EXPERIMENT NO. 1

Title: Installation of Oracle / MySQL Database Management System

Objective: To install and configure Oracle or MySQL DBMS software and verify successful installation.

Software Required:

- Oracle Database / MySQL Community Server
- MySQL Workbench (if using MySQL)
- Operating System: Windows / Linux

Theory:

A Database Management System (DBMS) is software used to store, retrieve, and manage data in a structured manner.
Oracle and MySQL are popular relational DBMSs that support SQL for database operations.
Before creating databases and tables, DBMS software must be installed and configured properly.

Procedure:

A. Installing MySQL

1. Download *MySQL Community Server* from the official MySQL website.
2. Run the installer (.msi file) and choose **Developer Default** setup.
3. Install required components:
    - MySQL Server
    - MySQL Workbench
4. Configure MySQL Server:
    - Select *Standalone Server*
    - Set port number (default: **3306**)
    - Create **root** password
5. Complete installation and launch MySQL Workbench.
6. Connect using root account to verify installation.

B. Installing Oracle

1. Download *Oracle Database Express Edition (XE)*.
2. Run the installer and accept the license agreement.
3. Create an administrator password during setup.
4. Complete installation and open **SQL Plus** or **Oracle SQL Developer**.
5. Connect using username: **system** and the created password.
6. Verify the installation by running a simple SQL query:
7. SELECT * FROM dual;

EXPERIMENT NO. 2

Title: Creating Entity–Relationship (ER) Diagram Using CASE Tools

Objective:

To design an Entity–Relationship Diagram (ERD) using a CASE tool such as MySQL Workbench, Oracle SQL Developer Data Modeler, or Draw.io.

Software Required:

- MySQL Workbench / Oracle SQL Data Modeler / Draw.io
- Computer with Windows/Linux OS

Theory:

An **Entity–Relationship Diagram (ERD)** is a graphical representation of entities, attributes, and relationships among data items in a database.
It helps in understanding the logical structure of a database before implementation.

Key Terms:

- **Entity:** Object or concept (e.g., Student, Course).
- **Attribute:** Properties of an entity (e.g., Roll_No, Name).
- **Primary Key:** Unique identifier for an entity.
- **Relationship:** Association between entities (e.g., Student *enrolls* in Course).
- **Cardinality:** Defines the relationship type (1:1, 1:M, M:N).

Procedure:

Using MySQL Workbench

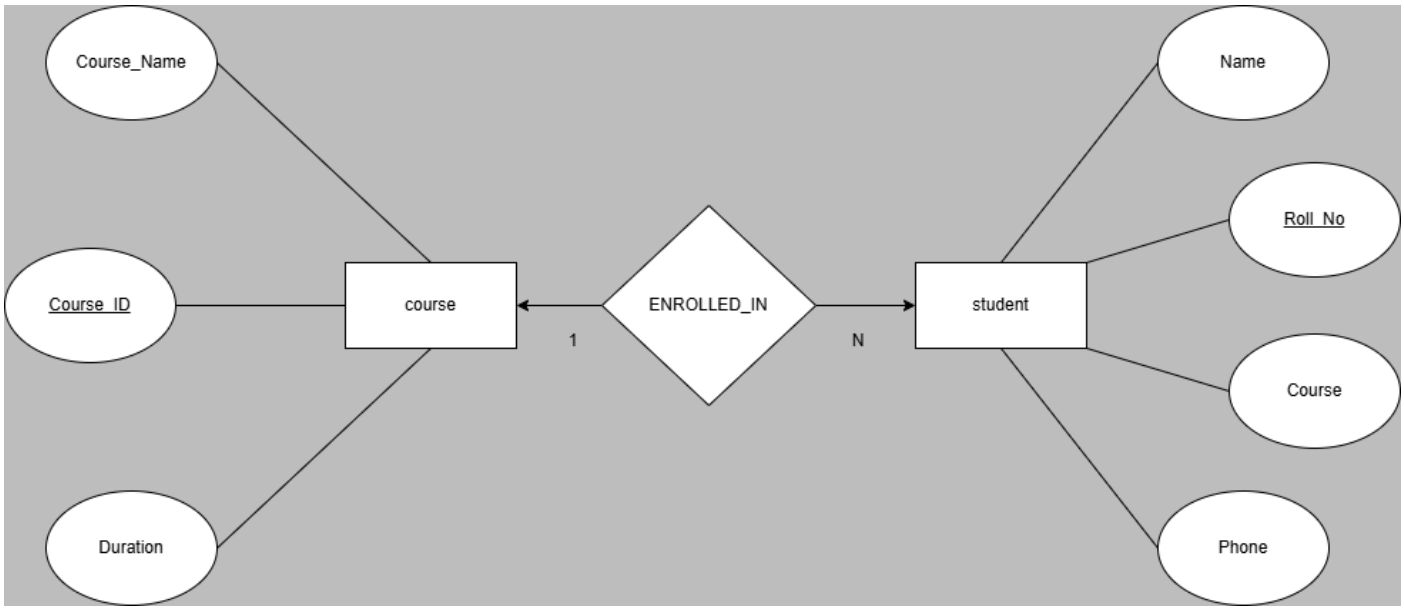
1. Open **MySQL Workbench**.
2. Go to **File → New Model**.
3. Click **Add Diagram** under EER Diagrams.
4. Drag **Tables** onto the canvas to represent entities.
5. Add **columns** for attributes and mark primary keys (PK).
6. Use the **relationship tool** to connect entities:
    - 1:1
    - 1:N
    - M:N (via a bridge table)
7. Arrange and label the diagram for clarity.
8. Save the model as .mwb file and export the diagram as an image/PDF.

