



# Adi Shankara

## INSTITUTE OF ENGINEERING AND TECHNOLOGY, KALADY

Approved by AICTE & Affiliated to APJ Abdul Kalam Technological University

Vidya Bharathi Nagar, Kalady, Ernakulam, Kerala

[www.adishankara.ac.in](http://www.adishankara.ac.in)

AIL 202

DATABASE MANAGEMENT SYSTEMS LAB

LAB MANUAL STUDENT'S

Name of the Student : Atul Biju

Roll No : ASI23CA023

Semester and Brach of Study : S4 CSE-AI

## ASIET VISION

- To emerge as a Centre of Excellence in Engineering, Technology and Management by imparting quality education, focusing on empowerment and innovation.

## ASIET MISSION

- Impart quality professional education for total upliftment of the society.
- Create congenial academic ambience that kindles innovative thinking and research.
- Mould competent professionals who are socially committed and responsible citizens.

## VISION OF THE DEPARTEMENT

- To be in the frontier of AI technology through quality of education, collaborative research and produce globally competitive, industry ready engineers with social commitment.

## MISSION OF THE DEPARTEMENT

- Achieve excellence in the educational experience, fostering collaborative research through state-of-the-art infrastructure and innovative elements.
- Establish industry collaboration to address interdisciplinary challenges across diverse applications.
- Inspire students to develop into ethical, Innovative and entrepreneurial leaders through a socially-centred program.

## COURSE SYLLABUS

### SYLLABUS

1. Design a database schema for an application with ER diagram from a problem description \*\*.
  2. Creation, modification, configuration, and deletion of databases using UI and SQL Commands \*\*.
  3. Creation of database schema - DDL (create tables, set constraints, enforce relationships, create indices, delete and modify tables). Export ER diagram from the database and verify relationships\*\* (with the ER diagram designed in step 1).
  4. Database initialization - Data insert, Data import to a database (bulk import using UI and SQL Commands)\*\*.
  5. Practice SQL commands for DML (insertion, updating, altering, deletion of data, and viewing/querying records based on condition in databases)\*\*.
  6. Implementation of built-in functions in RDBMS\*\*.
  7. Implementation of various aggregate functions in SQL\*\*.
  8. Implementation of Order By, Group By & Having clause \*\*.
  9. Implementation of set operators nested queries, and join queries \*\*.
  10. Implementation of queries using temp tables.
  11. Practice of SQL TCL commands like Rollback, Commit, Savepoint \*\*.
  12. Practice of SQL DCL commands for granting and revoking user privileges \*\*.
  13. Practice of SQL commands for creation of views and assertions \*\*.
  14. Implementation of various control structures like IF-THEN, IF-THEN-ELSE, IF-THEN-ELSIF, CASE, WHILE using PL/SQL \*\*.
  15. Creation of Procedures, Triggers and Functions\*\*.
  16. Creation of Packages \*\*.
  17. Creation of Cursors \*\*.
  18. Creation of PL/SQL blocks for exception handling \*\*.
  19. Database backup and restore using commands.
  20. Query analysis using Query Plan/Show Plan.
  21. Familiarization of NoSQL Databases and CRUD operations\*\*.
  22. Design a database application using any front end tool for any problem selected. The application constructed should have five or more tables\*\*.
- \*\* mandatory

## Text Books

1. Elmasri R. and S. Navathe, Database Systems: Models, Languages, Design and Application Programming, Pearson Education, 2013.
2. Sliberschatz A., H. F. Korth and S. Sudarshan, Database System Concepts, 6/e, McGraw Hill, 2011.

## References

1. Adam Fowler, NoSQL for Dummies, John Wiley & Sons, 2015
2. NoSQL Data Models: Trends and Challenges (Computer Engineering: Databases and Big Data), Wiley, 2018

## COURSE OUTCOMES

After the completion of this course, students shall be able to:

CO No.	Course Outcome	Knowledge Level
CO1	Design database schema for a given real world problemdomain using standard design and modeling approaches.	Applying (K3)
CO2	Construct queries using SQL for database creation, interaction, modification, and updation	Applying (K3)
CO3	Design and implement triggers and cursors.	Applying (K3)
CO4	Implement procedures, functions, and control structures using PL/SQL.	Applying (K3)
CO5	Perform CRUD operations in NoSQL Databases.	Applying (K3)
CO6	Develop database applications using front-end tools and back-end DBMS.	Create()

CO-PO and CO-PSO mapping :

CO\PO & PSO	PO1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO1 0	PO1 1	PO1 2	PSO 1	PSO 2
1	2	2	2		2			2		2		2		
2	3	2	3		1			2		2		2		
3	3	2	2	2	2			2		1		2		
4	3	3	2	2	1			2		2		2		
5	3	3	1		1			2		2		1		
6	3	3	1	1	2	1		2	1	2	2	1		

1-Slightly, 2-Moderately, 3-Strongly

GENERAL INSTRUCTIONS (for the lab)

1. During the lab hours, upon the completion of an experiment, ask your Staff in charge to verify your results. DO NOT PROCEED to the next experiment until the Staff in charge has verified and taken note of your results. These observations will be used to determine your in-lab performance.
2. Upon completion of an experiment, students will submit their individual lab journal along with their observations(input and output) /inferences to their respective staff in charges and get it signed.
3. Make up for an experiment will be granted for excused absences. In such a case, the experiment must be completed as soon as possible so that it will not interfere with the normal lab schedule. Furthermore, any student who has missed any lab due to an excused absence, is required to complete the experiment individually.
4. If an individual cannot complete the experiment due to equipment malfunction or other unforeseen situation, students are allowed to attend other lab sessions for that week, if space is available. If space cannot be found, then students can complete their work in the following week. However, every effort must be made to complete the experiment during its designated week.
5. Finally, any unexcused absence results in a grade of 0 for the lab. Weekly attendance for this lab is mandatory. Habitually late students (i.e., students late more than 15 minutes more than once) will receive point reductions in their grades for each occurrence.
6. Assessment Method: The Academic Assessment for the Programming lab will be done internally by the College. There will be a final Practical Exam out of 25 marks (internal by the College). The total marks for the lab obtained out of 50 will be converted into equivalent proportion out of 20 for Continuous Internal Evaluation (CIE) computation. Exams are taken individually and not as a team.

## INDEX

1	Design a database schema for an application with ER diagram from a problem description	CO1
2	Creation, modification, configuration, and deletion of databases using UI and SQL Commands **.	CO2
3	Creation of database schema - DDL (create tables, set constraints, enforce relationships, create indices, delete and modify tables). Export ER diagram from the database and verify relationships** (with the ER diagram designed in step 1).	CO2
4	Database initialization - Data insert, Data import to a database (bulk import using UI and SQL Commands)**.	CO2
5	Practice SQL commands for DML (insertion, updating, altering, deletion of data, and viewing/querying records based on condition in databases)**.	CO2
6	Implementation of built-in functions in RDBMS**.	CO2
7	Implementation of various aggregate functions in SQL**.	CO2
8	Implementation of Order By, Group By & Having clause **.	CO2
9	Implementation of set operators nested queries, and join queries **.	CO2
10	Implementation of queries using temp tables.	CO2
11	Practice of SQL TCL commands like Rollback, Commit, Savepoint **.	CO2
12	Practice of SQL DCL commands for granting and revoking user privileges **.	CO2
13	Practice of SQL commands for creation of views and assertions **.	CO2
14	Implementation of various control structures like IF-THEN, IF-THEN-ELSE, IFTHENELSIF, CASE, WHILE using PL/SQL **.	CO4

15	Creation of Procedures, Triggers and Functions**.	CO3, CO4
16	Creation of Packages **.	CO4
17	Creation of Cursors **.	CO3
18	Creation of PL/SQL blocks for exception handling **.	CO4
19	Database backup and restore using commands	CO2
20	Query analysis using Query Plan/Show Plan	CO2
21	Familiarization of NoSQL Databases and CRUD operations**.	CO5
22	Design a database application using any front end tool for any problem selected. The application constructed should have five or more tables**.	CO6

# CYCLE 1



## EXP NO: 1

Design a database schema for an application with ER diagram from a problem description AIM:

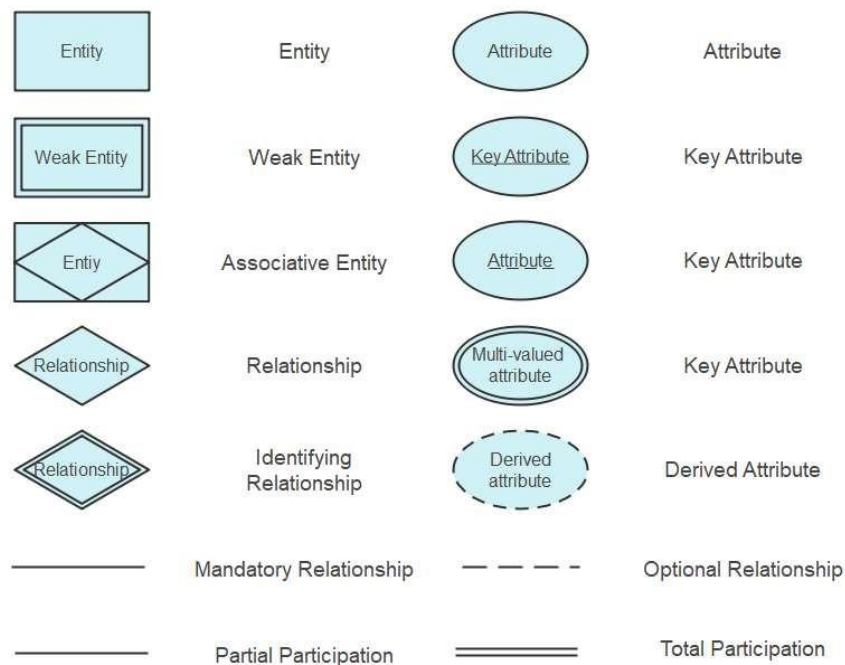
To design a database schema for an application with ER diagram from a problem description

### THEORY:

The ER model represents real world situations using concepts, which are commonly used by people. It allows defining a representation of the real world at logical level

- Entity: An entity is something which is described in the database by storing its data, it may be a concrete entity or a conceptual entity.
- Entity set: An entity set is a collection of similar entities.
- Attribute: An attribute describes a property associated with entities. Attribute will have a name and a value for each entity.
- Domain: A domain defines a set of permitted values for an attribute

### SYMBOLS IN E-R DIAGRAM



### Problem 1:

The COMPANY database keeps track of a company's employees, departments, and projects. Suppose that after the requirements collection and analysis phase, the database designers provide the following description of the miniworld—the part of the company that will be represented in the database.

