**DEPARTMENT OF SOFTWARE ENGINEERING**

Periyar Nagar, Vallam, Thanjavur, Tamil Nadu-613 403
Phone +91 – 4362 264600, Fax +91– 4362 264650
E mail- headmsc@pmu.edu

PERIYAR MANIAMMAI
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University)
Established Under Sec. 3 of UGC Act, 1956 • NAAC Accredited
think • innovate • transform

# Faculty of Computing Sciences and Engineering
# Department of Software Engineering

## B.Sc (Artificial Intelligence )

## Year : II

## Semester : III

## Academic Year : 2023-2024(Odd sem)

## Semester)

## Course Code : XAI303

## Course Name : DATA BASE MANAGEMENT SYSTEM

# LAB MANUAL

**DEPARTMENT OF SOFTWARE ENGINEERING**
Periyar Nagar, Vallam, Thanjavur, Tamil Nadu-613 403
Phone +91 – 4362 264600, Fax +91– 4362 264650
E mail- headmsc@pmu.edu

PERIYAR MANIAMMAI
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University)
Established Under Sec. 3 of UGC Act, 1956 • NAAC Accredited
think • innovate • transform

## Faculty of Computing Sciences and Engineering
## Department of Software Engineering

**B.Sc (Artificial Intelligence )**

**Year : II**

**Semester : III**

**Academic Year : (2023-2024 9Odd Semester)**

**Course Code : XAI303**

**Course Name : DATA BASE MANAGEMENT SYSTEM**

**Prepared by: Dr.D.Christy Sujatha , Assistant Professor (SG) /SE**

**COURSE INCHARGE**

**HOD SE** **DEAN/FCSE** **DEAN ACADEMIC(TLE)**

Our University is committed to the following Vision, Mission and core values, which guide us in carrying out our Software Engineering Department mission and realizing our vision:

| | |
|---|---|
| **INSTITUTION VISION** | |
| **To be a University of global dynamism with excellence in knowledge and innovation ensuring social responsibility for creating an egalitarian society.** | |
| **INSTITUTION MISSION** | |
| **UM1** | Offering well balanced programmes with scholarly faculty and state-of-art facilities to impart high level of knowledge. |
| **UM2** | Providing student - centered education and foster their growth in critical thinking, creativity, entrepreneurship, problem solving and collaborative work. |
| **UM3** | Involving progressive and meaningful research with concern for sustainable development. |
| **UM4** | Enabling the students to acquire the skills for global competencies. |
| **UM5** | Inculcating Universal values, Self respect, Gender equality, Dignity and Ethics. |
| **INSTITUTION CORE VALUES** | |

- **Student – centric vocation**
- **Academic excellence**
- **Social Justice, equity, equality, diversity, empowerment, sustainability**
- **Skills and use of technology for global competency.**
- **Continual improvement**
- **Leadership qualities.**
- **Societal needs**
- **Learning, a life – long process**
- **Team work**
- **Entrepreneurship for men and women**
- **Rural development**
- **Basic, Societal, and applied research on Energy, Environment, and Empowerment.**

## DEPARTMENT OF SOFTWARE ENGINEERING

| **DEPARTMENT VISION** | |
|---|---|
| **To be a leading department in the field of software development and digital design that offers the software education with the State-of-the-art skills. The Graduates will be recognized as globally competent by their dynamic work and produce valuable digital solutions for the society.** | |
| **DEPARTMENT MISSION** | |
| **DM1** | To construct the software related technical skills among the students. |
| **DM2** | To practice the cutting-edge technologies in the various areas of digital design and software development. |
| **DM3** | To contribute towards the betterment of the society by producing enhanced software solutions through research. |
| **DM4** | To generate the spirit of inquiry, team work, novelty and professionalism among the students. |

### Mapping of University Mission (UM) and Department Mission (DM)

| | **DM1** | **DM2** | **DM3** | **DM4** | **Total** |
|---|---|---|---|---|---|
| **UM1** | 2 | 3 | 1 | 0 | **6** |
| **UM2** | 1 | 2 | 0 | 2 | **5** |
| **UM3** | 1 | 1 | 3 | 0 | **5** |
| **UM4** | 3 | 1 | 1 | 1 | **6** |
| **UM5** | 0 | 0 | 2 | 3 | **5** |

**1-Low 2- Medium 3 – High**

## PROGRAMME EDUCATIONAL OBJECTIVES (PEO)

Based on the mission of the department, the programme educational objective is formulated as The B.Sc. Computer Science dedicated to produce graduates who have ability to

| | |
|---|---|
| **PEO1** | **evolve as globally proficient computer professionals by giving enriched performance in problem solving, analysis and synthesis for computer science related issues..** |
| **PEO2** | exercise with contemporary tools and technologies to provide an effective user friendly interface for the real time social concerns. |
| **PEO3** | communicate effectively in a multidisciplinary team and manage the team members through the acquired leadership skills to achieve the target in time. |
| **PEO4** | handle the customers and stakeholders effectively with the awareness of human values and ethical concerns. |
| **PEO5** | pursue lifelong learning through the cutting-edge Learning Management Systems and thus satisfy the up-to-date industry expectations. |

### Mapping of Program Educational Objectives (PEOs) with Department Mission (DM)

| B.Sc. (CS) | PEO1 | PEO2 | PEO3 | PEO4 | Total |
|---|---|---|---|---|---|
| **DM1** | 3 | 2 | 0 | 0 | **5** |
| **DM2** | 2 | 2 | 1 | 1 | **6** |
| **DM3** | 1 | 2 | 1 | 1 | **5** |
| **DM4** | 0 | 1 | 3 | 1 | **5** |

***1- Low 2 – Medium 3-High***

### PROGRAMME OUTCOME (PO)

At the time of graduation, competency of the student is measured through the attainment of programme outcomes. The quantification of programme outcomes attainment is measured through the assessment of established course outcomes for each course. Graduates of the B.Sc. Computer Science programme will have attained the ability to

| PROGRAMME OUTCOMES | |
|---|---|
| **PO 1** | identify and analyze the acquainted or unacquainted real time issues and afford solution using the necessary computing, mathematical and basic science skill set. |
| **PO 2** | design and develop algorithms for providing an appropriate solution to gratify the industrial and social needs. |
| **PO 3** | express ideas and thoughts effectively to the team members and customers through written and oral communication. |
| **PO 4** | work jointly with different team members in order to complete the agreed work in time. |
| **PO 5** | inspire and guide the team members using management skills to achieve the target in an efficient and smooth way. |
| **PO 6** | provide a remarkable impact on the society by contributing resolutions to social issues with the awareness of ethical responsibility by discriminating ethical &unethical behaviors and understanding human, professional values& responsibilities. |
| **PO 7** | utilize computer literacy in the learning and working places and self-adapt with the changing environment by participating in learning activities throughout the life. |
| **PROGRAMMME SPECIFIC OUTCOME** | |
| **PSO1** | provide the professional user friendly interface with the help of state-of-the-art tools and technologies. |
| **PSO2** | design the interactive& responsive web based and mobile applications. |

**Table : 3 Mapping of Program Educational Objectives (PEOs) with Program Outcomes (POs)**

| B.Sc. CS | PO | | | | | | | PSO | | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 1 | 2 | |
| PEO1 | 1 | 3 | 2 | 1 | 1 | 0 | 0 | 3 | 1 | 13 |
| PEO2 | 0 | 2 | 2 | 3 | 1 | 1 | 0 | 2 | 1 | 13 |
| PEO3 | 1 | 1 | 0 | 1 | 0 | 3 | 3 | 1 | 1 | 12 |
| PEO4 | 0 | 1 | 1 | 2 | 2 | 1 | 0 | 1 | 3 | 13 |

**1 - Low 2 – Medium 3 - High**

## Mapping of Program Outcomes (POs) with Graduate Attributes (GAs)

| SI.No | Graduate Attributes | PO | | | | | | | PSO | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 1 | 2 |
| A1 | Subject Specialist | 3 | | | | | | | 3 | 3 |
| A2 | Problem analysis | 3 | | | | | | | 3 | 3 |
| A3 | Design/Development of solutions | | 3 | | | | | | 3 | 3 |
| A4 | Conduct investigations of complex problems | | | 3 | | | | | 3 | 3 |
| A5 | Modern tool usage | | | | 3 | | | | 3 | 3 |
| A6 | Environment and Sustainability | | | | | 3 | | | 1 | 1 |
| A7 | Ethics and Social Responsibility | | 1 | | | 2 | 2 | | 1 | 1 |
| A8 | Effective Communication | | | | | | | 3 | 1 | 1 |
| A9 | Individual and Team Work | | | | | | 3 | | 2 | 2 |
| A10 | Life-long learning | | | | | | | 3 | 2 | 2 |

**1- Slightly** **2 – Supportive** **3 - Highly related**

<table>
<tr><td colspan="3" rowspan="2"><b>XBC402</b></td><td rowspan="4"><b>DATA BASE MANAGEMENT SYSTEM</b></td><td><b>L</b></td><td><b>T</b></td><td><b>P</b></td><td><b>SS</b></td><td><b>C</b></td></tr>
<tr><td>3</td><td>1</td><td>1</td><td>1</td><td><b>6</b></td></tr>
<tr><td><b>C</b></td><td><b>P</b></td><td><b>A</b></td><td><b>L</b></td><td><b>T</b></td><td><b>P</b></td><td><b>SS</b></td><td><b>H</b></td></tr>
<tr><td><b>3</b></td><td><b>1</b></td><td><b>0</b></td><td>3</td><td>1</td><td>3</td><td>1</td><td><b>8</b></td></tr>
</table>

**PREREQUISITE:** Computer Fundamentals

| **Course Outcomes** | | **Domain** | **Level** |
|---|---|---|---|
| After the completion of the course, students will be able to | | | |
| **CO1** | Explain Basic Data Base concepts and Data Base models | Cognitive | Understand |
| **CO2** | Apply and Display SQL Queries and Relational Algebra | Cognitive<br>Psychomotor | Applying<br>Set |
| **CO3** | Explain and Describe File Organization and Concurrency | Cognitive<br>Psychomotor | Understanding<br>Perception |
| **CO4** | Comprehend Big Data concepts and State Implementation techniques. | Cognitive<br>Psychomotor | Understanding<br>Set |
| **CO5** | Analyze and Identify Data Base Programming Languages | Cognitive<br>Psychomotor | Analyzing<br>Perception |

| **UNIT I** | **INTRODUCTION** | **9+3+9** |
|---|---|---|

Basic Database Concepts, Terminology, and Architecture; Types of Database Management Systems. Differences between Relational and other Database Models. Data Modelling: Relations, Schemas, Constraints, Queries, and Updates; Conceptual vs. Physical Modeling; Entity Types, attributes, ER Diagrams.

**Lab:**

**1: E-R Model**

Analyze the organization and identify the entities, attributes and relationships in it. .

Identify the primary keys for all the entities. Identify the other keys like candidate keys, partial keys, if any.

**2: Concept design with E-R Model**

Relate the entities appropriately. Apply cardinalities for each relationship. Identify strong entities and weak entities (if any).

| **UNIT II** | **RELATIONAL DATABASES** | **9+3+9** |
|---|---|---|

SQL Data Definition: Specifying Tables, Data Types, Constraints; Simple SELECT, INSERT, UPDATE, DELETE Statements; Complex SELECT Queries, including Joins and Nested Queries; Actions and Triggers; Views; Altering Schemas. Relational Algebra: Definition of Algebra; Relations as Sets; Operations: SELECT, PROJECT, JOIN, etc. Normalization Theory and Functional Dependencies, 2NF, 3NF, BCNF, 4NF, 5NF.