Using Oracle SQL Developer Data Modeler

1. Open the Data Modeler tool.

2. Create a **New Relational Model**.
3. Add **Entities** and assign attributes.
4. Define **Primary Keys** and **Foreign Keys**.
5. Draw relationships between entities.
6. Save and export the diagram.

Sample ER Diagram Example:



STUDENT (Roll_No, Name, Course, Phone)
COURSE (Course_ID, Course_Name, Duration)
Relationship: Student **enrolled in** Course (1:N)

EXPERIMENT NO. 3

Title: Writing SQL Statements Using Oracle / MySQL

Objective: To understand and execute various SQL queries such as basic SELECT statements, restricting and sorting data, joining multiple tables, using group functions, and manipulating data.

Software Required:

- Oracle SQL Plus / Oracle SQL Developer
- MySQL Server / MySQL Workbench

A) BASIC SQL SELECT STATEMENTS

Theory:

The **SELECT** statement is used to retrieve data from a table.

Syntax:

SELECT column1, column2 FROM table_name;

Example Queries:

Queries: SELECT * FROM employees;

Output:

id	name	salary	dept_id
101	Rahul	35000	1
102	Priya	45000	2
103	Aman	55000	1

Queries: SELECT name, salary FROM employees;

Output:

name	salary
Rahul	35000
Priya	45000
Aman	55000

B) RESTRICTING AND SORTING DATA

Restricting Data using WHERE clause:

SELECT * FROM employees

WHERE salary > 30000;

id	name	salary	dept_id
101	Rahul	35000	1
102	Priya	45000	2
103	Aman	55000	1

Sorting Data using ORDER BY:

SELECT name, salary FROM employees

ORDER BY salary DESC;

name	salary
Aman	55000
Priya	45000
Rahul	35000

C) DISPLAYING DATA FROM MULTIPLE TABLES

Using JOINs

INNER JOIN:

SELECT employees.name, departments.dept_name

FROM employees

INNER JOIN departments

ON employees.dept_id = departments.dept_id;

name	dept_name
Rahul	HR
Priya	IT
Aman	HR

LEFT JOIN:

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

name	dept_name
Rahul	HR
Priya	IT
Aman	HR

CROSS JOIN:

```
SELECT * FROM employees CROSS JOIN departments;
```

id	name	salary	dept_id	dept_id	dept_name
101	Rahul	35000	1	1	HR
101	Rahul	35000	1	2	IT
102	Priya	45000	2	1	HR
102	Priya	45000	2	2	IT
103	Aman	55000	1	1	HR
103	Aman	55000	1	2	IT

D) AGGREGATING DATA USING GROUP FUNCTIONS

Common Group Functions:

- COUNT()
- SUM()
- AVG()
- MIN()
- MAX()

Examples:

SELECT COUNT(*) FROM employees;

COUNT(*)
3

SELECT AVG(salary) FROM employees;

AVG(salary)
45000

SELECT dept_id, SUM(salary)

FROM employees

GROUP BY dept_id;

dept_id	SUM(salary)
1	90000
2	45000

Using HAVING clause:

SELECT dept_id, AVG(salary)

FROM employees

GROUP BY dept_id

HAVING AVG(salary) > 30000;

dept_id	AVG(salary)
1	45000
2	45000

E) MANIPULATING DATA

1. INSERT data

INSERT INTO employees (id, name, salary, dept_id)

VALUES (101, 'Rahul', 35000, 1);

id	name	salary	dept_id
101	Rahul	35000	1

2. UPDATE data

UPDATE employees

SET salary = 40000

WHERE id = 101;

id	name	salary	dept_id
101	Rahul	40000	1

3. DELETE data

DELETE FROM employees

WHERE id = 101;

id	name	salary	dept_id
(No rows)			

EXPERIMENT NO. 4

Title: Creating and Managing Tables in Oracle / MySQL

Objective: To learn how to create, modify, describe, and delete tables in a database using SQL statements.

Software Required:

- Oracle SQL Developer / SQL Plus
- MySQL Server / MySQL Workbench

Theory:

A **table** is the basic unit of data storage in a relational database.
Creating and managing tables involves:

- Creating tables
- Viewing table structure
- Adding, modifying, or deleting columns
- Renaming tables
- Dropping tables

Procedure & SQL Commands

1. Creating a Table

Syntax:

```
CREATE TABLE table_name (
   column1 datatype constraints,
   column2 datatype constraints,
   ...
);
```

Example:

```
CREATE TABLE students (
   roll_no INT PRIMARY KEY,
   name VARCHAR(50),
   age INT,
   course VARCHAR(50)
);
```

```
Table created.
```

---

2. Viewing Table Structure

Oracle:

DESC students;

MySQL:

DESCRIBE students;