- The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- We store each employee's name, Social Security number, address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
- We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

### Problem 2:

The Library Management System database keeps track of readers with the following considerations

—

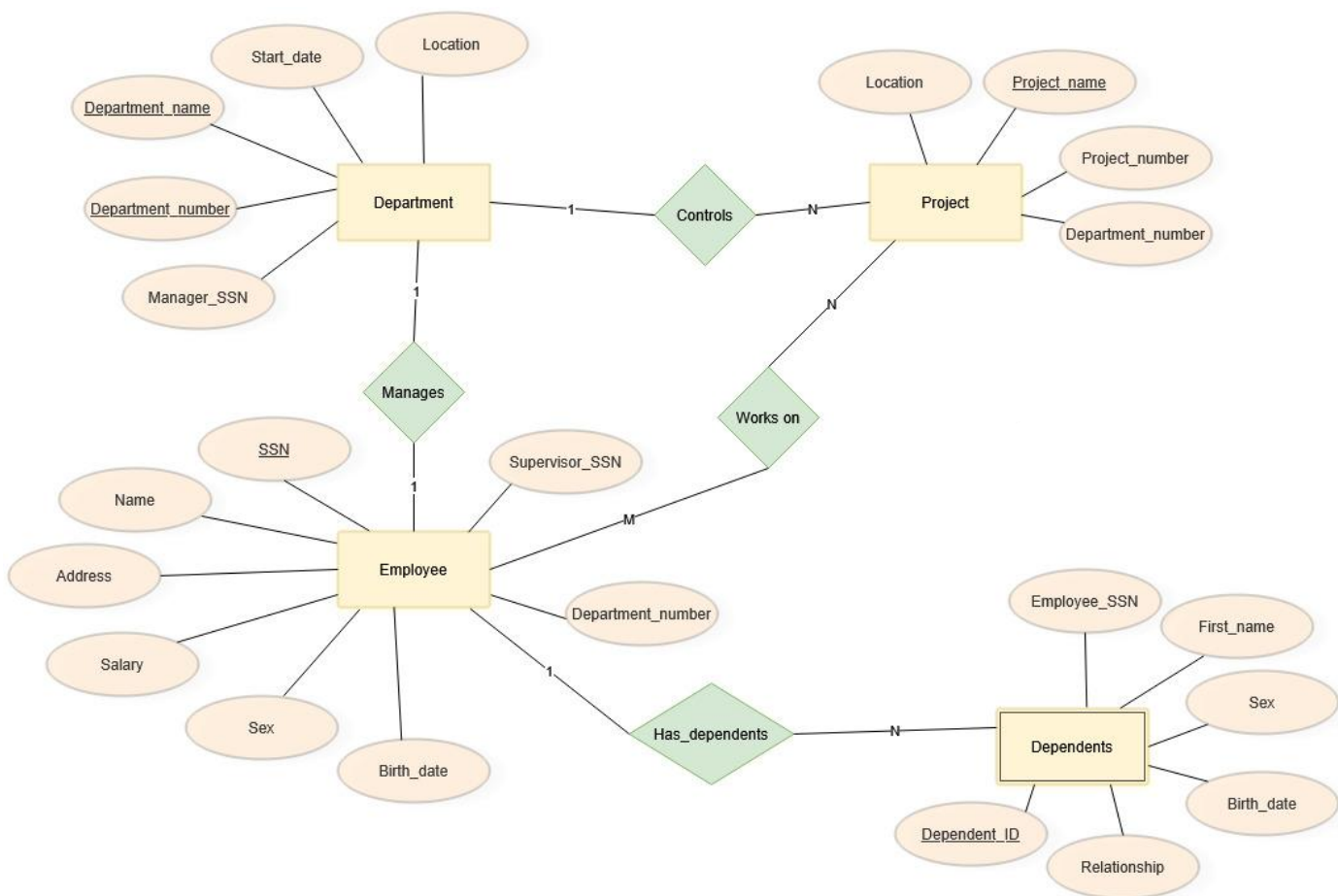
- The system keeps track of the staff with a single point authentication system comprising login Id and password.
- Staff maintains the book catalog with its ISBN, Book title, price(in INR), category(novel, general, story), edition, author Number and details.
- A publisher has publisher Id, Year when the book was published, and name of the book.
- Readers are registered with their user\_id, email, name (first name, last name), Phone no (multiple entries allowed), communication address. The staff keeps track of readers.
- Readers can return/reserve books that stamps with issue date and return date. If not returned within the prescribed time period, it may have a due date too.

- Staff also generate reports that has readers id, registration no of report, book no and return/issue info.

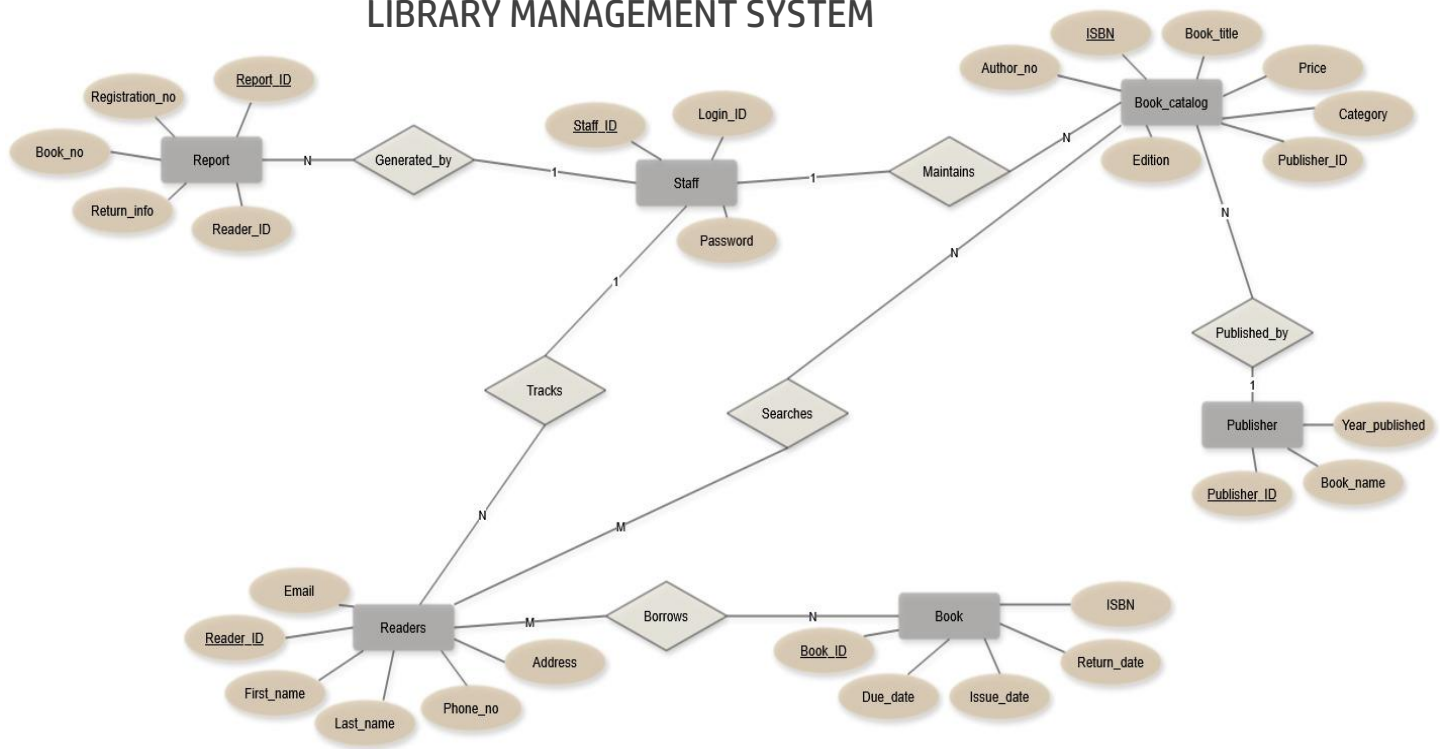
Question :

1. Draw ER diagrams for both problem description

### COMPANY DATABASE



## LIBRARY MANAGEMENT SYSTEM



## RESULT

Designed ER Diagrams for both problem descriptions

## EXP NO: 2

Creation, modification, configuration, and deletion of databases  
Commands

AIM:

To creation, modification, configuration, and deletion of databases using UI and SQL  
Commands COMMANDS

THEORY

### 1. CREATE DATABASE

The **CREATE DATABASE** statement is used to create a new SQL database.