**Lab:**

**3: Relational Model**

Represent all the entities (Strong, Weak) in tabular fashion. Represent relationships in a tabular fashion.

**4: Normalization**

Apply the First, Second and Third Normalization levels on the database designed for the organization

**UNIT III | DATABASE DESIGN | 9+3+9**

Indexing: Files, Blocks, and Records, Hashing; RAID; Replication; Single-Level and Multi-Level Indexes; B-Trees and B+-Trees. Query Processing Translation of SQL into Query Plans; Basics of Transactions, Concurrency and Recovery.

**Lab:**

**5: Installation of Mysql and practicing DDL commands**

Installation of MySql. Creating databases, how to create tables, altering the database, dropping tables and databases if not required. Try truncate, rename commands etc.

**6: Practicing DML commands on the Database created for the example organization**

DML commands are used to for managing data within schema objects. Some examples:

- SELECT - retrieve data from a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain

**UNIT IV | DATABASE PROGRAMMING | 9+3+9**

DATABASE PROGRAMMING: Embedded SQL; Dynamic SQL, JDBC; Avoiding Injection Attacks; Stored Procedures; Lightweight Data Access Layers for Python and JavaScript Applications; PHP and MySQL, Object Relational Modeling: Hibernate for Java, Active Record for Rails.

**Lab:**

**7: Querying**

practice queries (along with sub queries) involving ANY, ALL, IN, Exists, NOT EXISTS, UNION, INTERSECT, Constraints etc.

**8 and 9: Querying (continued…)**

Practice queries using Aggregate functions (COUNT, SUM, AVG, and MAX and MIN), GROUP BY, HAVING and Creation and dropping of Views.

**UNIT V | IMPLEMENTATION TECHNIQUES | 9+3+9**

BIG DATA: Motivations; OLAP vs. OLTP; Batch Processing; MapReduce and Hadoop; Spark; Other Systems: HBase. Working with POSTGRES, REDIS, MONGO, and NEO: Setting up the same Database on Four Platforms; Basic Queries and Reporting.

**Lab:**

**10: Triggers**

Work on Triggers. Creation of, insert trigger, delete trigger, update trigger. Practice triggers using the above database

| LECTURE | TUTORIAL | PRACTICAL | SELF-STUDY | TOTAL |
|---|---|---|---|---|
| 45 | 15 | 45 | 15 | 105+15 |

**EFERENCES:**

1. Abraham Silberschatz, Henry F. Korth, S. Sudharshan, 2011"Database System Concepts", Sixth Edition, Tata McGraw Hill.
2. RamezElmasri, Shamkant B. Navathe., 2008. "Fundamentals of Database Systems", Fifth Edition, Pearson.
3. Raghu Ramakrishnan., 2010. "Database Management Systems", Fourth Edition, Tata McGraw Hill.
4. G.K.Gupta, 2011."Database Management Systems", Tata McGraw Hill.

**Table 1: Mapping of Course Outcomes (CO) with Programme Outcomes (PO):**

| B.Sc CS | PO | | | | | | | PSO | |
|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **1** | **2** |
| **CO1** | 0 | 1 | 2 | 0 | 1 | 0 | 0 | 3 | 3 |
| **CO2** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| **CO3** | 1 | 3 | 1 | 1 | 1 | 0 | 0 | 3 | 3 |
| **CO4** | 1 | 3 | 2 | 1 | 1 | 1 | 1 | 3 | 3 |
| **CO5** | 3 | 3 | 2 | 2 | 1 | 1 | 1 | 3 | 2 |
| **Average** | 1 | 2 | 2 | 1 | 1 | 0 | 0 | 3 | 2 |

3–High Relation, 2–Medium Relation, 1–Low Relation, 0–No Relation

# XAI303 : Data Base Management System

## Index

# EX.No.1 ENTITY RELATIONSHIP MODEL FOR TICKET BOOKING IN RAILWAY RESERVATION SYSTEM USING MS ACCESS

## AIM

To analyze the problem with the entities which identify data persisted in the database which contains entities and attributes for Railway Reservation System.

## Objectives:

Student will able to learn the Entity-Relationship(ER) modeling to develop a conceptual model of data.

## Outcomes:

Student gains the ability

- To Construct E-R diagrams with

  Entities(strong, weak, associative)

  Attributes (simple, multi-valued,derived, composite attribute)

  Relations (unary, binary,ternary)

## Railway Reservation System

**Requirements Analysis Roadway**

**Reservations**

Reservations are directly handled by booking office. Reservations can be made 30 days in advance and tickets issued to passenger. One passenger/ person can book many tickets (to his/her family). In the process of Computerization of Roadway Travels we have to design and developa Database which consists the data of Train, Passengers, Tickets .

**Following steps are involved in the process:**

1. Analyzing the problem and identifying the Entities and Relationships
2. E-R Diagram
3. Creating database and ticket booking form Using MS Access

## 1. Analyzing the problem and identifying the Entities and Relationships

Analyze the problem carefully and come up with the entities in it. Identify what data has to be persisted in the database. This contains the entities, attributes etc. In this we wqill analyze different types of entities with attributes of “Roadways Travels”.

**Entity:** An Entity is an object or concept about which we want to store information

**Relationship:** A relationship is an association between several entities.

**Attributes:** An object is characterized by its properties or attributes. In relational database system attributes correspond to fields.

**The Railway Reservation System Consists Of The Following Entities:**

- ✓ Train
- ✓ Ticket
- ✓ Passenger

These Entities have the following attributes

**Train**

- ➢ Train_id
- ➢ Train _name
- ➢ From _Place
- ➢ To_place
- ➢ Number of Seats
- ➢ Time
- ➢ Date

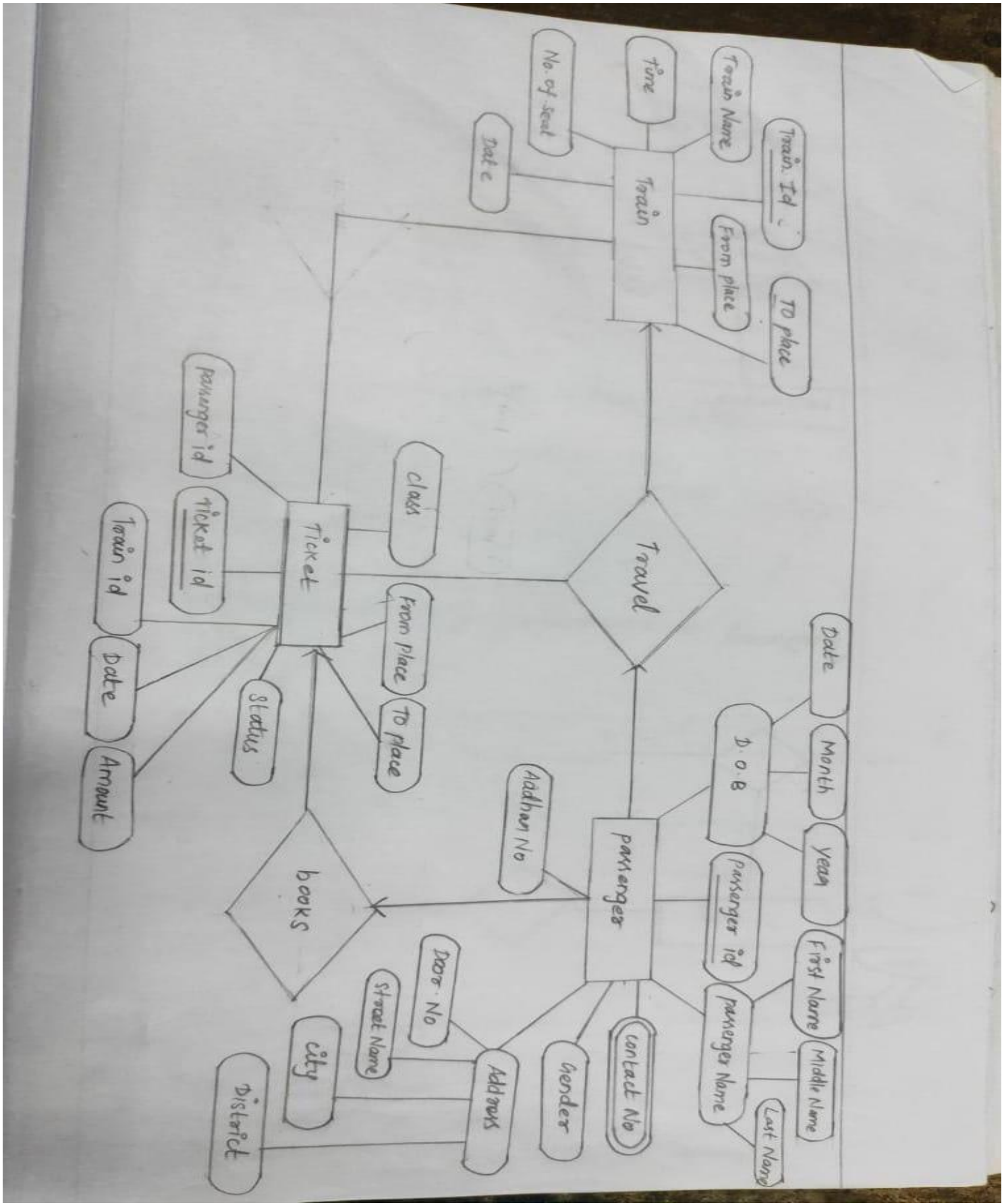
**Ticket:**

- ➢ Ticket_id
- ➢ Passenger_ID
- ➢ Train_ID
- ➢ Class
- ➢ Seat_No
- ➢ Date
- ➢ From _Place
- ➢ To_Place
- ➢ Amount
- ➢ Status

**Passenger:**

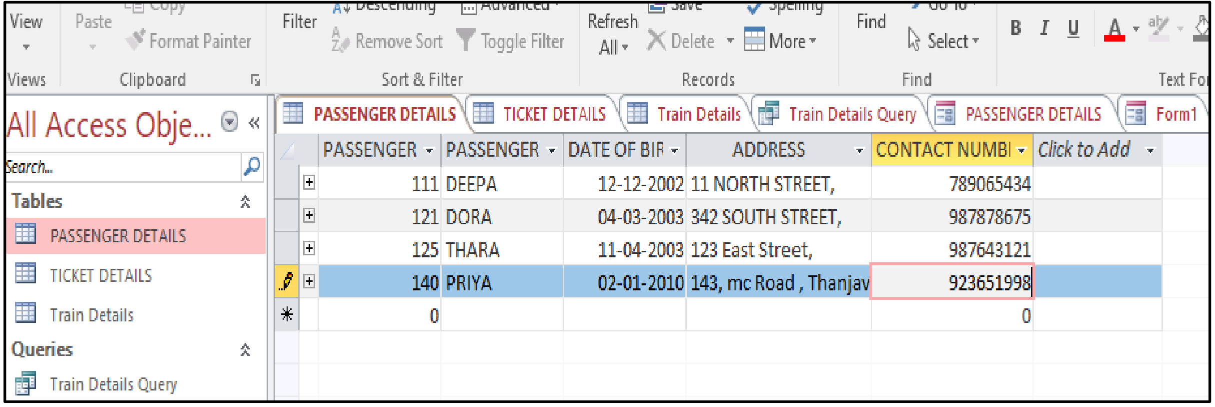
- Passenger_id
- Passenger_Name
- Gender
- Address
- Aadhar Number
- Contact Number
- DOB

**2.E R diagram:**


Train
Train Id
Train Name
Time
No. of seat
Date
From place
TO place
Travel
Ticket
class
passenger id
ticket id
Train id
Date
Amount
Status
From place
TO place
books
passenger
Aadhar No
D.O.B
Date
Month
year
passenger id
First Name
Middle Name
Last Name
passenger Name
contact No
Gender
Address
Door No
Street Name
city
District