Field	Type	Null	Key	Default	Extra
roll_no	INT	NO	PRI	NULL	
name	VARCHAR(50)	YES		NULL	
age	INT	YES		NULL	
course	VARCHAR(50)	YES		NULL	

---

3. Adding a New Column

ALTER TABLE students

ADD email VARCHAR(50);

Field	Type	Null	Key	Default	Extra
roll_no	INT	NO	PRI	NULL	
name	VARCHAR(50)	YES		NULL	
age	INT	YES		NULL	
course	VARCHAR(50)	YES		NULL	
email	VARCHAR(50)	YES		NULL	

---

4. Modifying an Existing Column

ALTER TABLE students

MODIFY age INT NOT NULL;

Field	Type	Null	Key	Default	Extra
roll_no	INT	NO	PRI	NULL	
name	VARCHAR(50)	YES		NULL	
age	INT	NO		NULL	
course	VARCHAR(50)	YES		NULL	
email	VARCHAR(50)	YES		NULL	

5. Renaming a Column

ALTER TABLE students

RENAME COLUMN name TO student_name;

Field	Type	Null	Key	Default	Extra
roll_no	INT	NO	PRI	NULL	
student_name	VARCHAR(50)	YES		NULL	
age	INT	NO		NULL	
course	VARCHAR(50)	YES		NULL	
email	VARCHAR(50)	YES		NULL	

---

6. Renaming a Table

RENAME TABLE students TO student_details;

```
mysql> show tables;
+------------------+
| Tables_in_demo   |
+------------------+
| students         |
+------------------+
1 row in set (0.11 sec)
```

```
mysql> show tables;
+------------------+
| Tables_in_demo   |
+------------------+
| student_details  |
+------------------+
1 row in set (0.07 sec)
```

---

7. Dropping a Column

ALTER TABLE student_details

DROP COLUMN email;

Field	Type	Null	Key	Default	Extra
roll_no	INT	NO	PRI	NULL	
student_name	VARCHAR(50)	YES		NULL	
age	INT	NO		NULL	
course	VARCHAR(50)	YES		NULL	

---

8. Dropping a Table

DROP TABLE student_details;

```
mysql> show tables;
+------------------+
| Tables_in_demo   |
+------------------+
|                  |
+------------------+
Empty set (0.01 sec)
```

EXPERIMENT NO. 5

Title: Normalization

Objective: To understand and apply different normalization forms (1NF, 2NF, 3NF, BCNF) to remove data redundancy and improve database design.

Theory:

Normalization is a process of organizing data in a database to reduce redundancy and improve data integrity.

Why Normalize?

- Removes duplicate data
- Ensures data consistency
- Makes database efficient
- Avoids insertion, deletion & update anomalies

Types of Normal Forms

1. First Normal Form (1NF)

A table is in **1NF** if:

- All values are **atomic** (no multiple values in a single cell)
- No repeating groups

Example (Unnormalized):

Student	Subjects
A	Math, English

1NF Table:

Student	Subject
A	Math
A	English

---

2. Second Normal Form (2NF)

A table is in **2NF** if:

- It is already in **1NF**
- No **partial dependency**
  (A non-key attribute must depend on the **whole** primary key)

Example:

Unnormalized Table:

RollNo Subject StudentName

StudentName depends only on RollNo, not on (RollNo, Subject).

Student	Subjects
A	Math, English

After applying 1NF

(Every cell contains only atomic values)

Student	Subject
A	Math
A	English

2NF Tables:

Student Table
| RollNo | StudentName |

Subject Table
| RollNo | Subject |

RollNo	Subject	StudentName

After applying 2NF

Student Table

RollNo	StudentName

Subject Table

RollNo	Subject

---

3. Third Normal Form (3NF)

A table is in **3NF** if:

- It is already in **2NF**
- No **transitive dependency**
  (Non-key attribute should not depend on another non-key attribute)

Example:

| RollNo | Name | City | Pincode |
Here **City → Pincode** (transitive dependency)

3NF Tables:

Student
| RollNo | Name | City |

City
| City | Pincode |

RollNo	Name	City	Pincode

After applying 3NF

Student Table

RollNo	Name	City

City Table

City	Pincode

---

4. Boyce–Codd Normal Form (BCNF)

A stronger version of 3NF.
A table is in **BCNF** if:

- For every functional dependency A → B,
  A must be a super key.

Course	Instructor	Room

Course Table
| Course | Instructor |

Room Table
| Instructor | Room |

EXPERIMENT NO. 6

Title: Creating Cursor in PL/SQL / MySQL

Objective: To understand how to create, open, fetch, and close a cursor for row-by-row processing in a database.

Theory:

A **cursor** is a database pointer that allows you to process query results **one row at a time**.

Types of Cursors:

1. **Implicit Cursor** – Automatically created by Oracle/MySQL for simple queries (INSERT, UPDATE, DELETE, SELECT INTO).
2. **Explicit Cursor** – Declared by the programmer to manually control fetching of rows.