SYNTAX

**CREATE DATABASE** <databasename>

EXAMPLE

Create database CMS

### 2. DROP DATABASE

The **DROP DATABASE** statement is used to drop an existing SQL database.

SYNTAX

**DROP DATABASE** <databasename>

EXAMPLE

Drop database CMS

### 3. USE DATABASE

This will display the database created

SYNTAX : Use <database name>

EXAMPLE : Use CMS

### Questions:

1. Create a database for company Management System(CMS)

```
mysql> CREATE database CMS;  
Query OK, 1 row affected (0.01 sec)
```

2. Drop database

```
mysql> DROP database CMS;  
Query OK. 0 rows affected (0.01 sec)
```

3. Rename any database created

```
RENAME CMS CMS2;
```

4. Use database

```
mysql> use CMS;  
Database changed
```

5. Describe database

```
mysql> SHOW TABLES;  
+-----+  
| Tables_in_CMS |  
+-----+  
| employees      |  
| departments    |  
| projects       |  
+-----+  
3 rows in set (0.00 sec)
```

### RESULT

Created and used CMS database successfully. DDL commands Executed successfully and desired output is received.

## EXP NO: 3

### Applications of DDL Commands Using UI and SQL

#### Aim

Creation of database schema - DDL (create tables, set constraints, enforce relationships, create indices, delete and modify tables).

#### THEORY

##### 1. CREATE TABLE

The CREATE TABLE statement is used to create a new table in a database.

##### SYNTAX

```
CREATE TABLE table_name (  
    column1 datatype, column2  
    datatype, column3 datatype, ....  
);
```

##### 2. ALTER TABLE

The **ALTER TABLE** statement is used to add, delete, or modify columns in an existing table.

The **ALTER TABLE** statement is also used to add and drop various constraints on an existing table.

##### SYNTAX

1. **ALTER TABLE** table\_name **ADD** column\_name datatype;
2. **ALTER TABLE** table\_name **DROP COLUMN** column\_name;
3. **ALTER TABLE** table\_name **RENAME COLUMN** old\_name **to** new\_name;

##### 3. MODIFY TABLE

1. **ALTER TABLE** table\_name **MODIFY COLUMN** column\_name datatype;

##### 4. DROP TABLE

The **DROP TABLE** statement is used to drop an existing table in a database.

1. **DROP TABLE** table\_name;

##### 5. TRUNCATE TABLE

The **TRUNCATE TABLE** statement is used to delete the data inside a table, but not the table itself. 1.

```
TRUNCATE TABLE table_name;
```

## SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- [NOT NULL](#) - Ensures that a column cannot have a NULL value
- [UNIQUE](#) - Ensures that all values in a column are different
- [PRIMARY KEY](#) - A combination of a **NOT NULL** and **UNIQUE**. Uniquely identifies each row in a table
- [FOREIGN KEY](#) - Prevents actions that would destroy links between tables
- [CHECK](#) - Ensures that the values in a column satisfies a specific condition
- [DEFAULT](#) - Sets a default value for a column if no value is specified
- [CREATE INDEX](#) - Used to create and retrieve data from the database very quickly

5. Create following tables based on the description provided in Exp 1

a. Department Table:

Attributes: DepartmentID (Primary Key), Name, Number, ManagerID  
(Foreign Key referencing Employee), StartDate, Location

```
CREATE TABLE Department (  
    DepartmentID INT PRIMARY KEY,  
    Name VARCHAR(100) NOT NULL,  
    Number INT NOT NULL,  
    ManagerID INT,  
    StartDate DATE NOT NULL,  
    Location VARCHAR(100)  
);  
Query OK. 0 rows affected (0.01 sec)
```



- b. **Project Table:** Attributes: ProjectID (Primary Key), Name, Number, Location, DepartmentID (Foreign Key referencing Department)

```
CREATE TABLE Project (  
    ProjectID INT PRIMARY KEY,  
    Name VARCHAR(100) NOT NULL,  
    Number INT NOT NULL,  
    Location VARCHAR(100),  
    DepartmentID INT  
);  
Query OK. 0 rows affected (0.01 sec)
```

- c. **Employee Table:**

Attributes: EmployeeID (Primary Key), Name, SSN, Address, Salary, Gender, BirthDate, Role, DepartmentID (Foreign Key referencing Department), SupervisorID (Foreign Key referencing Employee)

```
CREATE TABLE Employee (  
    EmployeeID INT PRIMARY KEY,  
    Name VARCHAR(100) NOT NULL,  
    SSN CHAR(11) NOT NULL UNIQUE,  
    Address VARCHAR(255),  
    Salary DECIMAL(10, 2),  
    Gender CHAR(1),  
    BirthDate DATE,  
    Role VARCHAR(100),  
    DepartmentID INT,  
    SupervisorID INT  
);  
Query OK. 0 rows affected (0.02 sec)
```

- d. **ProjectAssignment Table:**

Attributes: AssignmentID (Primary Key), EmployeeID (Foreign Key referencing Employee), ProjectID (Foreign Key referencing Project), HoursPerWeek

```
CREATE TABLE ProjectAssignment (  
    AssignmentID INT PRIMARY KEY,  
    EmployeeID INT,  
    ProjectID INT,  
    HoursPerWeek DECIMAL(4, 2)  
);  
Query OK. 0 rows affected (0.01 sec)
```

- e. **Dependent Table:**

Attributes: DependentID (Primary Key), EmployeeID (Foreign Key referencing Employee), FirstName, Gender, BirthDate, Relationship

```
CREATE TABLE Dependent (  
    DependentID INT PRIMARY KEY,  
    EmployeeID INT,  
    FirstName VARCHAR(100),  
    Gender CHAR(1) CHECK (Gender IN ('M', 'F')),  
    BirthDate DATE,  
    Relationship VARCHAR(50)  
);  
Query OK. 0 rows affected (0.01 sec)
```

## 6. Alter tables to add appropriate constraints

```
mysql> ALTER TABLE Department ADD CONSTRAINT FK_Manager FOREIGN KEY  
(ManagerID) REFERENCES Employee(EmployeeID);  
Query OK, 0 rows affected (0.06 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> ALTER TABLE Project ADD CONSTRAINT FK_Department_Project  
FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID);  
Query OK, 0 rows affected (0.06 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> ALTER TABLE Employee ADD CONSTRAINT FK_Employee_Department  
FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),  
    ADD CONSTRAINT FK_Supervisor FOREIGN KEY (SupervisorID)  
REFERENCES Employee(EmployeeID);  
Query OK, 0 rows affected (0.06 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> ALTER TABLE ProjectAssignment ADD CONSTRAINT  
FK_Employee_Project FOREIGN KEY (EmployeeID) REFERENCES  
Employee(EmployeeID),  
    ADD CONSTRAINT FK_Project_Assignment FOREIGN KEY (ProjectID)  
REFERENCES Project(ProjectID);  
Query OK, 0 rows affected (0.06 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> ALTER TABLE Dependent ADD CONSTRAINT FK_Employee_Dependent  
FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID);  
Query OK, 0 rows affected (0.05 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

## RESULT

Created Tables with appropriate constraints successfully.

## EXP NO: 4

Database initialization - Data insert, Data import to a database (bulk import using UI and SQL Commands).

### AIM

To insert data to tables used in experiment no 3 using insert commands and bulk import using UI and sql commands.

### THEORY

The **INSERT INTO** statement is used to insert new records in a table.

### INSERT:

1. **INSERT INTO** table\_name (column1, column2, column3,..) **VALUES** (value1, value2, value3, ...);
2. **INSERT INTO** table\_name  
**VALUES** (value1, value2, value3, ...);

### QUERIES

1. Insert appropriate values to all tables created in Exp No. 3