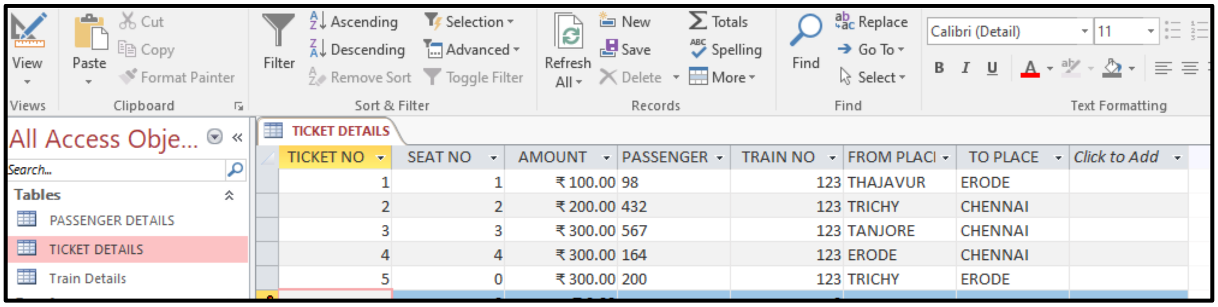
## 3.Data Base creation  and Ticket booking form  Using MS Access

### Passenger Details:



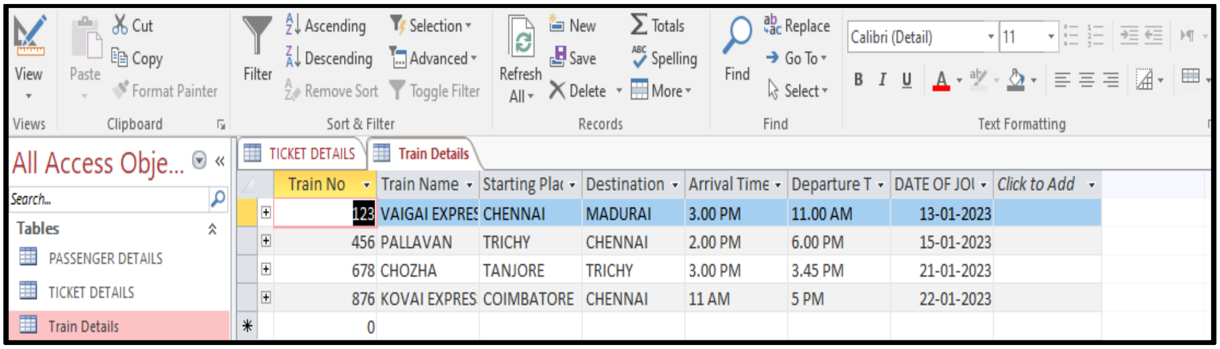
| PASSENGER | PASSENGER | DATE OF BIR | ADDRESS | CONTACT NUMBI | Click to Add |
|---|---|---|---|---|---|
| 111 | DEEPA | 12-12-2002 | 11 NORTH STREET, | 789065434 | |
| 121 | DORA | 04-03-2003 | 342 SOUTH STREET, | 987878675 | |
| 125 | THARA | 11-04-2003 | 123 East Street, | 987643121 | |
| 140 | PRIYA | 02-01-2010 | 143, mc Road , Thanjav | 923651998 | |
| 0 | | | | 0 | |

### Ticket Details:

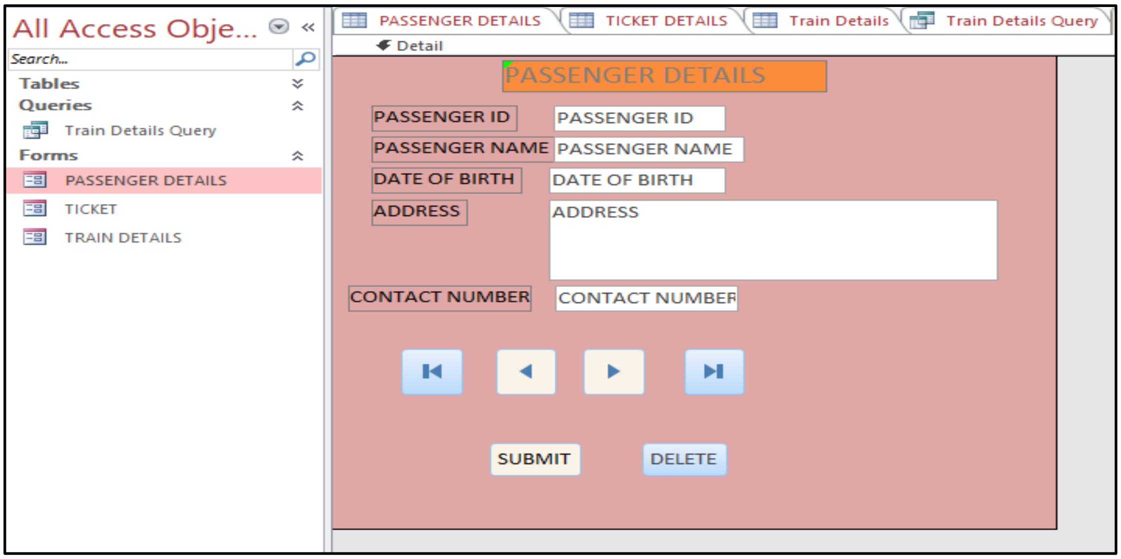


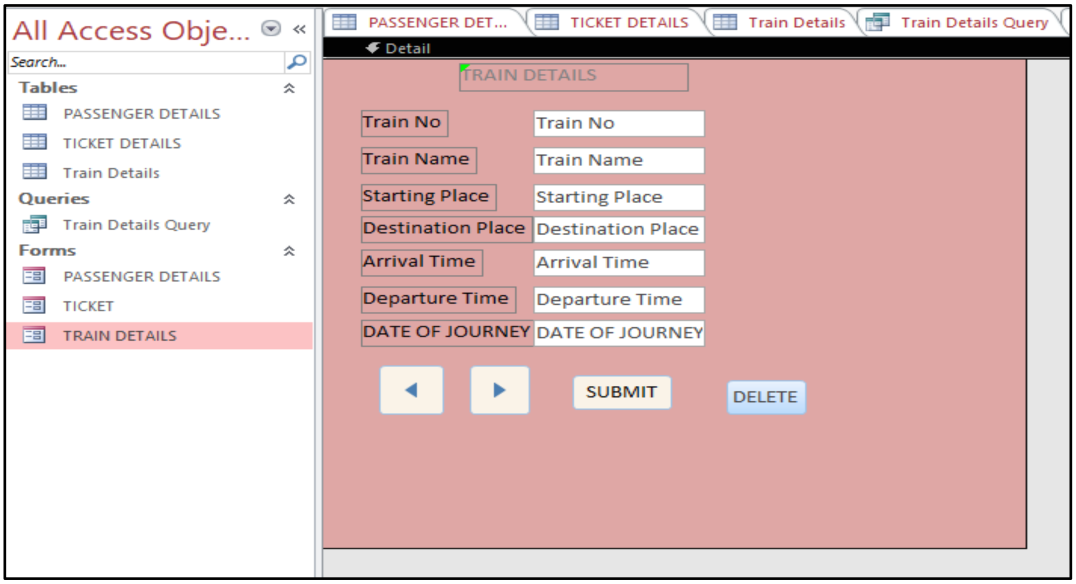
| TICKET NO | SEAT NO | AMOUNT | PASSENGER | TRAIN NO | FROM PLACI | TO PLACE | Click to Add |
|---|---|---|---|---|---|---|---|
| 1 | 1 | ₹ 100.00 | 98 | 123 | THAJAVUR | ERODE | |
| 2 | 2 | ₹ 200.00 | 432 | 123 | TRICHY | CHENNAI | |
| 3 | 3 | ₹ 300.00 | 567 | 123 | TANJORE | CHENNAI | |
| 4 | 4 | ₹ 300.00 | 164 | 123 | ERODE | CHENNAI | |
| 5 | 0 | ₹ 300.00 | 200 | 123 | TRICHY | ERODE | |

### Train Details :



| Train No | Train Name | Starting Plac | Destination | Arrival Time | Departure T | DATE OF JOI | Click to Add |
|---|---|---|---|---|---|---|---|
| 123 | VAIGAI EXPRES | CHENNAI | MADURAI | 3.00 PM | 11.00 AM | 13-01-2023 | |
| 456 | PALLAVAN | TRICHY | CHENNAI | 2.00 PM | 6.00 PM | 15-01-2023 | |
| 678 | CHOZHA | TANJORE | TRICHY | 3.00 PM | 3.45 PM | 21-01-2023 | |
| 876 | KOVAI EXPRES | COIMBATORE | CHENNAI | 11 AM | 5 PM | 22-01-2023 | |
| 0 | | | | | | | |





**Result :** Thus ER Diagram has been  designed, Data Base has been created and forms have been created to enter data using MS Access

# Ex No 2 Data Definition Language (DDL Commands)

**Aim** : To implement DDL Commands using oracle 10 g.

**Outcome :**

Student will able to learn DDL Statements to Create and Manage Tables.

**SQL -DDL Commands:**

**CREATE TABLE**

It is used to create a table.

**Syntax:**

Create table table name (column_name1 data_ type constraints, column_name2 data_ type constraints …)

**Example:**

Create table Emp ( EmpNo number(5), EName VarChar(15), Job Char(10), DeptNo number(3));

Create table stud (sname varchar2(20) not null, roll 9no number(10) not null, dob date not null);

**Rules:**

1. Oracle reserved words cannot be used.
2. Underscore, numerals, letters are allowed but not blank space.
3. Maximum length for the table name is 30 characters.
4. 2 different tables should not have same name.
5. We should specify a unique column name.
6. We should specify proper data type along with width.
7. We can include "not null" condition when needed. By default it is 'null'.

**ALTER TABLE**

Alter command is used to:

1. Add a new column.
2. Modify the existing column definition.

3. To include or drop integrity constraint.

**Syntax:** alter table tablename add/modify (attribute datatype(size));

**Example:**

1. Alter table emp add (phone_no char (20));
2. Alter table emp modify(phone_no number (10));
3. ALTER TABLE EMP ADD CONSTRAINT Pkey1 PRIMARY KEY (EmpNo);

**DROP TABLE**

It will delete the table structure provided the table should be empty.

**Example:** drop table prog20;

Here prog20 is table name

**TRUNCATE TABLE**

If there is no further use of records stored in a table and the structure has to be retained then the records alone can be deleted.

**Syntax:** TRUNCATE TABLE <TABLE NAME>

Example: Truncate table stud;

**DESC**

This is used to view the structure of the table.