Cursor Steps:

1. **Declare** the cursor
2. **Open** the cursor
3. **Fetch** rows from the cursor
4. **Close** the cursor

PL/SQL Example (Oracle)

```
DECLARE
  CURSOR emp_cur IS
    SELECT empno, ename, salary FROM employees;
  v_empno employees.empno%TYPE;
  v_ename employees.ename%TYPE;
  v_salary employees.salary%TYPE;

BEGIN
  OPEN emp_cur;
  LOOP
    FETCH emp_cur INTO v_empno, v_ename, v_salary;
    EXIT WHEN emp_cur%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('Emp No: ' || v_empno ||
                         ' Name: ' || v_ename ||
                         ' Salary: ' || v_salary);
```

```
    END LOOP;
    CLOSE emp_cur;
END;
/
```

Output:

```
Emp No: 101 Name: Rahul Salary: 35000
Emp No: 102 Name: Priya Salary: 45000
Emp No: 103 Name: Aman Salary: 55000

PL/SQL procedure successfully completed.
```

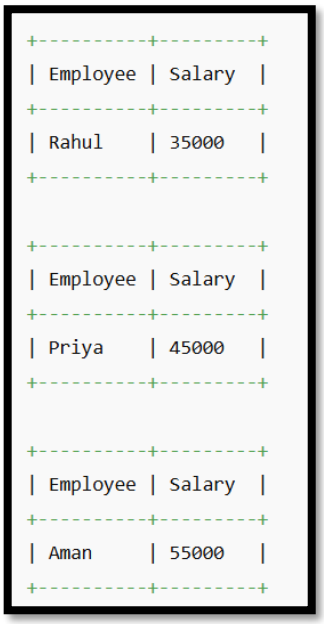
MySQL Cursor Example (Stored Procedure)

```
DELIMITER $$
CREATE PROCEDURE displayEmployees()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_name VARCHAR(50);
    DECLARE v_salary INT;
    DECLARE emp_cur CURSOR FOR
        SELECT name, salary FROM employees;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN emp_cur;
    read_loop: LOOP
        FETCH emp_cur INTO v_name, v_salary;
        IF done = 1 THEN
            LEAVE read_loop;
        END IF;
        SELECT v_name AS Employee, v_salary AS Salary;
    END LOOP;
    CLOSE emp_cur;
END $$
DELIMITER ;
```

To run:

CALL displayEmployees();

Output:



```
+----------+---------+
| Employee | Salary  |
+----------+---------+
| Rahul    | 35000   |
+----------+---------+

+----------+---------+
| Employee | Salary  |
+----------+---------+
| Priya    | 45000   |
+----------+---------+

+----------+---------+
| Employee | Salary  |
+----------+---------+
| Aman     | 55000   |
+----------+---------+
```

EXPERIMENT NO. 7

Title: Creating Procedures and Functions

Objective: To understand how to create, execute, and manage **stored procedures** and **functions** in Oracle / MySQL.

Theory:

Stored Procedure:

A stored procedure is a precompiled SQL program that performs a task such as inserting, updating, or retrieving data.

Function:

A function is similar to a procedure but **must return a value**.

A) CREATING PROCEDURE

(1) Procedure in Oracle (PL/SQL)

```
CREATE OR REPLACE PROCEDURE show_employee (
   p_empno IN employees.empno%TYPE
)
IS
   v_name employees.ename%TYPE;
   v_salary employees.salary%TYPE;
BEGIN
   SELECT ename, salary
   INTO v_name, v_salary
   FROM employees
   WHERE empno = p_empno;

   DBMS_OUTPUT.PUT_LINE('Name: ' || v_name || ' Salary: ' || v_salary);
END;
/
```

Executing Procedure

```
EXEC show_employee(101);
```

Output:

```
Name: Rahul Salary: 35000
PL/SQL procedure successfully completed.
```

(2) Procedure in MySQL

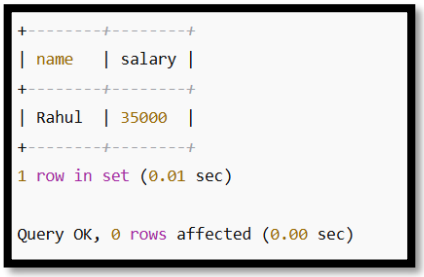
```
DELIMITER $$

CREATE PROCEDURE showEmployee(IN emp_id INT)
BEGIN
   SELECT name, salary
   FROM employees
   WHERE id = emp_id;
END $$

DELIMITER ;
```

Calling the Procedure

```
CALL showEmployee(101);
```



```
+--------+--------+
| name   | salary |
+--------+--------+
| Rahul  | 35000  |
+--------+--------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.00 sec)
```

B) CREATING FUNCTION

(1) Function in Oracle

```
CREATE OR REPLACE FUNCTION get_salary (
   p_empno IN employees.empno%TYPE
)
RETURN NUMBER
IS
   v_salary employees.salary%TYPE;
BEGIN
   SELECT salary INTO v_salary
   FROM employees
   WHERE empno = p_empno;
```

```
    RETURN v_salary;
END;
/
```

Executing Function

```
SELECT get_salary(101) FROM dual;
```

Output:

```
GET_SALARY(101)
---------------
          35000
```

---

(2) Function in MySQL

```
DELIMITER $$

CREATE FUNCTION getSalary(emp_id INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE v_salary INT;
    SELECT salary INTO v_salary
    FROM employees
    WHERE id = emp_id;

    RETURN v_salary;
END $$
DELIMITER ;
```

Calling Function

```
SELECT getSalary(101);
```

Output:

```
+-----------------+
| getSalary(101)  |
+-----------------+
|        35000    |
+-----------------+
1 row in set (0.00 sec)
```

EXPERIMENT NO. 8

Title: Creating Packages and Triggers

Objective: To understand how to create and use **packages** and **triggers** in Oracle / MySQL.