```
mysql> INSERT INTO Department (DepartmentID, Name, Number, ManagerID,
StartDate, Location)
-> VALUES
-> (1, 'HR', '9U47000000', NULL, '2020-01-01', 'Kochi'),
-> (2, 'IT', '9U47000001', NULL, '2021-06-15', 'Thiruvananthapuram'),
-> (3, 'Finance', '9U47000002', NULL, '2022-03-10', 'Kozhikode'),
-> (4, 'Marketing', '9U47000003', NULL, '2019-02-20', 'Thrissur'),
-> (5, 'Operations', '9U47000004', NULL, '2020-08-05', 'Alappuzha'),
-> (6, 'Sales', '9U47000005', NULL, '2021-11-12', 'Pathanamthitta'),
-> (7, 'Support', '9U47000006', NULL, '2021-01-25', 'Kochi');
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO Project (ProjectID, Name, Number, Location, DepartmentID)
-> VALUES
-> (1, 'Employee Portal', 'PR100001', 'Kochi', 1),
-> (2, 'AI System', 'PR100002', 'Thiruvananthapuram', 2),
-> (3, 'Finance Tracker', 'PR100003', 'Kozhikode', 3),
-> (4, 'Marketing Campaign', 'PR100004', 'Thrissur', 4),
-> (5, 'Inventory System', 'PR100005', 'Alappuzha', 5),
-> (6, 'Sales Dashboard', 'PR100006', 'Pathanamthitta', 6),
-> (7, 'Helpdesk Portal', 'PR100007', 'Malappuram', 7);
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO Employee (EmployeeID, Name, SSN, Address, Salary, Gender, BirthDate, Role,
DepartmentID, SupervisorID)
-> VALUES
-> (1, 'Arjun', 'SSN001', 'Kochi', 60000, 'M', '1995-05-15', 'Manager', 1, NULL),
-> (2, 'Meera', 'SSN002', 'Kollam', 55000, 'F', '1998-03-20', 'Engineer', 2, 1),
-> (3, 'Rahul', 'SSN003', 'Kannur', 52000, 'M', '1997-12-10', 'Analyst', 3, 2),
-> (4, 'Nina', 'SSN004', 'Thrissur', 50000, 'F', '1999-11-05', 'Designer', 4, 2),
-> (5, 'John', 'SSN005', 'Kozhikode', 48000, 'M', '2000-07-30', 'Support', 5, 3),
-> (6, 'Sara', 'SSN006', 'Kochi', 62000, 'F', '1993-08-22', 'HR Lead', 1, 1),
-> (7, Atul, 'SSN007', 'Mala', 510000, 'M', '2005-04-13', 'CEO', 5, 1);
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO ProjectAssignment (AssignmentID, EmployeeID, ProjectID, HoursPerWeek)
-> VALUES
-> (1, 1, 1, 20),
-> (2, 2, 2, 25),
-> (3, 3, 3, 30),
-> (4, 4, 4, 20),
-> (5, 5, 5, 15),
-> (6, 6, 6, 18),
-> (7, 7, 7, 22);
Query OK, 7 rows affected (0.00 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO Dependent (DependentID, EmployeeID, FirstName, Gender, BirthDate,
Relationship)
-> VALUES
-> (1, 1, 'Aarav', 'M', '2015-06-10', 'Son'),
-> (2, 2, 'Anaya', 'F', '2018-09-12', 'Daughter'),
-> (3, 3, 'Krish', 'M', '2016-11-20', 'Son'),
-> (4, 4, 'Maya', 'F', '2013-04-05', 'Daughter'),
-> (5, 5, 'Vikram', 'M', '2017-02-14', 'Son'),
-> (6, 6, 'Diya', 'F', '2020-01-30', 'Daughter'),
-> (7, 7, 'Kabir', 'M', '2022-05-25', 'Son');
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

## RESULT

Data inserted into Tables successfully.

## EXP NO: 5

### DML Commands

#### AIM

Practice SQL commands for DML (insertion, updating, altering, deletion of data, and viewing/querying records based on condition in databases)

#### THEORY

##### SELECT:

2. `SELECT column1, column2, .. FROM table_name;`
3. `SELECT DISTINCT column1, column2, ...`  
`FROM table_name; (Shows only unique rows)`
4. `SELECT column1, column2, ... FROM table_name WHERE condition;`  
`(Where is used to filter records)`
5. `SELECT column1, column2, ... FROM table_name`  
`ORDER BY column1, column2, ... ASC|DESC`  
`INSERT : ( will sort the columns in ascending or descending order)`
6. `SELECT column1, column2, ... FROM table_name`  
`WHERE columnN LIKE pattern;`

##### UPDATE:

1. `UPDATE table_name`  
`SET column1 = value1, column2 = value2, ... WHERE condition;`

##### DELETE:

1. `DELETE FROM table_name WHERE condition;`

#### Algorithm:

Consider the employee database created. Find the following

- 1) Find the names of all employees who work for SBI.

```
mysql> SELECT Name FROM employees WHERE CompanyName = 'SBI';
+-----+
| Name          |
+-----+
| Atul          |
| Meera K       |
| Kiran S       |
+-----+
3 rows in set (0.00 sec)
```

- 2) Find all employees in the database who live in the same cities as the companies for which they work.

```
mysql> SELECT Name
-> FROM employees WHERE City = CompanyCity;
+-----+
| Name          |
+-----+
| Atul          |
| Rajesh Nair   |
| Anjali Pillai |
| Sreeram G     |
| Nandini R     |
| Binu K        |
| Anju S        |
+-----+
7 rows in set (0.00 sec)
```

- 3) Find all employees and their managers in the database who live in the same cities and on the same street number as do their managers.

```
mysql> SELECT Name
-> FROM employees
-> WHERE City = ManagerCity;
+-----+
| Name          |
+-----+
| Atul          |
| Babu Battery  |
| Anjali Pillai |
+-----+
3 rows in set (0.00 sec)
```

- 4) Find all employees who earn more than the average salary of all employees of their company.

```
mysql> SELECT E.Name
-> FROM employees E
-> WHERE E.Salary > (SELECT AVG(Salary) FROM employees WHERE
CompanyName = E.CompanyName);
+-----+
| Name          |
+-----+
| Zayed Ikka    |
| Anjali Pillai |
| Sreeram G     |
| Suresh Kumar  |
| Sahil         |
| Anju S        |
+-----+
5 rows in set (0.00 sec)
```

- 5) Find the company that pays the least total salary along with the salary paid.

```
mysql> SELECT CompanyName, SUM(Salary) AS TotalSalary
-> FROM employees
-> GROUP BY CompanyName
-> ORDER BY TotalSalary ASC
-> LIMIT 1;
+-----+-----+
| CompanyName | TotalSalary |
+-----+-----+
| Indian Bank | 100000.00   |
+-----+-----+
1 row in set (0.00 sec)
```

- 6) Give all managers of SBI a 10 percent raise.

```
mysql> UPDATE employees
-> SET Salary = Salary * 1.10
-> WHERE CompanyName = 'SBI' AND Role LIKE '%Manager%';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```



7) Find the company that has the most employees

```
mysql> SELECT CompanyName
-> FROM employees
-> GROUP BY CompanyName
-> ORDER BY COUNT(EmployeeID) DESC
-> LIMIT 1;

+-----+
| CompanyName |
+-----+
| SBI          |
+-----+
1 row in set (0.00 sec)
```

8) Find those companies whose employees earn a higher salary, on average than the average salary at Indian Bank.

```
mysql> SELECT CompanyName
-> FROM employees
-> GROUP BY CompanyName
-> HAVING AVG(Salary) > (SELECT AVG(Salary) FROM employees
WHERE CompanyName = 'Indian Bank');

+-----+
| CompanyName |
+-----+
| SBI          |
| HDFC         |
| ICICI        |
+-----+
3 rows in set (0.00 sec)
```

9) Change the department of one employee to another.

```
mysql> UPDATE employees
-> SET Department = 'HR'
-> WHERE EmployeeID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

10) Remove employee with ID 203 from the database.

```
mysql> DELETE FROM employees WHERE EmployeeID = 203;
Query OK, 0 rows affected (0.00 sec)
```

## RESULT

Thus, the DML was performed successfully and executed.

## EXP NO: 6

### Implementation of Built in Functions

AIM:

Implementation of built in functions in RDBMS

Theory

#### RDBMS Built in Functions

There are two types of functions:

- 1) Single Row Functions: Single row or Scalar functions return a value for every row that is processed in a query.
- 2) Group Functions: These functions group the rows of data based on the values returned by the query. This is discussed in SQL GROUP Functions. The group functions are used to calculate aggregate values like total or average, which return just one total or one average value after processing a group of rows.

There are four types of single row functions. They are:

- 1) Numeric Functions: These are functions that accept numeric input and return numeric values.
- 2) Character or Text Functions: These are functions that accept character input and can return both character and number values.
- 3) Date Functions: These are functions that take values that are of datatype DATE as input and return values of datatype DATE, except for the MONTHS\_BETWEEN function, which returns a number.
- 4) Conversion Functions: These are functions that help us to convert a value in one form to another form. For Example: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO\_CHAR, TO\_NUMBER, TO\_DATE etc.

You can combine more than one function together in an expression. This is known as nesting of functions.

#### 1. Numeric Functions:

Numeric functions are used to perform operations on numbers. They accept numeric values as input and return numeric values as output. Few of the Numeric functions are:

Function Name	Return Value
ABS (x)	Absolute value of the number 'x'
CEIL (x)	Integer value that is Greater than or equal to the number 'x'
FLOOR (x)	Integer value that is Less than or equal to the number 'x'
TRUNC (x, y)	Truncates value of number 'x' up to 'y' decimal places
ROUND (x, y)	Rounded off value of the number 'x' up to the number 'y' decimal places

The following examples explain the usage of the above numeric functions :

Function Name	Examples	Return Value
ABS (x)	ABS (1) ABS (-1)	1 -1
CEIL (x)	CEIL (2.83) CEIL (2.49) CEIL (-1.6)	3 3 -1
FLOOR (x)	FLOOR (2.83) FLOOR (2.49) FLOOR (-1.6)	2 2 -2
ROUND (x, y)	ROUND (125.456, 1) ROUND (125.456, 0) ROUND (124.456, -1)	125.4 125 120

## 2. Character or Text Functions:

Character or text functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output.