**Example:** desc emp;

```
Name Null? Type
-------------------------------- --------
EmpNo NOT NULL   Number(5)
EName VarChar(15)
Job NOT NULL Char(10)
DeptNo NOTNULL number(3)
PHONE_NO number (10)
```

**DOMAIN INTEGRITY**

**Example:** Create table cust(custid number(6) not null, name char(10));

Alter table cust modify (name not null);

**CHECK CONSTRAINT**

**Example:** Create table student (regno number (6), mark number (3) constraint b check mark >=0 and mark <=100));

Alter table student add++------------------------------------ constraint b2 check (length(regno<=4));

**ENTITY INTEGRITY**

**a) Unique key constraint**

**Example:** Create table cust(custid number(6) constraint uni unique, name char(10));

Alter table cust add(constraint c unique(custid));

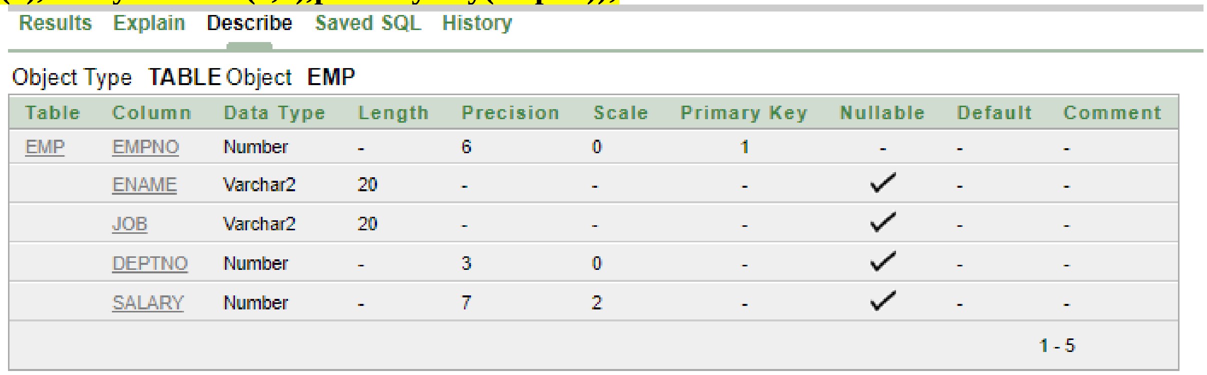
b) **Primary Key Constraint**

**Example:** Create table stud(regno number(6) constraint primary key, name char(20));

**Queries:**

**Q1: Create a table called emp with the following structure**

**create table emp(empno number(6),ename varchar(20),job varchar(20),deptno number (3),salary number(7,2),primary key(empno));**



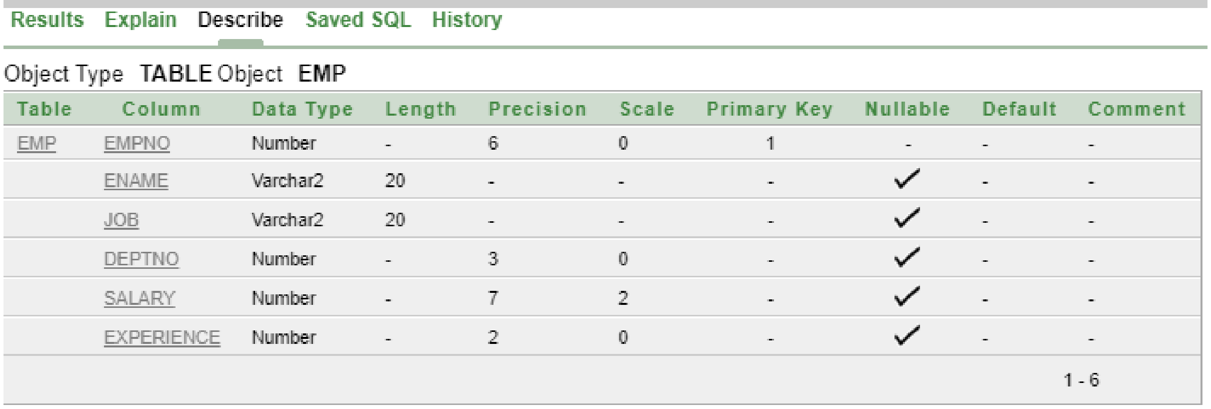
Object Type TABLE Object EMP

| Table | Column | Data Type | Length | Precision | Scale | Primary Key | Nullable | Default | Comment |
|---|---|---|---|---|---|---|---|---|---|
| EMP | EMPNO | Number | - | 6 | 0 | 1 | - | - | - |
| | ENAME | Varchar2 | 20 | - | - | - | ✓ | - | - |
| | JOB | Varchar2 | 20 | - | - | - | ✓ | - | - |
| | DEPTNO | Number | - | 3 | 0 | - | ✓ | - | - |
| | SALARY | Number | - | 7 | 2 | - | ✓ | - | - |

1 - 5

**Q2: Add a column experience to the emp table,experience numeric null allowed**

**alter table emp add(experience number(2) not null);**



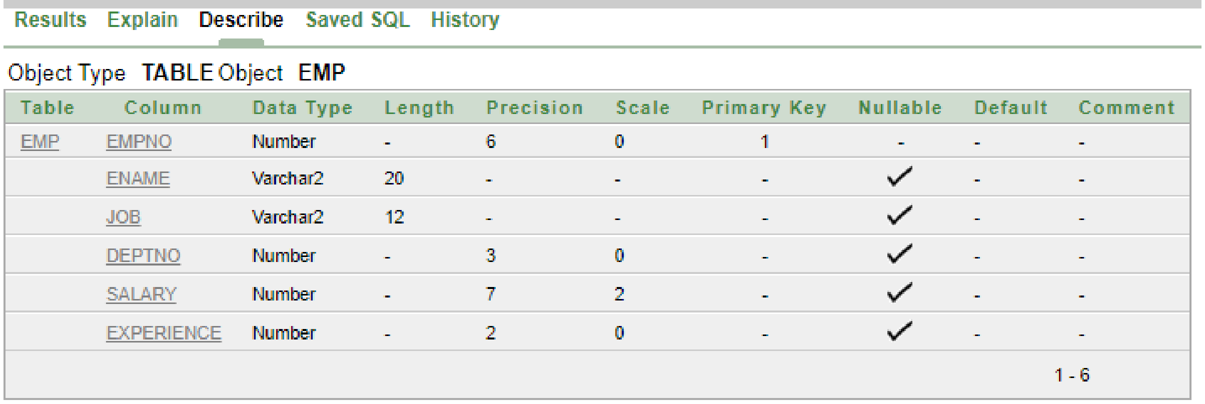
Object Type TABLE Object EMP

| Table | Column | Data Type | Length | Precision | Scale | Primary Key | Nullable | Default | Comment |
|---|---|---|---|---|---|---|---|---|---|
| EMP | EMPNO | Number | - | 6 | 0 | 1 | - | - | - |
| | ENAME | Varchar2 | 20 | - | - | - | ✓ | - | - |
| | JOB | Varchar2 | 20 | - | - | - | ✓ | - | - |
| | DEPTNO | Number | - | 3 | 0 | - | ✓ | - | - |
| | SALARY | Number | - | 7 | 2 | - | ✓ | - | - |
| | EXPERIENCE | Number | - | 2 | 0 | - | ✓ | - | - |

1 - 6

**Q3: Modify the column width of the job field of emp table**

**alter table emp modify(job varchar(12));**



Object Type TABLE Object EMP

| Table | Column | Data Type | Length | Precision | Scale | Primary Key | Nullable | Default | Comment |
|---|---|---|---|---|---|---|---|---|---|
| EMP | EMPNO | Number | - | 6 | 0 | 1 | - | - | - |
| | ENAME | Varchar2 | 20 | - | - | - | ✓ | - | - |
| | JOB | Varchar2 | 12 | - | - | - | ✓ | - | - |
| | DEPTNO | Number | - | 3 | 0 | - | ✓ | - | - |
| | SALARY | Number | - | 7 | 2 | - | ✓ | - | - |
| | EXPERIENCE | Number | - | 2 | 0 | - | ✓ | - | - |

1 - 6

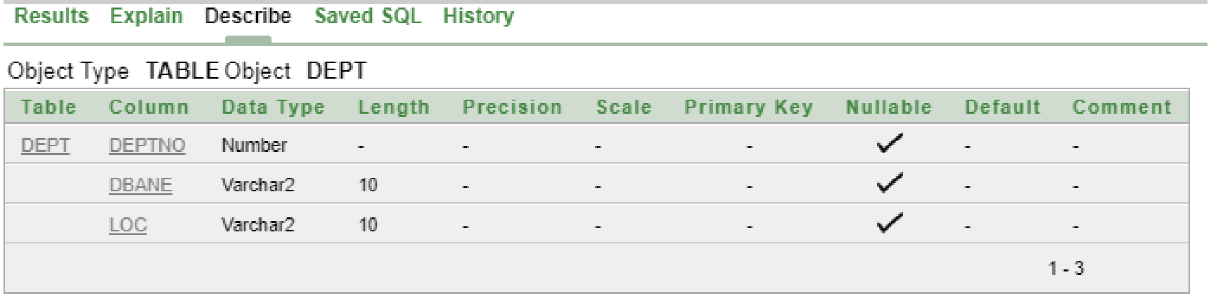
**Q4: Create dept table with following structure:**

**Name** **Type**

| | |
|---|---|
| **DEPTNO** | **Number(2)** |
| **DName** | **Varchar(10)** |
| **Loc** | **Varchar(10)** |

**DEPTNO as the primary key**

```
create table dept(deptno number,dname varchar(10),loc varchar(10));
```



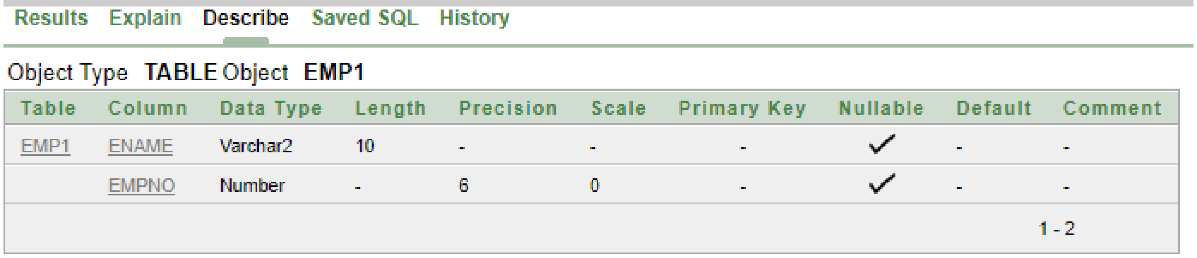
Object Type TABLE Object DEPT

| Table | Column | Data Type | Length | Precision | Scale | Primary Key | Nullable | Default | Comment |
|---|---|---|---|---|---|---|---|---|---|
| DEPT | DEPTNO | Number | - | - | - | - | ✓ | - | - |
| | DBANE | Varchar2 | 10 | - | - | - | ✓ | - | - |
| | LOC | Varchar2 | 10 | - | - | - | ✓ | - | - |

1 - 3

**Q5: Create the emp1 table with ename and empno, add constraints to check the empno value while entering (ie)empno>100**

```
create table emp1(ename varchar(10),empno number(6) constraint ch check(empno>100));
```



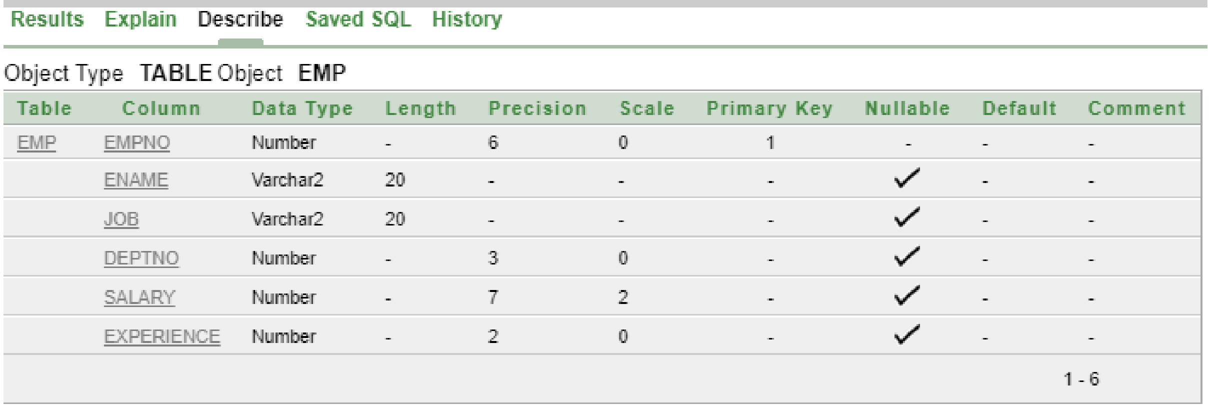
Object Type TABLE Object EMP1

| Table | Column | Data Type | Length | Precision | Scale | Primary Key | Nullable | Default | Comment |
|---|---|---|---|---|---|---|---|---|---|
| EMP1 | ENAME | Varchar2 | 10 | - | - | - | ✓ | - | - |
| | EMPNO | Number | - | 6 | 0 | - | ✓ | - | - |