PART A — PACKAGES (Oracle)

Theory:

A **Package** is a collection of related procedures, functions, variables, and cursors stored together in the database.

A package has two parts:

1. **Package Specification (Interface)** – Declares procedures/functions
2. **Package Body** – Contains actual code (definitions)

Advantages of Packages

- Better organization of code
- Encapsulation
- Improved performance
- Reusability

---

1. Creating Package Specification

```
CREATE OR REPLACE PACKAGE emp_package AS
   PROCEDURE getEmployee(p_empno IN NUMBER);
   FUNCTION getSalary(p_empno IN NUMBER) RETURN NUMBER;
END emp_package;
/
```

---

2. Creating Package Body

```
CREATE OR REPLACE PACKAGE BODY emp_package AS

   PROCEDURE getEmployee(p_empno IN NUMBER) IS
      v_name employees.ename%TYPE;
   BEGIN
      SELECT ename INTO v_name
      FROM employees
      WHERE empno = p_empno;
```

```
    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_name);
  END getEmployee;

  FUNCTION getSalary(p_empno IN NUMBER)
    RETURN NUMBER IS
    v_salary employees.salary%TYPE;
  BEGIN
    SELECT salary INTO v_salary
    FROM employees
    WHERE empno = p_empno;

    RETURN v_salary;
  END getSalary;

END emp_package;
/
```

---

3. Executing Package Elements

EXEC emp_package.getEmployee(101);

```
Employee Name: Rahul
PL/SQL procedure successfully completed.
```

SELECT emp_package.getSalary(101) AS Salary FROM dual;

```
    SALARY
------------
       35000
```

PART B — TRIGGERS (Oracle / MySQL)

Theory:

A **Trigger** is a stored PL/SQL block that executes automatically in response to an event:

- INSERT
- UPDATE

- DELETE

Uses of Triggers

- Enforcing business rules
- Auditing database changes
- Maintaining logs
- Preventing invalid transactions

---

1. BEFORE INSERT Trigger (Oracle / MySQL)

Example: Automatically set joining date

```
CREATE OR REPLACE TRIGGER set_joining_date
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
   :NEW.joining_date := SYSDATE;
END;
/
```

Query (Insert):

```
INSERT INTO employees (empno, ename, salary)
VALUES (104, 'Neha', 30000);
```

Table:

empno	ename	salary	joining_date
101	Rahul	35000	01-JAN-2025
102	Priya	45000	05-JAN-2025
103	Aman	55000	10-JAN-2025

After:

empno	ename	salary	joining_date
101	Rahul	35000	01-JAN-2025
102	Priya	45000	05-JAN-2025
103	Aman	55000	10-JAN-2025
104	Neha	30000	**SYSDATE** (auto)

2. AFTER INSERT Trigger (Audit Log)

```
CREATE OR REPLACE TRIGGER emp_audit
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
   INSERT INTO employees_log(empno, action_date, action_type)
   VALUES (:NEW.empno, SYSDATE, 'INSERT');
END;
/
```

empno	action_date	action_type
104	12-JAN-2025	INSERT

---

3. UPDATE Trigger Example

```
CREATE OR REPLACE TRIGGER salary_update_log
AFTER UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
   INSERT INTO salary_log(empno, old_salary, new_salary, update_date)
   VALUES (:OLD.empno, :OLD.salary, :NEW.salary, SYSDATE);
END;
/
```

Query (Update):

```
UPDATE employees
SET salary = 38000
WHERE empno = 101;
```

Output:

Employees Table After Update

empno	ename	salary
101	Rahul	**38000**
102	Priya	45000
103	Aman	55000
104	Neha	30000

salary_log (Trigger Output)

empno	old_salary	new_salary	update_date
101	35000	38000	12-JAN-2025

---

4. DELETE Trigger Example

```
CREATE OR REPLACE TRIGGER emp_delete_log
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
   INSERT INTO delete_log(empno, deleted_on)
   VALUES (:OLD.empno, SYSDATE);
END;
/
```

Query (Delete):

```
DELETE FROM employees WHERE empno = 103;
```

Output:

Employees Table After Delete

empno	ename	salary
101	Rahul	38000
102	Priya	45000
104	Neha	30000

delete_log (Trigger Output)

empno	deleted_on
103	12-JAN-2025

EXPERIMENT NO. 9

Title: Design and Implementation of Payroll Processing System

Objective: To design the database structure and implement SQL queries for a **Payroll Management System** that calculates employee salary, allowances, deductions, and net pay.