Few of the character or text functions are as given below:

Function Name	Return Value
LOWER (string_value)	All the letters in 'string_value' is converted to lowercase.
UPPER (string_value)	All the letters in 'string_value' is converted to uppcase.
LTRIM (string_value, trim_text)	All occurrences of 'trim_text' is removed from the left of 'string_value'.
RTRIM (string_value, trim_text)	All occurrences of 'trim_text' is removed from the right of 'string_value' .
TRIM (trim_text FROM string_value)	All occurrences of 'trim_text' from the left and right of 'string_value' , 'trim_text' can also be only one character long .
SUBSTR (string_value, m, n)	Returns 'n' number of characters from 'string_value' starting from the 'm' position.
LENGTH (string_value)	Number of characters in 'string_value' in returned.
LPAD (string_value, n, pad_value)	Returns 'string_value' left-padded with 'pad_value' . The length of the whole string will be of 'n' characters.
RPAD (string_value, n, pad_value)	Returns 'string_value' right-padded with 'pad_value' . The length of whole string will be of 'n' characters.

For Example, we can use the above UPPER() text function with the column value as follows. SELECT UPPER (product\_name) FROM product;

The following examples explains the usage of the above character or text functions

Function Name	Examples	Return Value
LOWER(string_value)	LOWER('Good Morning')	good morning
UPPER(string_value)	UPPER('Good Morning')	GOOD MORNING
INITCAP(string_value)	INITCAP('GOOD MORNING')	Good Morning
LTRIM(string_value, trim_text)	LTRIM ('Good Morning', 'Good')	Morning

RTRIM (string_value, trim_text)	RTRIM ('Good Morning', 'Morning')	Good
TRIM (trim_text FROM string_value)	TRIM ('o' FROM 'Good Morning')	Gd Mrning
SUBSTR (string_value, m, n)	SUBSTR ('Good Morning', 6, 7)	Morning
LENGTH (string_value)	LENGTH ('Good Morning')	12
LPAD (string_value, n, pad_value)	LPAD ('Good', 6, '*')	**Good
RPAD (string_value, n, pad_value)	RPAD ('Good', 6, '*')	Good**

### 3. Date Functions:

These are functions that take values that are of datatype DATE as input and return values of datatypes DATE, except for the MONTHS\_BETWEEN function, which returns a number as output.

Few date functions are as given below:

Function Name	Return Value
ADD_MONTHS (date, n)	Returns a date value after adding 'n' months to the date 'x'.
MONTHS_BETWEEN (x1, x2)	Returns the number of months between dates x1 and x2.
ROUND (x, date_format)	Returns the date 'x' rounded off to the nearest century, year, month, date, hour, minute, or second as specified by the 'date_format'.
TRUNC (x, date_format)	Returns the date 'x' lesser than or equal to the nearest century, year, month, date, hour, minute, or second as specified by the 'date_format'.
NEXT_DAY (x, week_day)	Returns the next date of the 'week_day' on or after the date 'x' occurs.
LAST_DAY (x)	It is used to determine the number of days remaining in a month from the date 'x' specified.
SYSDATE	Returns the systems current date and time.

NEW_TIME (x, zone1, zone2)	Returns the date and time in zone2 if date 'x' represents the time in zone1.
----------------------------	--

The below table provides the examples for the above functions

Function Name	Examples	Return Value
ADD_MONTHS ( )	ADD_MONTHS ('16-Sep-81', 3)	16-Dec-81
MONTHS_BETWEEN( )	MONTHS_BETWEEN('16-Sep-81', '16-Dec-81')	3
NEXT_DAY( )	NEXT_DAY ('01-Jun-08', 'Wednesday')	04-JUN-08
LAST_DAY( )	LAST_DAY ('01-Jun-08')	30-Jun-08
NEW_TIME( )	NEW_TIME ('01-Jun-08', 'IST', 'EST')	31-May-08

#### 4. Conversion Functions:

These are functions that help us to convert a value in one form to another form. For Ex: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO\_CHAR, TO\_NUMBER, TO\_DATE.

Few of the conversion functions available in oracle are:

Function Name	Return Value
TO_CHAR (x [,y])	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.
TO_DATE (x [, date_format])	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by 'date_format'.
NVL (x, y)	If 'x' is NULL, replace it with 'y'. 'x' and 'y' must be of the same datatype.
DECODE(a, b, c, d, e, default_value)	Checks the value of 'a', if a = b, then returns 'c'. If a = d, then returns Else, returns default_value.

The below table provides the examples for the above functions

Function Name	Examples	Return Value
TO_CHAR ()	TO_CHAR (3000, '\$9999')	\$3000
	TO_CHAR (SYSDATE, 'Day, Month YYYY')	Monday, June 2008
TO_DATE ()	TO_DATE ('01-Jun-08')	01-Jun-08
NVL ()	NVL (null, 1)	1

Queries:

Q1: Display all the details of the records whose employee name starts with 'A'. Solution:

```
mysql> SELECT * FROM employees WHERE Name LIKE 'A%';
```

EmployeeID	Name	SSN	Address	Salary	Gender	BirthDate
Role	CompanyName	Department	ManagerName	City		
3	Anjali Pillai	34567	Kochi, Kerala	55000.00	F	1992-07-15
Finance Analyst	Indian Bank	Finance	Rajesh Nair	Kochi		
10	Anju S	01234	Kollam, Kerala	52000.00	F	1994-08-25
Marketing Manager	ICICI	Marketing	Ravi Varma	Kollam		

2 rows in set (0.00 sec)

Q2: Display all the details of the records whose employee name does not start with 'A'.

```
mysql> SELECT * FROM employees WHERE Name NOT LIKE 'A%';
```

EmployeeID	Name	SSN	Address	Salary	Gender	BirthDate
Role	CompanyName	Department	ManagerName	City		
1	Gokul P	12345	Palakkad, Kerala	50000.00	M	1990-05-10
Software Engineer	SBI	HR	NULL	Palakkad		
2	Rajesh Nair	23456	Ernakulam, Kerala	66000.00	M	1988-03-25
HR Manager	SBI	HR	NULL	Ernakulam		
4	Sreeram G	45678	Trivandrum, Kerala	65000.00	M	1987-02-20
Developer	SBI	IT	Rajesh Nair	Trivandrum		

3 rows in set (0.00 sec)

Q3: Display the rows whose salary ranges from 15000 to 30000.

```
mysql> SELECT * FROM employees WHERE Salary BETWEEN 15000 AND 30000;
Empty set (0.00 sec)
```

Q4: Calculate the total and average salary amount of the emp table.

```
mysql> SELECT SUM(Salary) AS TotalSalary, AVG(Salary) AS
AverageSalary FROM employees;
+-----+-----+
| TotalSalary | AverageSalary |
+-----+-----+
| 551000.00   | 55100.000000  |
+-----+-----+
1 row in set (0.00 sec)
```

Q5: Count the total records in the emp table.

```
mysql> SELECT COUNT(*) AS TotalRecords FROM employees;
+-----+
| TotalRecords |
+-----+
| 10           |
+-----+
1 row in set (0.00 sec)
```

Q6: Determine the max and min salary and rename the column as max\_salary and min\_salary.

```
mysql> SELECT MAX(Salary) AS max_salary, MIN(Salary) AS min_salary
FROM employees;
+-----+-----+
| max_salary | min_salary |
+-----+-----+
| 70000.00   | 40000.00   |
+-----+-----+
1 row in set (0.00 sec)
```

Q7: Display the month between “1-jun-10” and 1-aug-10 in full.

```
SELECT MONTHS_BETWEEN('01-JUN-10', '01-AUG-10') FROM dual;
MONTHS_BETWEEN('01-JUN-10', '01-AUG-10')
-----
-2
```

Q8: Display the last day of that month in “05-Oct-09”.

```
SELECT LAST_DAY('05-OCT-09') FROM dual;
LAST_DAY
-----
31-10-09
```



Q9: Find how many job titles are available in the employee table.

```
SELECT COUNT(DISTINCT Role) AS TotalJobTitles FROM employees;
+-----+
| TotalJobTitles |
+-----+
|           10 |
+-----+
1 row in set (0.00 sec)
```

Q10: What is the difference between maximum and minimum salaries of employees in the organization?

```
SELECT MAX(Salary) - MIN(Salary) AS SalaryDifference FROM employees;
+-----+
| SalaryDifference |
+-----+
|       30000.00 |
+-----+
1 row in set (0.00 sec)
```

## RESULT

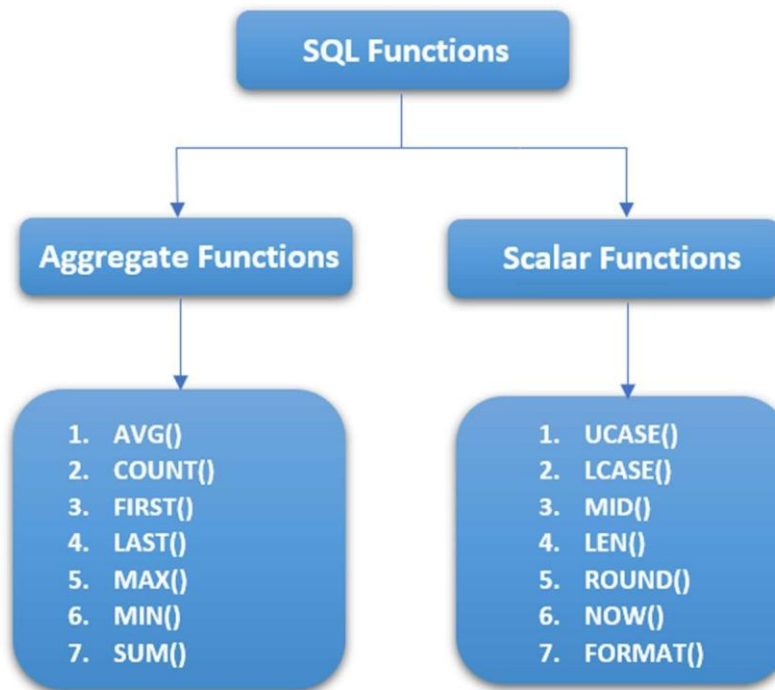
Thus, the nested Queries and join Queries were performed successfully and executed.

