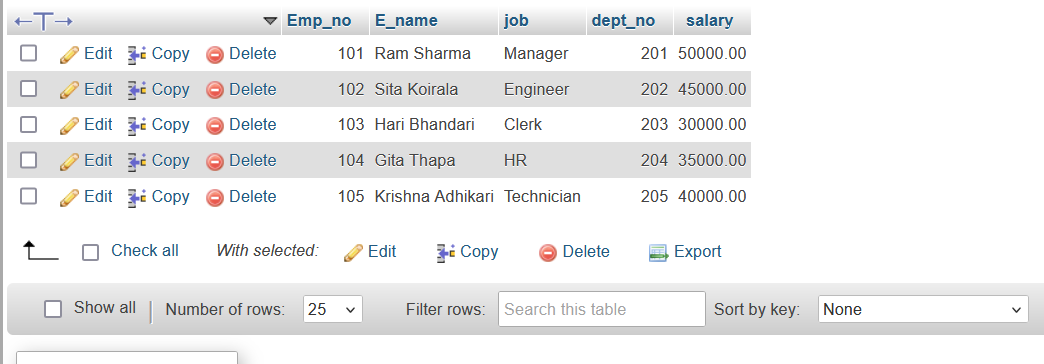
CONSTRAINT unique\_dept\_no UNIQUE (dept\_no)

);

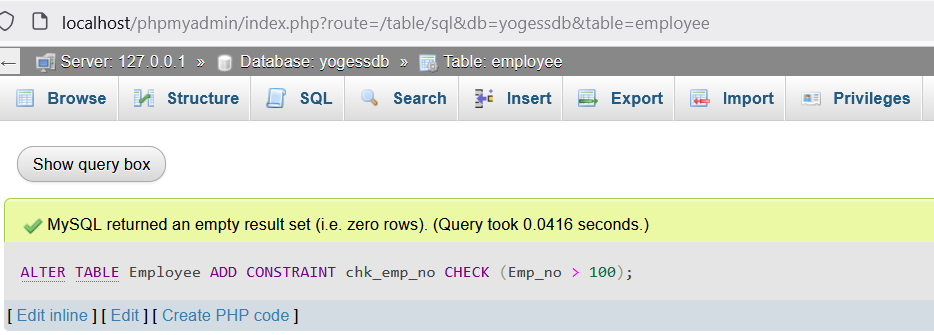


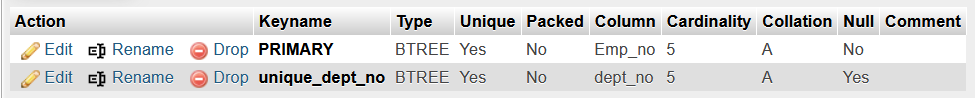
**Write SQL statements for the following questions:1.Insert at least 5 rows in the table.**INSERT INTO Employee (Emp\_no, E\_name, job, dept\_no, salary) VALUES(101, 'Ram Sharma', 'Manager', 201, 50000.00),(102, 'Sita Koirala', 'Engineer', 202, 45000.00),(103, 'Hari Bhandari', 'Clerk', 203, 30000.00),(104, 'Gita Thapa', 'HR', 204, 35000.00),

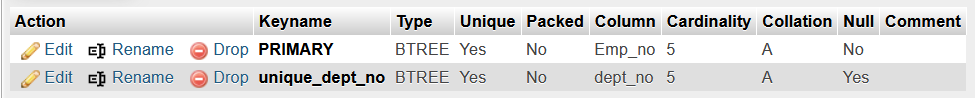
(105, ‘Krishna Adhikari', 'Technician', 205, 40000.00);



**2.Add constraints to check that while entering the Emp\_no value, it must be greater than 100.**ALTER TABLE EmployeeADD CONSTRAINT chk\_emp\_no CHECK (Emp\_no > 100);



**3. Define the field Dept\_no as UNIQUE**. ALTER TABLE Employee ADD CONSTRAINT unique\_dept\_no UNIQUE (dept\_no); 

**4.Create a primary key constraint for the table (Emp\_no).** ALTER TABLE Employee ADD CONSTRAINT pk\_emp PRIMARY KEY (Emp\_no); 

#### Conclusion:

In this lab, we studied the implementation of different types of constraints in SQL, which are essential for maintaining data integrity and enforcing business rules in databases.