Theory:

A **Payroll Processing System** automates salary computation for employees based on:

- Basic salary
- Allowances (DA, HRA, TA, etc.)
- Deductions (PF, Tax, Insurance, etc.)
- Net Salary = Earnings – Deductions

This experiment involves:

1. Designing ER Diagram
2. Creating tables
3. Inserting sample data
4. Writing queries to generate payroll reports

---

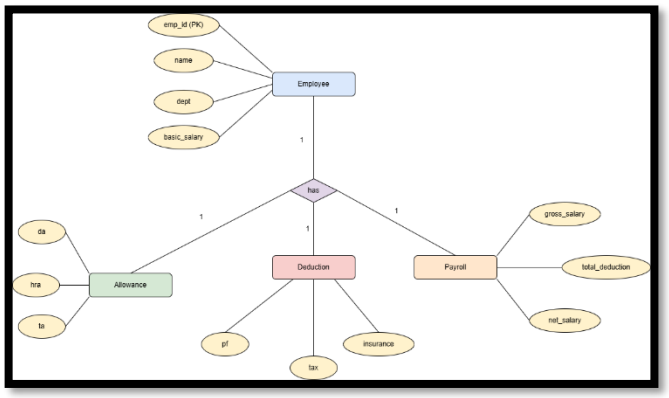
1. ER DIAGRAM

Entities:

- **Employee(emp_id, name, dept, basic_salary)**
- **Allowance(emp_id, da, hra, ta)**
- **Deduction(emp_id, pf, tax, insurance)**
- **Payroll(emp_id, gross_salary, total_deduction, net_salary)**

Relationships:

Employee **1 : 1** Allowance
Employee **1 : 1** Deduction
Employee **1 : 1** Payroll



2. TABLE CREATION

Employee Table

```
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    dept VARCHAR(30),
    basic_salary INT
);
```

Allowance Table

```
CREATE TABLE allowance (
    emp_id INT,
    da INT,
    hra INT,
    ta INT,
    FOREIGN KEY (emp_id) REFERENCES employee(emp_id)
);
```

Deduction Table

```
CREATE TABLE deduction (
    emp_id INT,
    pf INT,
    tax INT,
    insurance INT,
    FOREIGN KEY (emp_id) REFERENCES employee(emp_id)
);
```

Payroll Table

```
CREATE TABLE payroll (
    emp_id INT PRIMARY KEY,
    gross_salary INT,
    total_deduction INT,
    net_salary INT,
    FOREIGN KEY (emp_id) REFERENCES employee(emp_id)
);
```

3. INSERTING SAMPLE DATA

INSERT INTO employee VALUES

(101, 'Rahul', 'HR', 30000),

(102, 'Priya', 'IT', 45000);

INSERT INTO allowance VALUES

(101, 5000, 3000, 2000),

(102, 7000, 5000, 2500);

INSERT INTO deduction VALUES

(101, 2000, 1500, 500),

(102, 3000, 2500, 700);

Output:

emp_id	name	dept	basic_salary
101	Rahul	HR	30000
102	Priya	IT	45000

5. CALCULATING PAYROLL

ALLOWANCE TABLE (INSERT + SELECT)

```
mysql> INSERT INTO allowance VALUES
    -> (101, 5000, 3000, 2000),
    -> (102, 7000, 5000, 2500);
Query OK, 2 rows affected (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Select Query

mysql> SELECT * FROM allowance;

```
+--------+------+-------+------+
| emp_id | da   | hra   | ta   |
+--------+------+-------+------+
|    101 | 5000 |  3000 | 2000 |
|    102 | 7000 |  5000 | 2500 |
+--------+------+-------+------+
2 rows in set (0.01 sec)
```

4. CALCULATING PAYROLL

Gross Salary = Basic + DA + HRA + TA

Total Deduction = PF + TAX + Insurance

Net Salary = Gross Salary – Total Deduction

```
INSERT INTO payroll (emp_id, gross_salary, total_deduction, net_salary)
SELECT
    e.emp_id,
    (e.basic_salary + a.da + a.hra + a.ta) AS gross_salary,
    (d.pf + d.tax + d.insurance) AS total_deduction,
    ((e.basic_salary + a.da + a.hra + a.ta) -
     (d.pf + d.tax + d.insurance)) AS net_salary
FROM employee e
JOIN allowance a ON e.emp_id = a.emp_id
JOIN deduction d ON e.emp_id = d.emp_id;
```

Query

```
SELECT * FROM payroll;
```

Output:

emp_id	gross_salary	total_deduction	net_salary
101	40000	4000	36000
102	59500	6200	53300

---

5. DISPLAY PAYROLL REPORT