1 - 2

**Q6: Drop a column experience to the emp table**

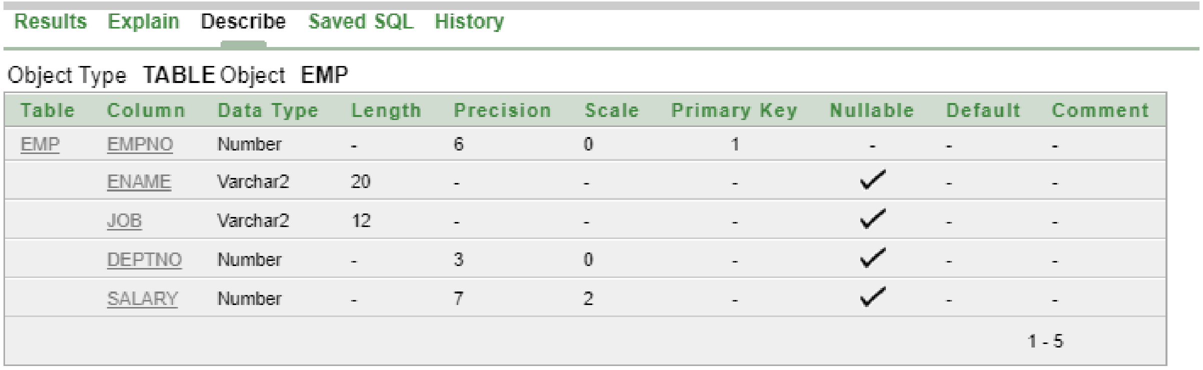
```
alter table emp drop column experience;
```



Object Type TABLE Object EMP

| Table | Column | Data Type | Length | Precision | Scale | Primary Key | Nullable | Default | Comment |
|---|---|---|---|---|---|---|---|---|---|
| EMP | EMPNO | Number | - | 6 | 0 | 1 | - | - | - |
| | ENAME | Varchar2 | 20 | - | - | - | ✓ | - | - |
| | JOB | Varchar2 | 20 | - | - | - | ✓ | - | - |
| | DEPTNO | Number | - | 3 | 0 | - | ✓ | - | - |
| | SALARY | Number | - | 7 | 2 | - | ✓ | - | - |
| | EXPERIENCE | Number | - | 2 | 0 | - | ✓ | - | - |

1 - 6



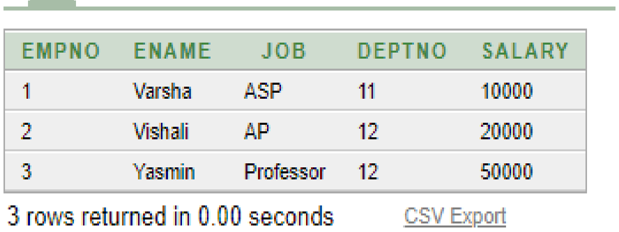
Object Type TABLE Object EMP

| Table | Column | Data Type | Length | Precision | Scale | Primary Key | Nullable | Default | Comment |
|---|---|---|---|---|---|---|---|---|---|
| EMP | EMPNO | Number | - | 6 | 0 | 1 | - | - | - |
|  | ENAME | Varchar2 | 20 | - | - | - | ✓ | - | - |
|  | JOB | Varchar2 | 12 | - | - | - | ✓ | - | - |
|  | DEPTNO | Number | - | 3 | 0 | - | ✓ | - | - |
|  | SALARY | Number | - | 7 | 2 | - | ✓ | - | - |

1 - 5

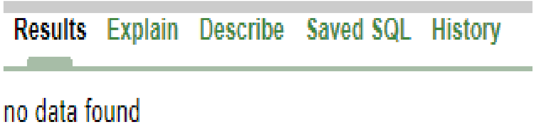
## Q7: Truncate the emp table and drop the dept table

**truncate table emp;**



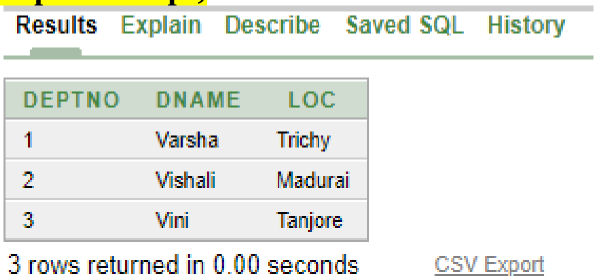
| EMPNO | ENAME | JOB | DEPTNO | SALARY |
|---|---|---|---|---|
| 1 | Varsha | ASP | 11 | 10000 |
| 2 | Vishali | AP | 12 | 20000 |
| 3 | Yasmin | Professor | 12 | 50000 |

3 rows returned in 0.00 seconds



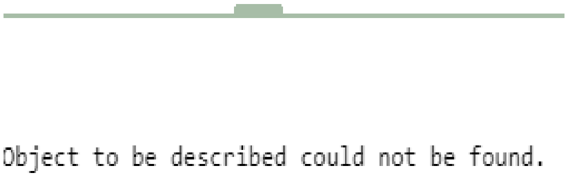
no data found

**drop table dept;**



| DEPTNO | DNAME | LOC |
|---|---|---|
| 1 | Varsha | Trichy |
| 2 | Vishali | Madurai |
| 3 | Vini | Tanjore |

3 rows returned in 0.00 seconds



```
Object to be described could not be found.
```

**Result:** Thus the data definition language commands was performed and implemented successfully

## EX.NO.3 Data Manipulation Language (DML Commands )

**Aim :** To implement DML Commands using Oracle 10g

**Objectives:**

Student will able to learn commands that make changes in relational database and transaction management.

**Outcomes:**

Student gains the knowledge to perform transactions like updating, deleting, inserting and selecting datafrom a data base.

**SQL Commands:**

**INSERT COMMAND**

**Inserting a single row into a table:**

**Syntax:** insert into <table name> values (value list)

**Example:** insert into s values('s3','sup3','blore',10)

**Inserting more than one record using a single insert commands: Syntax:** insert into <table name> values (&col1, &col2, ….) **Example:** Insert into stud values(&reg, '&name', &percentage);

**Skipping the fields while inserting:**

Insert into <tablename(coln names to which datas to b inserted)> values (list of values);

Other way is to give null while passing the values.

**SELECT COMMANDS**

**Selects all rows from the table**

**Syntax:** Select * from tablename;

**Example;** Select * from IT;

**The retrieval of specific columns from a table:**

It retrieves the specified columns from the table

**Syntax:** Select column_name1, …..,column_namen from table name;

**Example:** Select empno, empname from emp;

**Elimination of duplicates from the select clause:**

It prevents retrieving the duplicated values .Distinct keyword is to be used.

**Syntax:** Select DISTINCT col1, col2 from table name;

**Example:** Select DISTINCT job from emp;

**Select command with where clause:**

To select specific rows from a table we include 'where' clause in the select command. It can appear only after the 'from' clause.

**Syntax:** Select column_name1, …..,column_namen from table name where condition;

**Example:** Select empno, empname from emp where sal>4000;

**Select command with order by clause:**

**Syntax:** Select column_name1, …..,column_namen from table name where condition order by colmnname;

**Example:** Select empno, empname from emp order by empno;

**Select command to create a table:**

**Syntax:** create table tablename as select * from existing_tablename;

**Example:** create table emp1 as select * from emp;

**Select command to insert records:**

**Syntax:** insert into tablename ( select columns from existing_tablename);

**Example:** insert into emp1 ( select * from emp);

## UPDATE COMMAND

**Syntax:**update tablename set field=values where condition;

**Example:**Update emp set sal = 10000 where empno=135;

## DELETE COMMAND

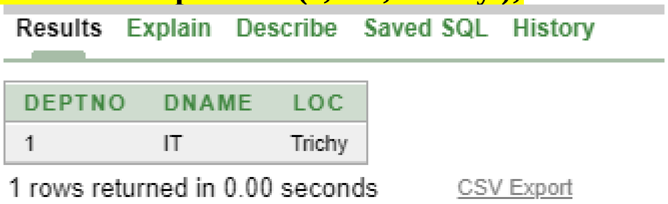
**Syntax:** Delete from table where conditions;

**Example:** delete from emp where empno=135;

## Queries:

**Q1: Insert a single record into dept table**

**insert into dept values(1,'IT','Trichy');**

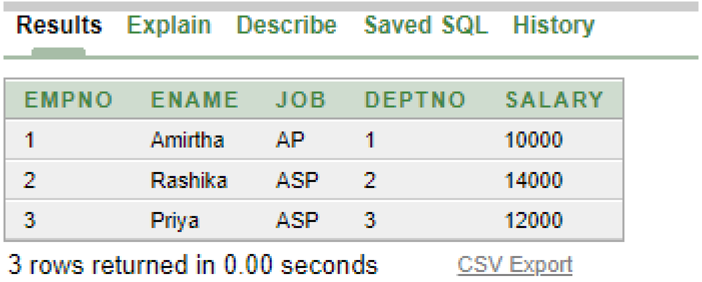


| DEPTNO | DNAME | LOC |
|---|---|---|
| 1 | IT | Trichy |

1 rows returned in 0.00 seconds

**Q2: Insert more than a record into emp table using a single insert command**

```
insert into emp values(1,'Amirtha','AP',1,10000);
insert into emp values(2,'Rashika','ASP',2,14000);
insert into emp values(3,'Priya','ASP',3,12000);
```

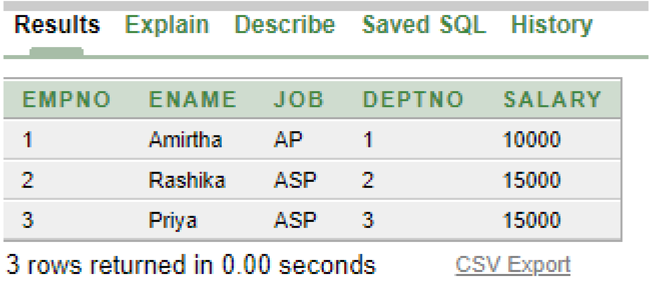


| EMPNO | ENAME | JOB | DEPTNO | SALARY |
|---|---|---|---|---|
| 1 | Amirtha | AP | 1 | 10000 |
| 2 | Rashika | ASP | 2 | 14000 |
| 3 | Priya | ASP | 3 | 12000 |