## EXP NO: 7

Implementation of various aggregate functions in SQL

Aim:

Implementation of various aggregate functions in SQL Theory:



Function Name	Meaning	Example
SUM(column name)	Total sum of the values in a numeric column	SUM(salary)
AVG(column name)	Average of the values in a column	AVG(salary)
MAX(column name)	Largest value in a column	MAX(salary)
MIN(column name)	Smallest value in a column	MIN(salary)
COUNT(*)	Count of the number of rows selected	COUNT(*)

1. **SELECT COUNT**(column\_name) **FROM** table\_name **WHERE** condition;
2. **SELECT AVG**(column\_name) **FROM** table\_name **WHERE** condition;
3. **SELECT SUM**(column\_name) **FROM** table\_name **WHERE** condition;
4. **SELECT MIN**(column\_name) **FROM** table\_name **WHERE** condition;
5. **SELECT MAX**(column\_name) **FROM** table\_name **WHERE** condition;

QUERIES

1. Count the total number of employees.

```
SELECT COUNT(*) AS TotalEmployees FROM employees;
+-----+
| TotalEmployees|
+-----+
|              10|
+-----+
1 row in set (0.00 sec)
```

2. Find the average salary of all employees.

```
SELECT AVG(Salary) AS AverageSalary FROM employees;
+-----+
| AverageSalary |
+-----+
| 55100.00000000 |
+-----+
1 row in set (0.00 sec)
```

3. Find the highest salary among all employees.

```
SELECT MAX(Salary) AS HighestSalary FROM employees;
+-----+
| HighestSalary |
+-----+
|      70000.00 |
+-----+
1 row in set (0.00 sec)
```

4. Count the number of employees in each role.

```
SELECT Role, COUNT(*) AS TotalEmployees FROM employees GROUP BY Role;
```

```
+-----+-----+
| Role                | TotalEmployees |
+-----+-----+
| Software Engineer   | 1 |
| HR Manager          | 1 |
| Finance Analyst     | 1 |
| Developer           | 1 |
| HR Executive        | 1 |
| Finance Manager     | 1 |
| Sales Executive     | 1 |
| Customer Support    | 1 |
| Operations Manager  | 1 |
| Marketing Manager   | 1 |
+-----+-----+
10 rows in set (0.00 sec)
```

5. Calculate the overall sum of hours worked on projects.

```
SELECT SUM(HoursPerWeek) AS TotalHoursWorked FROM ProjectAssignment;
```

```
+-----+
| TotalHoursWorked |
+-----+
| 368.00 |
+-----+
1 row in set (0.00 sec)
```

6. Count the total number of projects.

```
SELECT COUNT(ProjectID) AS TotalProjects FROM Project;
```

```
+-----+
| TotalProjects |
+-----+
| 10 |
+-----+
1 row in set (0.00 sec)
```

7. Calculate the average number of dependents for each employee.

```
SELECT EmployeeID, AVG(DependentCount) AS AvgDependents
FROM (
    SELECT EmployeeID, COUNT(DependentID) AS DependentCount
    FROM Dependent
    GROUP BY EmployeeID
) AS DependentsPerEmployee
GROUP BY EmployeeID;
```

```
+-----+-----+
| EmployeeID | AvgDependents |
+-----+-----+
|          1 |          1.0000 |
|          2 |          1.0000 |
|          3 |          1.0000 |
|          4 |          1.0000 |
|          5 |          1.0000 |
|          6 |          1.0000 |
|          7 |          1.0000 |
|          8 |          1.0000 |
|          9 |          1.0000 |
|         10 |          1.0000 |
+-----+-----+
10 rows in set (0.00 sec)
```

8. Find the project with the highest total hours worked.

```
SELECT ProjectID, SUM(HoursPerWeek) AS TotalHoursWorked
FROM ProjectAssignment
GROUP BY ProjectID
ORDER BY TotalHoursWorked DESC
LIMIT 1;
```

```
+-----+-----+
| ProjectID | TotalHoursWorked |
+-----+-----+
|          1 |          40.00 |
+-----+-----+
1 row in set (0.00 sec)
```

9. Find the earliest start date of employees in each department.

```
SELECT DepartmentID, MIN(BirthDate) AS EarliestStartDate
FROM Employee
GROUP BY DepartmentID;
```

```
+-----+-----+
|DepartmentID | EarliestStartDate |
+-----+-----+
|          1 | 1988-03-25      |
|          2 | 1987-02-20      |
|          3 | 1985-09-30      |
|          4 | 1994-08-25      |
|          5 | 1989-07-10      |
|          6 | 1991-12-12      |
|          7 | 1990-06-20      |
+-----+-----+
7 rows in set (0.00 sec)
```

10. Retrieve the total salary spent by each department.

```
SELECT DepartmentID, SUM(Salary) AS TotalSalary
FROM Employee
GROUP BY DepartmentID;
```

```
+-----+-----+
| DepartmentID | TotalSalary |
+-----+-----+
|          1 | 105000.00   |
|          2 | 115000.00   |
|          3 | 125000.00   |
|          4 | 52000.00    |
|          5 | 60000.00    |
|          6 | 40000.00    |
|          7 | 48000.00    |
+-----+-----+
7 rows in set (0.00 sec)
```

### 11. Average Salary of Employees in Each Department:

```
SELECT DepartmentID, AVG(Salary) AS AVERAGESalary
FROM Employee
GROUP BY DepartmentID;
```

```
+-----+-----+
|DepartmentID | AVERAGESalary |
+-----+-----+
|          1 | 52500.000000000 |
|          2 | 57500.000000000 |
|          3 | 62500.000000000 |
|          4 | 52000.000000000 |
|          5 | 60000.000000000 |
|          6 | 40000.000000000 |
|          7 | 48000.000000000 |
+-----+-----+
7 rows in set (0.00 sec)
```

### 12. Calculate the total hours worked on projects by each employee.

```
SELECT EmployeeID, SUM(HoursPerWeek) AS TotalHoursWorked
FROM ProjectAssignment
GROUP BY EmployeeID;
```

```
+-----+-----+
| EmployeeID | TotalHoursWorked |
+-----+-----+
|          1 |          40.00 |
|          2 |          35.00 |
|          3 |          40.00 |
|          4 |          38.00 |
|          5 |          36.00 |
|          6 |          40.00 |
|          7 |          30.00 |
|          8 |          32.00 |
|          9 |          37.00 |
|         10 |          40.00 |
+-----+-----+
10 rows in set (0.00 sec)
```

## RESULT

Thus, the Queries using Aggregate functions were performed successfully and executed.

## EXP NO: 8

Implementation of Order By, Group By & Having clause

AIM

Implementation of Order By, Group By & Having clause

THEORY

GROUP BY

The **GROUP BY** statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The **GROUP BY** statement is often used with aggregate functions (**COUNT()**, **MAX()**, **MIN()**, **SUM()**, **AVG()**) to group the result-set by one or more columns.

```
SELECT column_name(s) FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Example: SQL> SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY EMPNO;

GROUP BY-HAVING :

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```



Example : `SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5;`

ORDER BY:

This query is used to display a selected set of fields from a relation in an ordered manner based on some field.

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1, column2, ... ASC|DESC;
```

Example: `SQL> SELECT empno, ename, job FROM emp ORDER BY job;`

## QUESTIONS

### 1. Order Employees by Salary in Descending Order

```
SELECT Name, Salary  
FROM employees  
ORDER BY Salary DESC;  
+-----+-----+  
| Name          | Salary    |  
+-----+-----+  
| Suresh Kumar  | 70000.00  |  
| Rajesh Nair   | 66000.00  |  
| Sreream G     | 65000.00  |  
| Ravi Varma    | 60000.00  |  
| Anjali Pillai | 55000.00  |  
| Anju S        | 52000.00  |  
| Gokul P       | 50000.00  |  
| Binu K        | 48000.00  |  
| Nandini R     | 45000.00  |  
| Neethu M      | 40000.00  |  
+-----+-----+  
10 rows in set (0.00 sec)
```

## 2. Order Companies by Total Employees in Ascending Order

```
SELECT CompanyName, COUNT(EmployeeID) AS TotalEmployees
FROM employees
GROUP BY CompanyName
ORDER BY TotalEmployees ASC;
```

CompanyName	TotalEmployees
Indian Bank	2
HDFC	2
SBI	3
ICICI	3

4 rows in set (0.00 sec)

## 3. Find Companies with More Than 50 Employees

```
SELECT CompanyName, COUNT(EmployeeID) AS TotalEmployees
FROM employees
GROUP BY CompanyName
HAVING COUNT(EmployeeID) > 50;
```

Empty set (0.00 sec)

## 4. Count the number of employees in each city.

```
SELECT City, COUNT(EmployeeID) AS TotalEmployees
FROM employees
GROUP BY City;
```

City	TotalEmployees
Palakkad	1
Ernakulam	1
Kochi	1
Trivandrum	1
Kottayam	1
Kozhikode	1
Alappuzha	1
Malappuram	1
Pathanamthitta	1
Kollam	1

10 rows in set (0.00 sec)

5. Find the total salary paid by each company

```
SELECT CompanyName, SUM(Salary) AS TotalSalaryPaid
FROM employees
GROUP BY CompanyName;
```

```
+-----+-----+
| CompanyName | TotalSalaryPaid |
+-----+-----+
| SBI         | 181000.00      |
| Indian Bank | 100000.00      |
| HDFC        | 118000.00      |
| ICICI       | 152000.00      |
+-----+-----+
4 rows in set (0.00 sec)
```

RESULT

Studied and implementation of Order By, Group By & Having clause.