```
SELECT e.emp_id, e.name, e.dept,
      p.gross_salary, p.total_deduction, p.net_salary
FROM employee e
JOIN payroll p ON e.emp_id = p.emp_id;
```

Output:

Emp_ID	Name	Dept	Gross Salary	Deduction	Net Salary
101	Rahul	HR	40000	4000	36000
102	Priya	IT	59500	6200	53300

EXPERIMENT NO. 10

Title: Design and Implementation of Library Information System

Objective: To design the database structure and implement SQL queries for a **Library Information System (LIS)** to manage books, members, issue/return records, and fines.

Theory:

A **Library Information System** helps in managing:

- Books and authors
- Library members
- Issue and return of books
- Fine calculation
- Availability status

The system ensures efficient tracking and reduces manual errors.

---

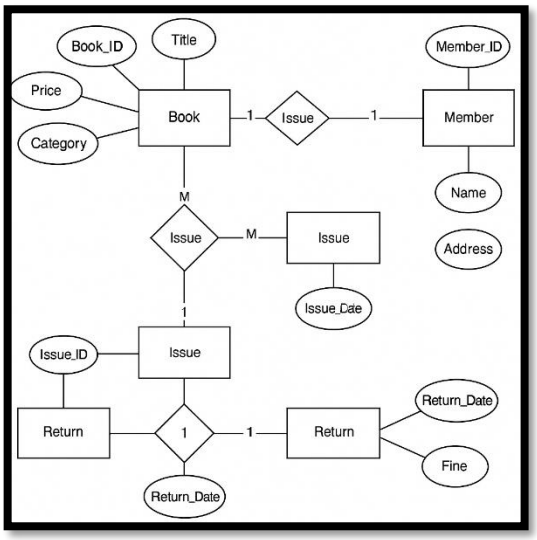
1. ER DIAGRAM

Entities:

1. **Book(Book_ID, Title, Author, Category, Price)**
2. **Member(Member_ID, Name, Address, Phone)**
3. **Issue(Issue_ID, Book_ID, Member_ID, Issue_Date, Due_Date)**
4. **Return(Return_ID, Issue_ID, Return_Date, Fine)**

Relationships:

- Book **1 : M** Issue
- Member **1 : M** Issue
- Issue **1 : 1** Return



2. TABLE CREATION

Book Table

```
CREATE TABLE book (
    book_id INT PRIMARY KEY,
    title VARCHAR(100),
    author VARCHAR(50),
    category VARCHAR(30),
    price INT
);
```

Member Table

```
CREATE TABLE member (
    member_id INT PRIMARY KEY,
    name VARCHAR(50),
    address VARCHAR(100),
    phone VARCHAR(15)
);
```

Issue Table

```
CREATE TABLE issue (
    issue_id INT PRIMARY KEY,
    book_id INT,
    member_id INT,
    issue_date DATE,
    due_date DATE,
    FOREIGN KEY (book_id) REFERENCES book(book_id),
    FOREIGN KEY (member_id) REFERENCES member(member_id)
);
```

Return Table

```
CREATE TABLE return_book (
    return_id INT PRIMARY KEY,
    issue_id INT,
    return_date DATE,
    fine INT,
    FOREIGN KEY (issue_id) REFERENCES issue(issue_id)
```

);

---

3. INSERTING SAMPLE DATA

Books

INSERT INTO book VALUES

(1, 'DBMS Concepts', 'Korth', 'Education', 500),

(2, 'Operating Systems', 'Galvin', 'Education', 650),

(3, 'Harry Potter', 'J.K. Rowling', 'Fiction', 400);

Members

INSERT INTO member VALUES

(101, 'Rahul', 'Delhi', '9876543210'),

(102, 'Priya', 'Mumbai', '9988776655');

Issue Records

INSERT INTO issue VALUES

(1001, 1, 101, '2025-01-01', '2025-01-10'),

(1002, 3, 102, '2025-01-05', '2025-01-12');

Output: Book Table After Insertion

book_id	title	author	category	price
1	DBMS Concepts	Korth	Education	500
2	Operating Systems	Galvin	Education	650
3	Harry Potter	J.K. Rowling	Fiction	400

Output: Member Table After Insertion

member_id	name	address	phone
101	Rahul	Delhi	9876543210
102	Priya	Mumbai	9988776655

Output: Issue Table After Insertion

issue_id	book_id	member_id	issue_date	due_date
1001	1	101	2025-01-01	2025-01-10
1002	3	102	2025-01-05	2025-01-12

---

4. IMPLEMENTING RETURN & FINE CALCULATION

Assume:
Fine = ₹10 per late day

Example Return Entry

```
INSERT INTO return_book VALUES
(501, 1001, '2025-01-12', NULL);
```

Calculate Fine

```
UPDATE return_book r
JOIN issue i ON r.issue_id = i.issue_id
SET r.fine =
   CASE
      WHEN DATEDIFF(r.return_date, i.due_date) > 0
      THEN DATEDIFF(r.return_date, i.due_date) * 10
      ELSE 0
   END
WHERE r.issue_id = 1001;
```

Return Table After Fine Calculation

```
SELECT * FROM return_book;
```

Output:

return_id	issue_id	return_date	fine
501	1001	2025-01-12	20

5. DISPLAY LIBRARY ISSUE–RETURN REPORT