3 rows returned in 0.00 seconds

**Q3: Update the emp table to set the salary employees to Rs.15000/- who are working as ASP**

**update emp set salary=15000 where job='ASP';**

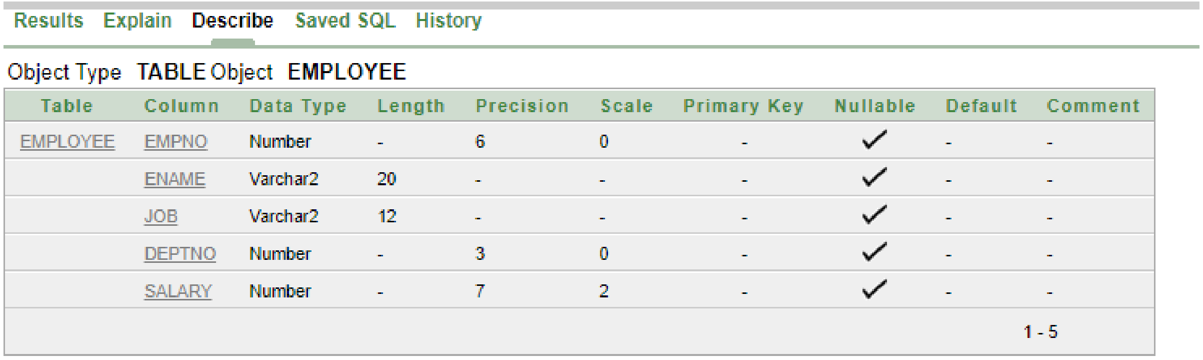


| EMPNO | ENAME | JOB | DEPTNO | SALARY |
|---|---|---|---|---|
| 1 | Amirtha | AP | 1 | 10000 |
| 2 | Rashika | ASP | 2 | 15000 |
| 3 | Priya | ASP | 3 | 15000 |

3 rows returned in 0.00 seconds

**<u>Q4: Create a pseudo cod table employee with the same structure as the table emp and insert rows into the table select clauses</u>**

**create table employee as select * from emp;**



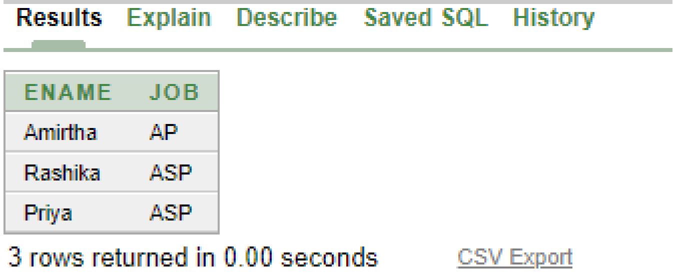
Object Type TABLE Object EMPLOYEE

| Table | Column | Data Type | Length | Precision | Scale | Primary Key | Nullable | Default | Comment |
|---|---|---|---|---|---|---|---|---|---|
| EMPLOYEE | EMPNO | Number | - | 6 | 0 | - | ✓ | - | - |
| | ENAME | Varchar2 | 20 | - | - | - | ✓ | - | - |
| | JOB | Varchar2 | 12 | - | - | - | ✓ | - | - |
| | DEPTNO | Number | - | 3 | 0 | - | ✓ | - | - |
| | SALARY | Number | - | 7 | 2 | - | ✓ | - | - |

1 - 5

**<u>Q5: Select employee name, job from emp table</u>**

**select ename,job from emp;**

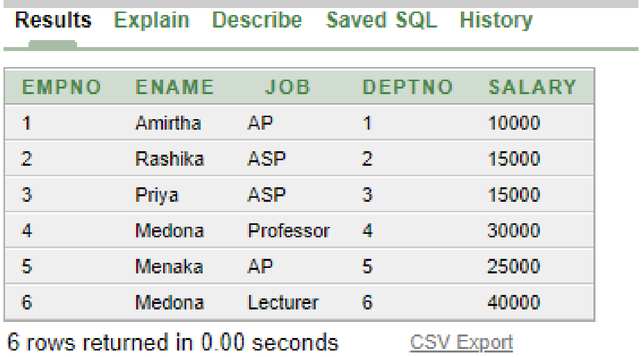


| ENAME | JOB |
|---|---|
| Amirtha | AP |
| Rashika | ASP |
| Priya | ASP |

3 rows returned in 0.00 seconds

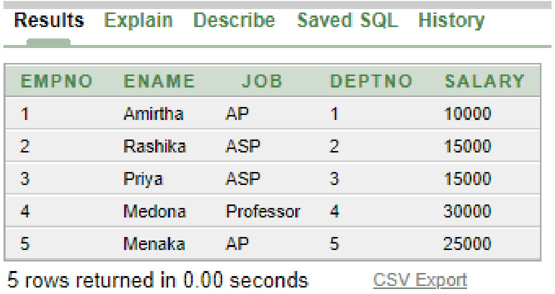
**<u>Q6: Delete only those who are working as Lecturer.</u>**

**delete from emp where job='Lecturer';**



| EMPNO | ENAME | JOB | DEPTNO | SALARY |
|---|---|---|---|---|
| 1 | Amirtha | AP | 1 | 10000 |
| 2 | Rashika | ASP | 2 | 15000 |
| 3 | Priya | ASP | 3 | 15000 |
| 4 | Medona | Professor | 4 | 30000 |
| 5 | Menaka | AP | 5 | 25000 |
| 6 | Medona | Lecturer | 6 | 40000 |

6 rows returned in 0.00 seconds

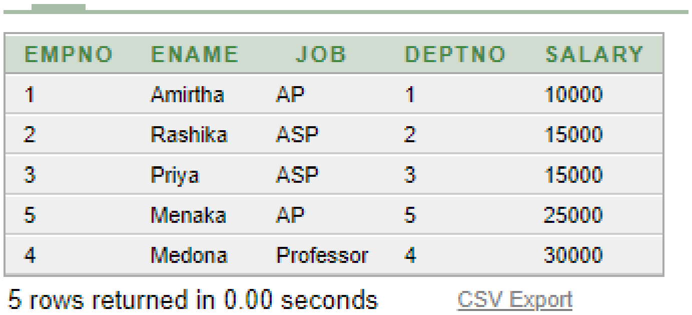


| EMPNO | ENAME | JOB | DEPTNO | SALARY |
|---|---|---|---|---|
| 1 | Amirtha | AP | 1 | 10000 |
| 2 | Rashika | ASP | 2 | 15000 |
| 3 | Priya | ASP | 3 | 15000 |
| 4 | Medona | Professor | 4 | 30000 |
| 5 | Menaka | AP | 5 | 25000 |

5 rows returned in 0.00 seconds

**<u>Q7: List the records in the emp table ordering salary in ascending order.</u>**

**select * from emp order by salary;**

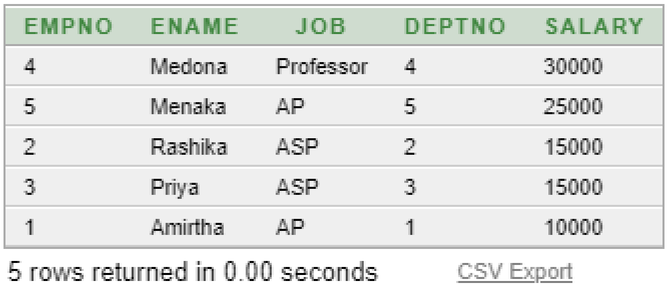


| EMPNO | ENAME | JOB | DEPTNO | SALARY |
|---|---|---|---|---|
| 1 | Amirtha | AP | 1 | 10000 |
| 2 | Rashika | ASP | 2 | 15000 |
| 3 | Priya | ASP | 3 | 15000 |
| 5 | Menaka | AP | 5 | 25000 |
| 4 | Medona | Professor | 4 | 30000 |

5 rows returned in 0.00 seconds CSV Export

**Q8: List the records in the emp table ordering salary in descending order.**

**select * from emp order by salary desc;**

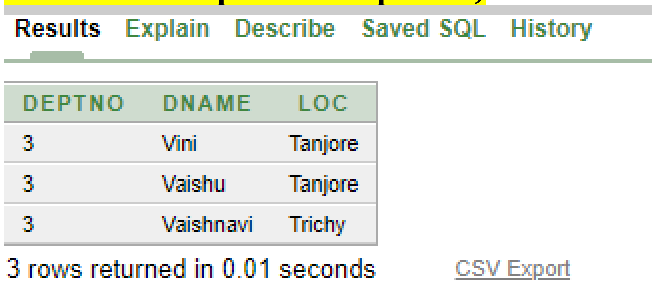


| EMPNO | ENAME | JOB | DEPTNO | SALARY |
|---|---|---|---|---|
| 4 | Medona | Professor | 4 | 30000 |
| 5 | Menaka | AP | 5 | 25000 |
| 2 | Rashika | ASP | 2 | 15000 |
| 3 | Priya | ASP | 3 | 15000 |
| 1 | Amirtha | AP | 1 | 10000 |

5 rows returned in 0.00 seconds CSV Export

**Q9: Display only those employees whose deptno is 30.**

**select * from dept where deptno=3;**

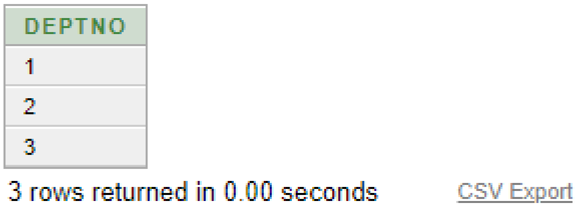


| DEPTNO | DNAME | LOC |
|---|---|---|
| 3 | Vini | Tanjore |
| 3 | Vaishu | Tanjore |
| 3 | Vaishnavi | Trichy |

3 rows returned in 0.01 seconds CSV Export

**Q10: Display deptno from table employee avoiding the duplicated values.**

**select distinct deptno from emp;**



| DEPTNO |
|---|
| 1 |
| 2 |
| 3 |

3 rows returned in 0.00 seconds CSV Export

**Result:**

Thus the DML commands using from where clause was performed successfully and executed.

# Ex :No 4 Queries using In Built Functions

Aim : To perform queries using Built In functions

## Built In Functions

### DATE FUNCTION

**1. Add_month**

This function returns a date after adding a specified date with specified number of months.

**Syntax:** Add_months(d,n); where d-date n-number of months

**Example:** Select add_months(sysdate,2) from dual;

**2. last_day**

It displays the last date of that month.

**Syntax:** last_day (d); where d-date

**Example:** Select last_day (‘1-jun-2009’) from dual;

**3. Months_between**

It gives the difference in number of months between d1 & d2.

**Syntax:** month_between (d1,d2); where d1 & d2 -dates

**Example:** Select month_between (‘1-jun-2009’,’1-aug-2009’) from dual;

**4. next_day**

It returns a day followed the specified date.

**Syntax**: next_day (d,day);

**Example:** Select next_day (sysdate,’wednesday’) from dual

## NUMERICAL FUNCTIONS

| Command | Query | Output |
|---|---|---|
| Abs(n) | Select abs(-15) from dual; | 15 |
| Ceil(n) | Select ceil(55.67) from dual; | 56 |
| Exp(n) | Select exp(4) from dual; | 54.59 |
| Floor(n) | Select floor(100.2) from | 100 |
| Power(m,n | dual; Select power(4,2) | 16 |
| ) Mod(m,n) | from dual; Select mod(10,3) | 1 |

## GROUP BY CLAUSE

This allows us to use simultaneous column name and group functions.