## EXP NO 9

Implementation of set operators nested queries, and join queries

AIM

Implementation of set operators nested queries, and join queries

### THEORY

#### Set Operators And Nested Queries

A subQuery is a form of an SQL statement that appears inside another SQL statement. It is also termed as a nested Query. The statement contains a subQuery called a parent statement. The rows returned by the subQuery are used by the following statement.

#### Union Clause:

The user can put together multiple Queries and combine their output using the union clause . The union clause merges the output of two or more Queries into a single set of rows and columns. The final output of union clause will be

Output: = Records only in Query one + records only in Query two + A single set of records which are common in both Queries.

#### Syntax:

```
SELECT columnname, columnname FROM tablename 1 UNION SELECT columnname, columnname  
From tablename2;
```

#### Intersect Clause:

The user can put together multiple Queries and their output using the interest clause.

The final output of the interest clause will be : A single set of records which are common in both Queries

```
SELECT columnname, columnname FROM tablename 1 INTERSECT
```

```
SELECT columnname, columnname FROM tablename 2;
```

#### Minus Clause:

The user can put together multiple Queries and combine their output

Output:= records only in Query one

Syntax:

```
SELECT columnname, columnname FROM tablename 1 MINUS
```

```
SELECT columnname, columnname FROM tablename 2
```

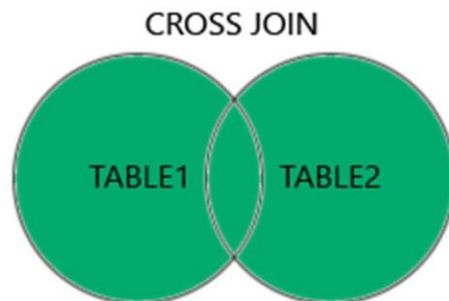
Join queries

Joint Multiple Table (Equi Join): Sometimes we require to treat more than one table as though to manipulate data from all the tables as though the tables were not separate objects but one single entity. To achieve this we have to join tables. Tables are joined on columns that have data type and data within tables.

The tables that have to be joined are specified in the FROM clause and the joining attributes in the WHERE clause.

## 1. Cartesian product/Cross Join

The **CROSS JOIN** keyword returns all records from both tables (table1 and table2).

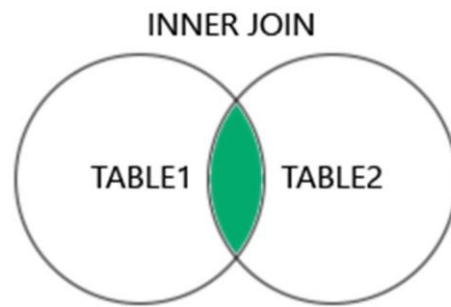


CROSS JOIN Syntax

```
SELECT column_name(s) FROM table1  
CROSS JOIN table2;
```

## 2. Inner Join:

The **INNER JOIN** keyword selects records that have matching values in both tables.

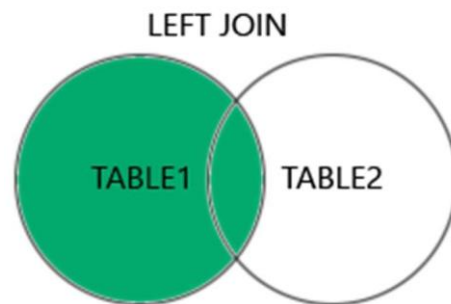


INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

### 3. Left Join:

The **LEFT JOIN** keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).

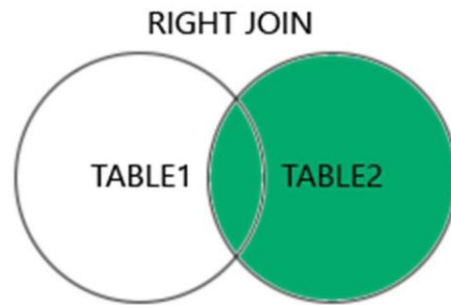


LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

### 4. Right Join:

The **RIGHT JOIN** keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).



#### RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

### Examples

Inner Join:

1. Retrieve the name of each employee along with the name of their department.

```
SELECT Employee.Name, Department.Name FROM Employee
```

```
INNER JOIN Department ON Employee.DepartmentID = Department.DepartmentID; Cross
```

Join:

2. Retrieve all possible combinations of employees and projects. `SELECT Employee.Name,`

```
Project.Name FROM Employee CROSS JOIN Project; Left Join:
```

3. Retrieve all employees along with their assigned project names (if any).

```
SELECT Employee.Name, Project.Name FROM Employee
```

```
LEFT JOIN ProjectAssignment ON Employee.EmployeeID =
```

```
ProjectAssignment.EmployeeID LEFT JOIN Project ON ProjectAssignment.ProjectID =
```

```
Project.ProjectID; Right Join:
```

4. Retrieve all projects along with the employees assigned to them (if any).

```
SELECT Project.Name, Employee.Name FROM Project
```

```
RIGHT JOIN ProjectAssignment ON Project.ProjectID = ProjectAssignment.ProjectID
```

```
RIGHT JOIN Employee ON ProjectAssignment.EmployeeID = Employee.EmployeeID;
```

## QUESTIONS

1. Retrieve the names of employees and their corresponding project names for all employees who are currently assigned to a project.

```
SELECT e.Name AS EmployeeName, p.ProjectName
FROM Employee e
JOIN ProjectAssignment pa ON e.EmployeeID = pa.EmployeeID
JOIN Project p ON pa.ProjectID = p.ProjectID;
+-----+-----+
| EmployeeName | ProjectName      |
+-----+-----+
| Zayed Ikka   | Website Redesign|
| Babu Battery | Mobile App Dev   |
| Sahil GN Du  | ERP System       |
| Bob Ross     | Marketing Ad     |
+-----+-----+
4 rows in set (0.00 sec)
```

2. Get the names of departments along with the total salary of employees in each department.

```
SELECT d.DepartmentName, SUM(e.Salary) AS TotalSalary
FROM Department d
LEFT JOIN Employee e ON d.DepartmentID = e.DepartmentID
GROUP BY d.DepartmentName;
+-----+-----+
| DepartmentName | TotalSalary      |
+-----+-----+
| HR              | 100000.00        |
| IT              | 150000.00        |
| Finance         | 120000.00        |
| Sales           | 80000.00         |
| Marketing       | NULL             |
+-----+-----+
5 rows in set (0.00 sec)
```



3. Get a list of all department names along with all project names without any specific relationship.

```
SELECT d.DepartmentName, p.ProjectName
FROM Department d
CROSS JOIN Project p;
```

DepartmentName	ProjectName
HR	Website Redesign
HR	Mobile App Dev
HR	ERP System
IT	Website Redesign
IT	Mobile App Dev
IT	ERP System
Finance	Website Redesign
Finance	Mobile App Dev
Finance	ERP System

9 rows in set (0.00 sec)

4. List all employees and their assigned project names, including those who are not assigned to any project.

```
SELECT e.Name AS EmployeeName, p.ProjectName
FROM Employee e
LEFT JOIN ProjectAssignment pa ON e.EmployeeID = pa.EmployeeID
LEFT JOIN Project p ON pa.ProjectID = p.ProjectID;
```

EmployeeName	ProjectName
Zayed Ikka	Website Redesign
Babu Battery	Mobile App Dev
Sahil GN Du	ERP System
Bob Ross	NULL

4 rows in set (0.00 sec)

5. Show all departments and the number of employees in each department, even if there are no employees.

```
SELECT d.DepartmentName, COUNT(e.EmployeeID) AS EmployeeCount
FROM Department d
LEFT JOIN Employee e ON d.DepartmentID = e.DepartmentID
GROUP BY d.DepartmentName;
```

DepartmentName	EmployeeCount
HR	2
IT	3
Finance	2
Sales	1
Marketing	0

5 rows in set (0.00 sec)

6. Retrieve all project names and the names of employees assigned to them, including projects without any assigned employees.

```
SELECT p.ProjectName, e.Name AS EmployeeName
FROM Project p
LEFT JOIN ProjectAssignment pa ON p.ProjectID = pa.ProjectID
LEFT JOIN Employee e ON pa.EmployeeID = e.EmployeeID;
```

ProjectName	EmployeeName
Website Redesign	Zayed Ikka
Mobile App Dev	Babu Battery
ERP System	Sahil GN Du
Marketing Ad	NULL

4 rows in set (0.00 sec)

## RESULT

Studied and implemented set operators, nested queries and Join queries.