```
SELECT
   m.name AS Member,
   b.title AS Book,
   i.issue_date,
   i.due_date,
   r.return_date,
   r.fine
FROM member m
JOIN issue i ON m.member_id = i.member_id
JOIN book b ON b.book_id = i.book_id
```

LEFT JOIN return_book r ON r.issue_id = i.issue_id;

Output:

Member	Book	Issue Date	Due Date	Return Date	Fine
Rahul	DBMS Concepts	2025-01-01	2025-01-10	2025-01-12	20
Priya	Harry Potter	2025-01-05	2025-01-12	NULL	NULL

EXPERIMENT NO. 11

Title: Design and Implementation of Student Information System

Objective: To design and implement a Student Information System (SIS) using database tables, relationships, and SQL queries to manage student details, courses, marks, and results.

Theory:

A Student Information System maintains:

- Student personal details
- Course details
- Enrollment information
- Marks/grades
- Result generation

The system ensures proper storage, retrieval, and organization of academic data.

---

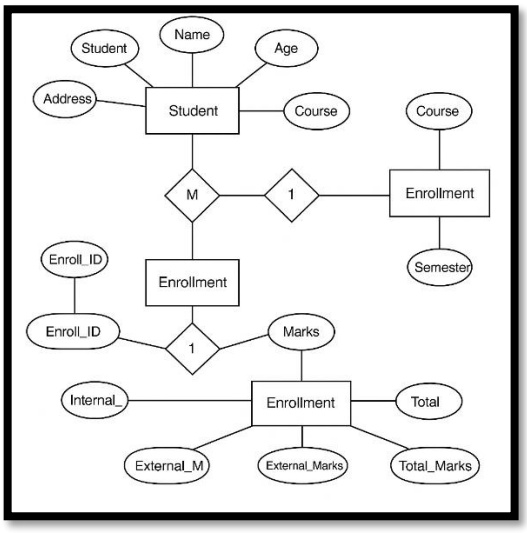
1. ER DIAGRAM

Entities:

1. **Student(Student_ID, Name, Age, Gender, Address)**
2. **Course(Course_ID, Course_Name, Credits)**
3. **Enrollment(Enroll_ID, Student_ID, Course_ID, Semester)**
4. **Marks(Mark_ID, Enroll_ID, Internal_Marks, External_Marks, Total_Marks, Grade)**

Relationships:

- **Student 1 : M Enrollment**
- **Course 1 : M Enrollment**
- **Enrollment 1 : 1 Marks**



2. TABLE CREATION

Student Table

```
CREATE TABLE student (
   student_id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT,
   gender VARCHAR(10),
   address VARCHAR(100)
);
```

Course Table

```
CREATE TABLE course (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(50),
   credits INT
);
```

Enrollment Table

```
CREATE TABLE enrollment (
   enroll_id INT PRIMARY KEY,
   student_id INT,
   course_id INT,
   semester VARCHAR(10),
   FOREIGN KEY (student_id) REFERENCES student(student_id),
   FOREIGN KEY (course_id) REFERENCES course(course_id)
);
```

Marks Table

```
CREATE TABLE marks (
   mark_id INT PRIMARY KEY,
   enroll_id INT,
   internal_marks INT,
   external_marks INT,
   total_marks INT,
   grade VARCHAR(2),
```

FOREIGN KEY (enroll_id) REFERENCES enrollment(enroll_id)

);

---

3. INSERT SAMPLE DATA

Students

INSERT INTO student VALUES

(1, 'Rahul', 20, 'Male', 'Delhi'),

(2, 'Priya', 21, 'Female', 'Mumbai');

Output: Student Table After Insertion

student_id	name	age	gender	address
1	Rahul	20	Male	Delhi
2	Priya	21	Female	Mumbai

Courses

INSERT INTO course VALUES

(101, 'DBMS', 4),

(102, 'Operating Systems', 4);

Output: Course Table After Insertion

course_id	course_name	credits
101	DBMS	4
102	Operating Systems	4

Enrollment

INSERT INTO enrollment VALUES

(5001, 1, 101, 'Sem-3'),

(5002, 2, 102, 'Sem-3');

Output: Enrollment Table After Insertion

enroll_id	student_id	course_id	semester
5001	1	101	Sem-3
5002	2	102	Sem-3

Marks

INSERT INTO marks VALUES

(9001, 5001, 20, 60, NULL, NULL),

(9002, 5002, 18, 65, NULL, NULL);

Output: Marks Table After Insertion

mark_id	enroll_id	internal_marks	external_marks	total_marks	grade
9001	5001	20	60	NULL	NULL
9002	5002	18	65	NULL	NULL

---

4. CALCULATE TOTAL MARKS & GRADE

Total_Marks = Internal + External

Assign Grade

A = ≥ 80
B = ≥ 60
C = ≥ 50
D = < 50

UPDATE marks

SET total_marks = internal_marks + external_marks;

Assign Grade Automatically

UPDATE marks

SET grade =

CASE

WHEN total_marks >= 80 THEN 'A'

WHEN total_marks >= 60 THEN 'B'

WHEN total_marks >= 50 THEN 'C'

ELSE 'D'

END;

Output: Total Marks Calculation

mark_id	enroll_id	internal_marks	external_marks	total_marks	grade
9001	5001	20	60	80	NULL
9002	5002	18	65	83	NULL

Output: Grade Assignment

mark_id	enroll_id	internal_marks	external_marks	total_marks	grade
9001	5001	20	60	80	A
9002	5002	18	65	83	A

---

5. DISPLAY COMPLETE STUDENT REPORT

SELECT s.student_id, s.name, c.course_name,

m.internal_marks, m.external_marks, m.total_marks, m.grade

FROM student s

JOIN enrollment e ON s.student_id = e.student_id

JOIN course c ON c.course_id = e.course_id

JOIN marks m ON m.enroll_id = e.enroll_id;

Output:

Student ID	Name	Course	Internal	External	Total	Grade
1	Rahul	DBMS	20	60	80	A
2	Priya	Operating Systems	18	65	83	A