**Example:** Select max(percentage), deptname from student group by deptname;

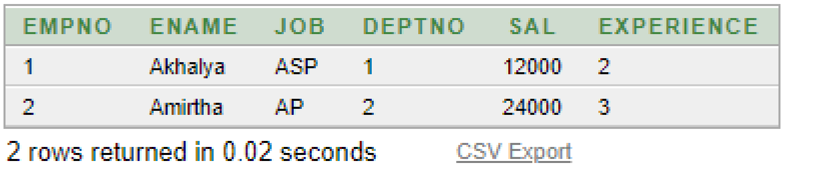
## HAVING CLAUSE

This is used to specify conditions on rows retrieved by using group by clause. **Example:** Select max(percentage), deptname from student group by deptname having count(*)>=50;**ueries:**

## Queries

**Q1: Display all the details of the records whose employee name starts with ‘A’.**

**select * from emp where ename like 'A%';**

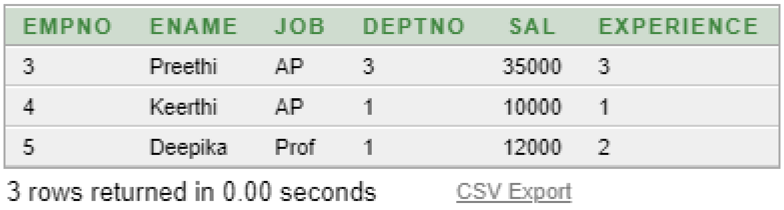


| EMPNO | ENAME | JOB | DEPTNO | SAL | EXPERIENCE |
|---|---|---|---|---|---|
| 1 | Akhalya | ASP | 1 | 12000 | 2 |
| 2 | Amirtha | AP | 2 | 24000 | 3 |

2 rows returned in 0.02 seconds

**Q2: Display all the details of the records whose employee name does not starts with ‘A’.**

**select * from emp where ename not like 'A%';**

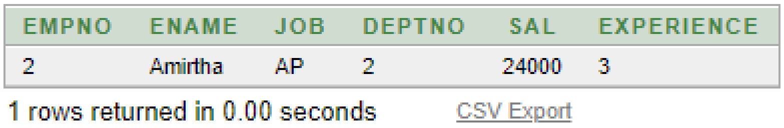


| EMPNO | ENAME | JOB | DEPTNO | SAL | EXPERIENCE |
|---|---|---|---|---|---|
| 3 | Preethi | AP | 3 | 35000 | 3 |
| 4 | Keerthi | AP | 1 | 10000 | 1 |
| 5 | Deepika | Prof | 1 | 12000 | 2 |

3 rows returned in 0.00 seconds CSV Export

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

**select * from emp where sal between 15000 and 30000;**

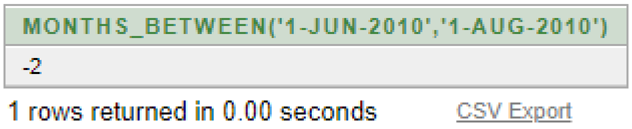


| EMPNO | ENAME | JOB | DEPTNO | SAL | EXPERIENCE |
|---|---|---|---|---|---|
| 2 | Amirtha | AP | 2 | 24000 | 3 |

1 rows returned in 0.00 seconds CSV Export

**Q4: Display the month between “1-Jun-10” and “1-Aug-10” in full.**

**select months_between('1-jun-2010','1-aug-2010') from dual;**

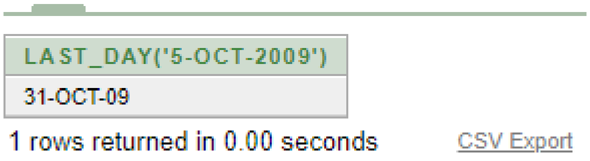


| MONTHS_BETWEEN('1-JUN-2010','1-AUG-2010') |
|---|
| -2 |

1 rows returned in 0.00 seconds CSV Export

**Q5: Display the last day of that month in “5-Oct-09”.**

**select last_day('5-oct-2009') from dual;**

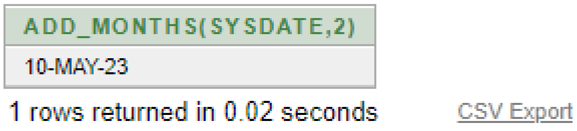


| LAST_DAY('5-OCT-2009') |
|---|
| 31-OCT-09 |

1 rows returned in 0.00 seconds CSV Export

**Q6: Display the month by adding 2 months to the system date from dual.**

**select add_months(sysdate,2)from dual;**

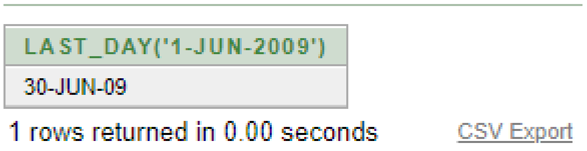


| ADD_MONTHS(SYSDATE,2) |
|---|
| 10-MAY-23 |

1 rows returned in 0.02 seconds CSV Export

**Q7: Display the last day of “1-Jun-2009” from dual.**

**select last_day('1-jun-2009')from dual;**

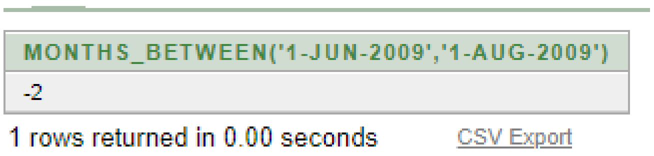


| LAST_DAY('1-JUN-2009') |
|---|
| 30-JUN-09 |

1 rows returned in 0.00 seconds CSV Export

**Q8: Display the number of months between “1-Jun-2009” and “1-Aug-2009” from dual.**

**select months_between('1-jun-2009','1-aug-2009')from dual;**

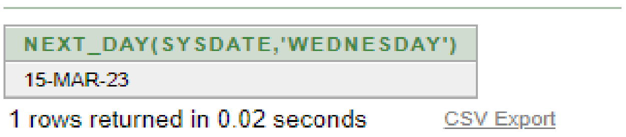


| MONTHS_BETWEEN('1-JUN-2009','1-AUG-2009') |
|---|
| -2 |

1 rows returned in 0.00 seconds CSV Export

**Q9: Display the next day of the system date from dual.**

**select next_day(sysdate,'wednesday')from dual;**

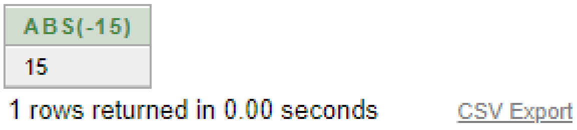


| NEXT_DAY(SYSDATE,'WEDNESDAY') |
|---|
| 15-MAR-23 |

1 rows returned in 0.02 seconds CSV Export

**Q10: Display the absolute value for -15 from dual**

**select abs(-15)from dual;**

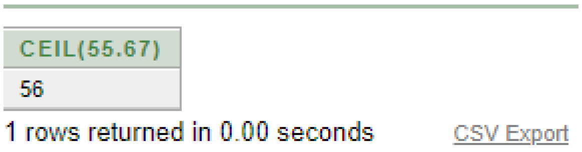


| ABS(-15) |
|---|
| 15 |

1 rows returned in 0.00 seconds CSV Export

**Q11: Display the ceiling value of 55.67 from dual**

**select ceil(55.67)from dual;**

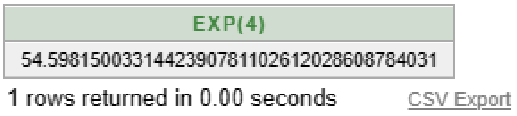


| CEIL(55.67) |
|---|
| 56 |

1 rows returned in 0.00 seconds CSV Export

**Q12: Display the exponential value of 4 from dual.**

**select exp(4)from dual;**

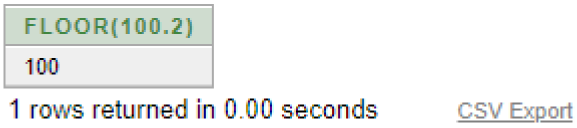


| EXP(4) |
|---|
| 54.5981500331442390781102612028608784031 |

1 rows returned in 0.00 seconds CSV Export

**Q13: Display the floor value of 100.2 from dual**

**select floor(100.2)from dual;**

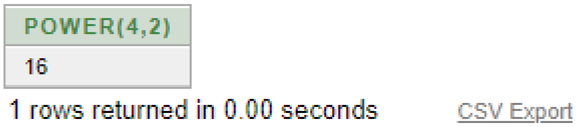


| FLOOR(100.2) |
|---|
| 100 |

1 rows returned in 0.00 seconds CSV Export

**Q14: Display the value of $4^2$ from dual**

**select power(4,2)from dual;**

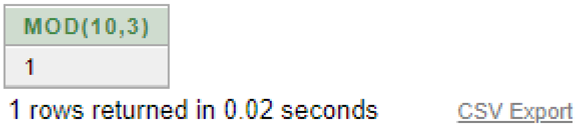


| POWER(4,2) |
|---|
| 16 |

1 rows returned in 0.00 seconds CSV Export

**<u>Q15: Display the remainder value of 10/3 from dual.</u>**

**select mod(10,3)from dual;**



| MOD(10,3) |
|---|
| 1 |

1 rows returned in 0.02 seconds CSV Export

**Result:**

Thus Built in Functions, Group Functions, Group By Clause , Having clause have been performed successfully and executed.

# Ex No 5 Aggregate Functions

Aim : to implement aggregate functions using oracle 10g

## AGGREGATE FUNCTIONS

A group function returns a result based on group of rows.

**1. avg - Example:** select avg (total) from student;

**2. max - Example**: select max (percentagel) from student;

**2.min - Example:** select min (marksl) from student;

**4. sum - Example:** select sum(price) from product;

## COUNT FUNCTION

In order to count the number of rows, count function is used.

**1. count(*)** – It counts all, inclusive of duplicates and nulls.

**Example:** select count(*) from student;

**2. count(col_name)–** It avoids null value.

**Example**: select count(total) from order;

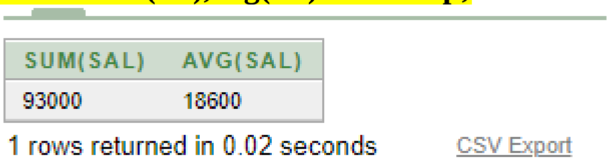
**2. count(distinct col_name)** – It avoids the repeated and null values.

**Example:** select count(distinct ordid) from order;

## Queries

**Q1: Calculate the total and average salary amount of the emp table.**

**select sum (sal),avg(sal) from emp;**



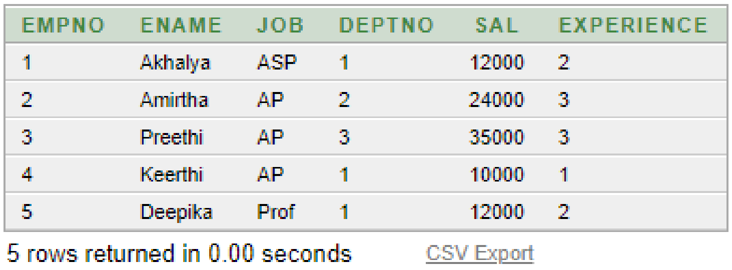
| SUM(SAL) | AVG(SAL) |
|---|---|
| 93000 | 18600 |

1 rows returned in 0.02 seconds

**Q2: Count the total records in the emp table.**

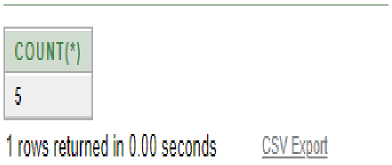
**select * from emp;**

**select count(*) from emp;**



| EMPNO | ENAME | JOB | DEPTNO | SAL | EXPERIENCE |
|---|---|---|---|---|---|
| 1 | Akhalya | ASP | 1 | 12000 | 2 |
| 2 | Amirtha | AP | 2 | 24000 | 3 |
| 3 | Preethi | AP | 3 | 35000 | 3 |
| 4 | Keerthi | AP | 1 | 10000 | 1 |
| 5 | Deepika | Prof | 1 | 12000 | 2 |