## EXP NO: 10

Practice of SQL TCL commands like Rollback, Commit, Savepoint

AIM

Practice of SQL TCL commands like Rollback, Commit, Savepoint

### THEORY

TRANSACTIONAL CONTROL LANGUAGE (TCL): A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be made permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either a roll back or commit statement.

#### COMMIT

The basic syntax for using a COMMIT command in SQL is as follows:

BEGIN;

{a set of SQL statements};

COMMIT;

A more simplified version of syntax for other relational databases like MYSQL is as follows: {a set of SQL statements};

COMMIT;

#### SAVEPOINT & ROLLBACK:

Save points are like marks to divide a very lengthy transaction to smaller ones. They are used to identify a point in a transaction to which we can later roll back. Thus, save point is used in conjunction with roll back.

Syntax: SQL>SAVE POINT ID;

Example: SQL>SAVE POINT xyz;

ROLLBACK:

A roll back command is used to undo the current transactions. We can roll back the entire transaction so that all changes made by SQL statements are undone (or) roll back a transaction to a save point so that the SQL statements after the save point are rolled back.

Syntax:

ROLLBACK (current transaction can be rolled back)

ROLLBACK to save point ID;

## Commit

In MySQL, the statement SET autocommit=0; is used to disable the autocommit feature. By default, autocommit is enabled, which means that each SQL statement is automatically committed as a separate transaction.

\*/

```
SET autocommit=0;
```

/\*

QUESTION 1: Create a transaction that inserts a new employee record and updates the salary of an existing employee. Ensure that both operations are successful before committing the transaction."

```
BEGIN
INSERT INTO employees (employee_id, first_name, last_name, email,
    hire_date, designation, salary)
VALUES (1207, 'John', 'Wick', 'john.wlck@wb.com', SYSDATE,
    'IT_PROG', 60000);
1 row created.
UPDATE employees
SET salary = salary + 2000
WHERE employee_id = 1014;
1 row updated.
COMMIT;
Commit complete.
```

QUESTION 2: Within a transaction, create a savepoint after inserting a new project. Rollback to the savepoint if an error occurs during subsequent operations

```
BEGIN
INSERT INTO projects (project_id, project_name, start_date)
VALUES (301, 'AI Research', TO_DATE('2025-04-01', 'YYYY-MM-DD'));
1 row created.
SAVEPOINT before_task_assignment;
INSERT INTO project_tasks (task_id, project_id, task_name)
VALUES (501, 999, 'Data Analysis');
Error starting at line 1 in command:
INSERT INTO project_tasks (task_id, project_id, task_name)
VALUES (501, 999, 'Data Analysis')
Error report:
ORA-02291: integrity constraint (FK_PROJECT_ID) violated - parent key
not found
ROLLBACK TO before_task_assignment;
Rollback complete.
COMMIT;
Commit complete.
```

**Question 3: Start a transaction and within it, begin a nested transaction. Insert a new department and update the name of an existing department. Commit the nested transaction, and then either commit or rollback the outer transaction based on specific conditions**

```
BEGIN
INSERT INTO departments (department_id, department_name)
VALUES (501, 'Cybersecurity');
1 row created.
SAVEPOINT nested_txn;
UPDATE departments
SET department_name = 'AI & ML'
WHERE department_id = 201;
1 row updated.
COMMIT;
Commit complete.
```

```
DECLARE v_condition BOOLEAN := TRUE;
BEGIN
    IF v_condition THEN
        COMMIT;
    ELSE
        ROLLBACK;
    END IF;
END;
/
Commit complete.
```

**Question 4: Start a transaction and insert new records into multiple tables. Intentionally introduce an error, and if the error occurs, rollback the entire transaction.**

```
BEGIN
INSERT INTO employees (employee_id, first_name, last_name, salary,
    department_id)
VALUES (301, 'Alice', 'Smith', 60000, 501);
1 row created.
INSERT INTO projects (project_id, project_name, start_date)
VALUES (401, 'Quantum Security', '01-MAY-2025');
1 row created.
INSERT INTO project_tasks (task_id, project_id, task_name)
VALUES (601, 999, 'Encryption Implementation');
Error starting at line 1 in command:
INSERT INTO project_tasks (task_id, project_id, task_name)
VALUES (601, 999, 'Encryption Implementation')
Error report:
ORA-02291: integrity constraint (FK_PROJECT_ID) violated - parent
key not found
ROLLBACK;
Rollback complete.
```

**QUESTION 5:** Within a transaction, create multiple nested save points and perform operations. Rollback to a specific save point if an error occurs in the nested operations

```
BEGIN
INSERT INTO projects (project_id, project_name, start_date)
VALUES (701, 'IoT Security', TO_DATE('2025-06-01', 'YYYY-MM-DD'));
1 row created.
SAVEPOINT sp1;
INSERT INTO project_tasks (task_id, project_id, task_name)
VALUES (801, 701, 'Sensor Encryption');
1 row created.
SAVEPOINT sp2;
INSERT INTO project_resources (resource_id, project_id, resource_name)
VALUES (901, 701, 'AI Firewall');
1 row created.
INSERT INTO project_resources (resource_id, project_id, resource_name)
VALUES (902, 702, 'Quantum Key Distribution');
Error starting at line 1 in command:
INSERT INTO project_resources (resource_id, project_id, resource_name)
VALUES (902, 702, 'Quantum Key Distribution')
Error report:
ORA-02291: integrity constraint (FK_PROJECT_ID) violated - parent key
not found
ROLLBACK TO sp2;
Rollback complete.
COMMIT;
Commit complete.
```

## RESULT

Explored and applied SQL TCL commands such as COMMIT, ROLLBACK, and SAVEPOINT to effectively control transactions and maintain database consistency.

## EXP NO: 11

Practice of SQL commands for creation of views and assertions

AIM

Practice of SQL commands for creation of views and assertions

THEORY

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

```
CREATE VIEW view_name AS
SELECT column1, column2,...
FROM table_name
WHERE condition;
```

QUESTIONS

1. Create a view that includes essential details about each employee.

```
CREATE VIEW emp_details AS
SELECT employee_id, first_name, last_name, job_id, salary
FROM employees;
```

View created.

2. Create a view that displays information about project assignments for employees

```
CREATE VIEW project_assignments AS
SELECT e.employee_id, e.first_name, e.last_name, p.project_id,
p.project_name
FROM employees e
JOIN project_assignments pa ON e.employee_id = pa.employee_id
JOIN projects p ON pa.project_id = p.project_id;
```

View created.

3. Create a view showing employee details along with information about their dependents.

```
CREATE VIEW emp_dependents AS
SELECT e.employee_id, e.first_name, e.last_name, d.dependent_name,
d.relationship
FROM employees e
JOIN dependents d ON e.employee_id = d.employee_id;
```

View created.

4. Create a view that combines project details with the corresponding department information

```
CREATE VIEW project_department AS
SELECT p.project_id, p.project_name, d.department_id,
d.department_name
FROM projects p
JOIN departments d ON p.department_id = d.department_id;
```

View created.

5. Create a view that summarizes information about each department, including the total number of employees and total salary.

```
CREATE VIEW dept_summary AS
SELECT d.department_id, d.department_name, COUNT(e.employee_id) AS
total_employees, SUM(e.salary) AS total_salary
FROM departments d
LEFT JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name;
```

View created.

6. Check data in view Emp-Sal

```
SELECT * FROM emp_sal;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
1001	John	Doe	55000
1002	Jane	Smith	62000



### 7. Search view for a given employee

```
SELECT * FROM emp_details WHERE employee_id = 1001;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID	SALARY
1001	John	Doe	IT_PROG	55000

### 8. UPDATE Emp View

```
UPDATE emp_details  
SET salary = salary + 2000  
WHERE employee_id = 1001;
```

1 row updated.

### 9. Check the updated data in view

```
SELECT * FROM emp_details WHERE employee_id = 1001;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID	SALARY
1001	John	Doe	IT_PROG	57000

### 10. Check if the source table is updated or not

```
SELECT employee_id, salary FROM employees WHERE employee_id = 1001;
```

EMPLOYEE_ID	SALARY
1001	57000

### 11. Update Base table, and check if View is updated or not

```
UPDATE employees  
SET salary = salary - 1000  
WHERE employee_id = 1001;
```

1 row updated.

## 12. Check if View got changes

```
SELECT * FROM emp_details WHERE employee_id = 1001;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID	SALARY
1001	John	Doe	IT_PROG	56000

## 13. Delete View

```
DROP VIEW emp_details;
```

View dropped.

## 14. Drop Base Table

```
DROP TABLE employees;
```

Table dropped.

## 15. Verify deletion of base table will automatically delete view

```
SELECT * FROM emp_details;
```

ORA-04063: view "EMP\_DETAILS" has errors

## RESULT

SQL commands for creating views, performing updates, and verifying data consistency were successfully demonstrated.

## ANNEXURES

### PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.