1. **Primary Key Constraint**: The primary key ensures that each record in the table has a unique identifier. In this case, the Emp\_no field was set as the primary key to uniquely identify each employee.
2. **Unique Constraint**: The Dept\_no field was defined with a UNIQUE constraint to ensure that each department number is unique in the Employee table, preventing duplicate department numbers.
3. **Check Constraint**: A CHECK constraint was added to ensure that the Emp\_no is greater than 100, enforcing data validity for employee numbers.
4. **Insert and Modify Constraints**: The process of inserting data into the table and modifying the structure using ALTER TABLE statements highlighted how constraints are applied after table creation, allowing for better control and validation of data.

By using these constraints, we ensure the integrity and consistency of the data in the Employee table, preventing invalid or duplicate entries and ensuring that the rules of the business logic are maintained. These constraints play a key role in database design and maintenance.

**LAB 8**

**Title :- implementation of database backup and recovery ,**

**rollback , commit, savepoint.**

**1. Write a query implement the save point**

**2. Write a query to implement the rollback**

**3. Write a query to implement the commit .**

**Theory:**In relational database management systems (RDBMS) like MySQL, transactions are used to manage changes to data. Transactions are essential for maintaining data integrity, consistency, and reliability. The operations within a transaction are atomic, meaning they either complete fully or not at all, which ensures that the database remains in a consistent state, even in case of failures.**Key Concepts of Transaction Control in SQL:** Commit: A commit operation saves all the changes made during the current transaction to the database permanently. Once a commit is executed, it is impossible to roll back those changes. Rollback: A rollback operation undoes any changes made by the current transaction since the last commit or savepoint. If no savepoint is used, the rollback undoes the entire transaction. Savepoint: A savepoint is used to set a point within a transaction where you can later roll back to. It allows partial rollback of a transaction, which is useful when you want to discard part of the transaction but keep the rest.**Transaction Control Commands in SQL:**

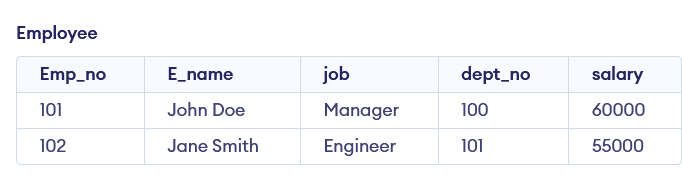
**START TRANSACTION:**

This command begins a new transaction in SQL. It marks the starting point of a series of operations that will be treated as a single unit.START TRANSACTION;**COMMIT:**The COMMIT command finalizes the changes made during the transaction and saves them to the database permanently. Once a commit is performed, the changes cannot be undone. After a commit, the transaction is considered complete.COMMIT;**ROLLBACK:**The ROLLBACK command undoes all changes made during the current transaction. If a SAVEPOINT is used, the rollback can be to that specific savepoint, discarding changes made after it.ROLLBACK;**SAVEPOINT:** A SAVEPOINT creates a point within a transaction to which you can roll back later. It does not end the transaction, and all changes before the savepoint are still active. This allows for more granular control over which parts of a transaction should be retained or discarded. SAVEPOINT savepoint\_name;**How these concepts work together:Starting a Transaction:**A transaction is initiated with START TRANSACTION, which tells the database that the following operations should be treated as a single unit.**Using Savepoints**:

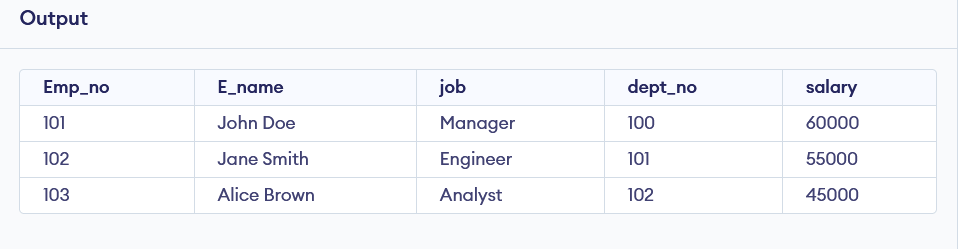
Inside a transaction, a savepoint can be defined using the SAVEPOINT command. This creates a point to which the transaction can be rolled back if needed, without undoing the entire transaction.**Rolling Back:**If an error occurs or if a certain condition is met, the ROLLBACK command can be used. If a savepoint exists, the transaction can be rolled back to that specific point, keeping the earlier changes intact.**Committing Changes:**Once the transaction is completed and no further changes are required, the COMMIT command is issued. This makes all changes permanent in the database and ends the transaction.**Why Transactions and Control Statements Are Important:Data Integrity**:Transactions help maintain ACID properties (Atomicity, Consistency, Isolation, Durability), which ensures the database is always in a consistent state.Changes within a transaction are atomic; they either fully happen or not at all.**Error Recovery:**Rollback provides a way to recover from errors by undoing changes made during the current transaction.Savepoints allow for partial rollbacks without affecting the entire transaction, providing more flexibility in managing operations. **Concurrency Control:**Transactions help manage concurrency in a multi-user environment, ensuring that conflicting operations do not cause data corruption.**Consistency in Operations:**The use of commit ensures that once changes are made, they are permanent, providing consistency across the system.

**OUTPUT**

****

****





#### Conclusion:

In this lab, we explored the fundamental concepts of transaction control in SQL by focusing on the use of the SAVEPOINT, ROLLBACK, and COMMIT commands. These commands are essential tools for managing and ensuring the integrity of data in a relational database management system (RDBMS), especially when multiple modifications are made to a database within a transaction.

The **SAVEPOINT** command allows us to create a point within a transaction, enabling a partial rollback to that point without affecting the entire transaction. This is especially useful in complex transactions where, instead of rolling back all changes, only certain portions can be undone, preserving the integrity of the database while correcting or retrying specific operations.

The **ROLLBACK** command is crucial for undoing changes made during a transaction. It ensures that if an error occurs or if the modifications are not desired, all changes made since the start of the transaction (or since the last savepoint) can be discarded. This helps in maintaining the consistency and correctness of the database by preventing the application of partial or faulty data. It serves as a safeguard against unexpected failures, allowing the database to revert to a stable state.

The **COMMIT** command is used to finalize all changes made during a transaction. Once a COMMIT is issued, the changes become permanent and visible to all other users of the database. This operation ensures that all modifications are successfully saved to the database and guarantees the consistency of the database state. By using COMMIT, developers can ensure that a group of changes are executed atomically—either all changes succeed, or none are applied.

Together, these commands provide a powerful mechanism for controlling the flow of data in an RDBMS. They allow developers to implement error handling, manage complex data updates, and guarantee that the database remains in a valid state even in the case of unexpected failures. Through effective use of transactions, the potential for data inconsistencies is minimized, and the robustness of the system is enhanced.

Furthermore, these commands are indispensable when working with mission-critical applications where data integrity and consistency are paramount. For example, in scenarios where multiple updates or inserts are performed simultaneously, ensuring atomicity and rollback capabilities guarantees that the database remains in a consistent and reliable state. Additionally, the ability to manage transactions allows for efficient error recovery and fine-grained control over database changes.

In summary, the use of SAVEPOINT, ROLLBACK, and COMMIT commands is a core component of transaction management in SQL, helping to ensure the safety, integrity, and consistency of the data in relational databases. By mastering these transaction control mechanisms, developers can handle complex operations with confidence and provide better error recovery, ensuring that the database's state is always consistent and reliable.