5 rows returned in 0.00 seconds CSV Export

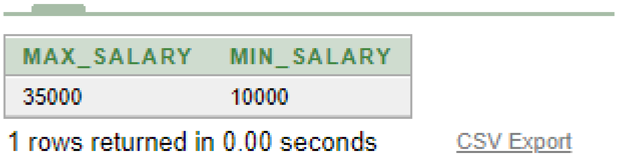


| COUNT(*) |
|---|
| 5 |

1 rows returned in 0.00 seconds CSV Export

**Q3: Determine the max and min salary and rename the column as max_salary and min_salary.**

**select max (sal)as max_salary,min(sal) as min_salary from emp;**



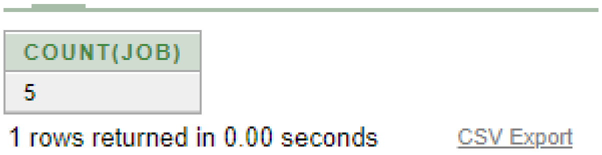
| MAX_SALARY | MIN_SALARY |
|---|---|
| 35000 | 10000 |

1 rows returned in 0.00 seconds CSV Export

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

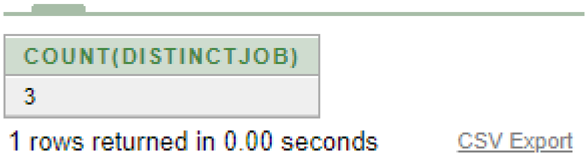
**select count(job)from emp;**

**select count(distinct job)from emp;**



| COUNT(JOB) |
|---|
| 5 |

1 rows returned in 0.00 seconds CSV Export

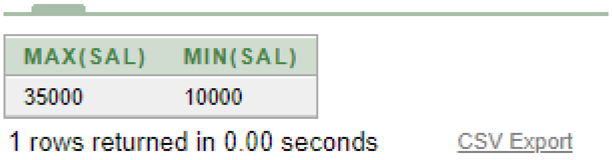


| COUNT(DISTINCTJOB) |
|---|
| 3 |

1 rows returned in 0.00 seconds CSV Export

**Q4: What is the difference between maximum and minimum salaries of employees in the organization?**

**select max(sal),min(sal)from emp;**

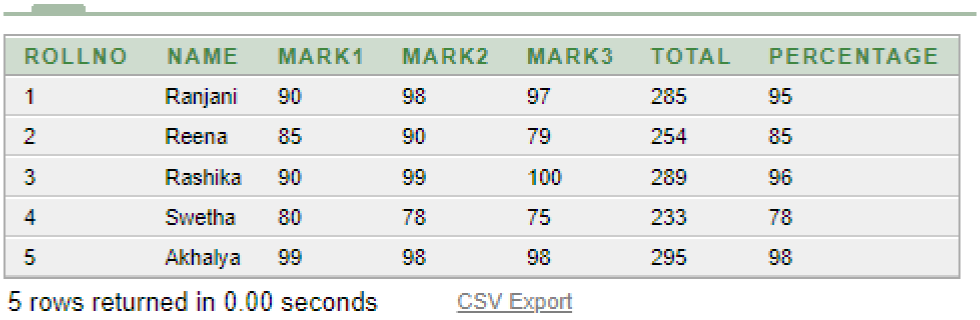


| MAX(SAL) | MIN(SAL) |
|---|---|
| 35000 | 10000 |

1 rows returned in 0.00 seconds CSV Export

**Q5: Calculate and display the total marks of the student**

**update student set total =(mark1+mark2+mark3);**

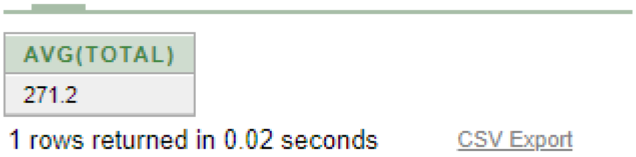


| ROLLNO | NAME | MARK1 | MARK2 | MARK3 | TOTAL | PERCENTAGE |
|---|---|---|---|---|---|---|
| 1 | Ranjani | 90 | 98 | 97 | 285 | 95 |
| 2 | Reena | 85 | 90 | 79 | 254 | 85 |
| 3 | Rashika | 90 | 99 | 100 | 289 | 96 |
| 4 | Swetha | 80 | 78 | 75 | 233 | 78 |
| 5 | Akhalya | 99 | 98 | 98 | 295 | 98 |

5 rows returned in 0.00 seconds CSV Export

**Q6: Display the average of the total from the student table.**

**select avg(total)from student;**

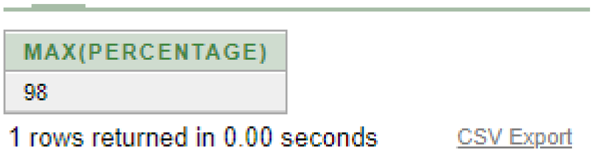


| AVG(TOTAL) |
|---|
| 271.2 |

1 rows returned in 0.02 seconds    CSV Export

**Q7: Display the maximum percentage from the student table.**

**select max(percentage)from student;**

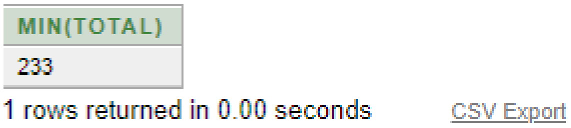


| MAX(PERCENTAGE) |
|---|
| 98 |

1 rows returned in 0.00 seconds    CSV Export

**Q8: Display the minimum total from the student table.**

**select min(total)from student;**

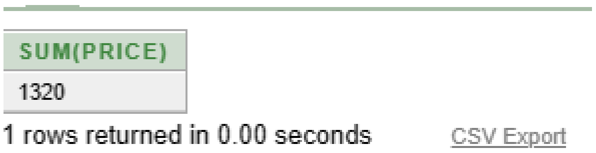


| MIN(TOTAL) |
|---|
| 233 |

1 rows returned in 0.00 seconds    CSV Export

**Q9: Display the sum of the price from the product table.**

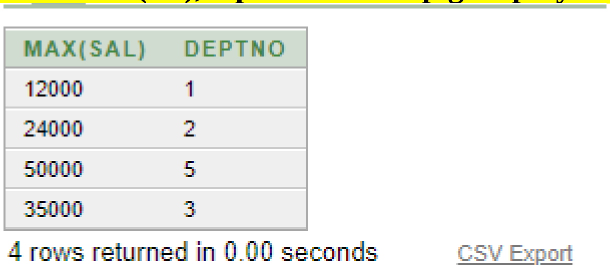
**select sum(price)from product;**



| SUM(PRICE) |
|---|
| 1320 |

1 rows returned in 0.00 seconds    CSV Export

**Q10: Display the maximum salary from dept and group the deptno from emp and dept.**
**select max(sal),deptno from emp group by deptno;**



| MAX(SAL) | DEPTNO |
|---|---|
| 12000 | 1 |
| 24000 | 2 |
| 50000 | 5 |
| 35000 | 3 |

4 rows returned in 0.00 seconds    CSV Export

# Ex No 6 : Queries using Union and Intersect

**AIM**

To perform Union and Intersect queries.

**SQL commands:**

**Union:** Returns all distinct rows selected by both the queries

**Syntax:**

Query1 Union Query2;

**Union all:** Returns all rows selected by either query including the duplicates.

**Syntax:**

Query1 Union all Query2;

**Intersect:** Returns rows selected that are common to both queries.

**Syntax:**

Query1 Intersect Query2;

**Minus:** Returns all distinct rows selected by the first query and are not by the second

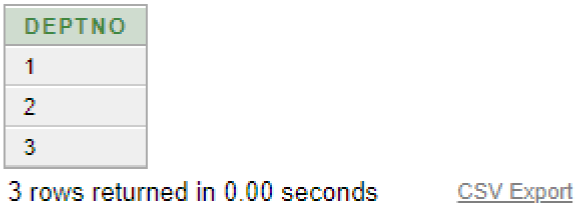
**Syntax:**

Query1 minus Query2;

**Queries:**

**Q1: Display all the dept numbers available with the dept and emp tables avoiding duplicates.**

**select deptno from emp union select deptno from dept;**

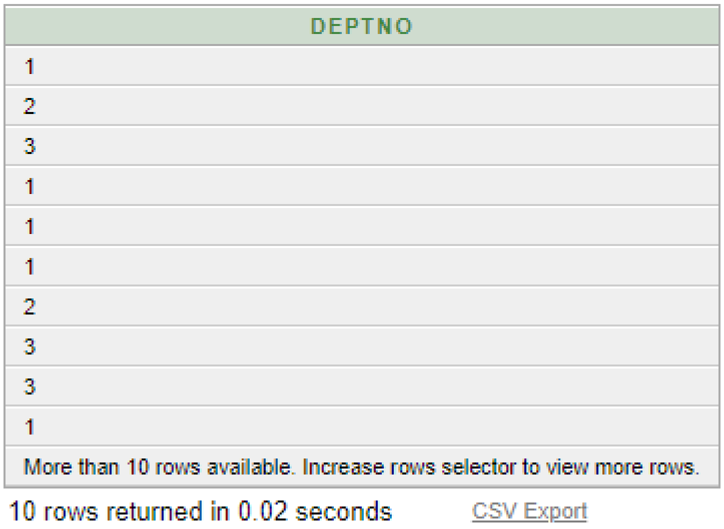


| DEPTNO |
|---|
| 1 |
| 2 |
| 3 |

3 rows returned in 0.00 seconds CSV Export

**Q2: Display all the dept numbers available with the dept and emp tables.**

**select deptno from emp union all select deptno from dept;**



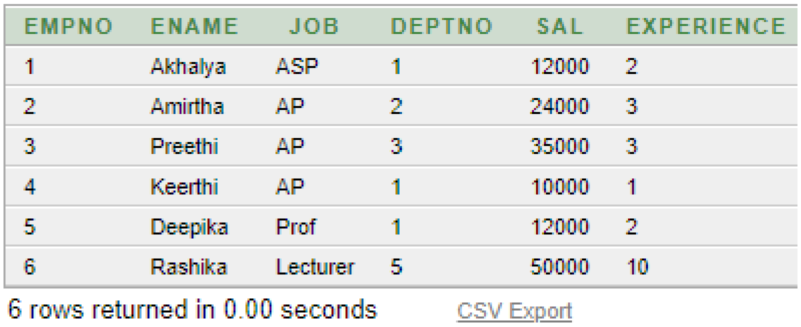
| DEPTNO |
|---|
| 1 |
| 2 |
| 3 |
| 1 |
| 1 |
| 1 |
| 2 |
| 3 |
| 3 |
| 1 |

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.02 seconds CSV Export

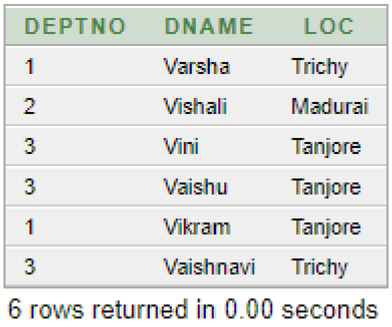
**Q3: Display all the dept numbers available in emp and not in dept tables and vice versa.**

**select deptno from emp minus select deptno from dept;**



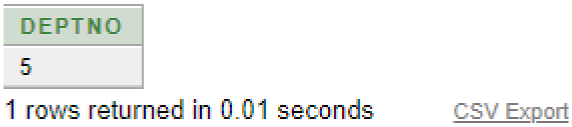
| EMPNO | ENAME | JOB | DEPTNO | SAL | EXPERIENCE |
|---|---|---|---|---|---|
| 1 | Akhalya | ASP | 1 | 12000 | 2 |
| 2 | Amirtha | AP | 2 | 24000 | 3 |
| 3 | Preethi | AP | 3 | 35000 | 3 |
| 4 | Keerthi | AP | 1 | 10000 | 1 |
| 5 | Deepika | Prof | 1 | 12000 | 2 |
| 6 | Rashika | Lecturer | 5 | 50000 | 10 |

6 rows returned in 0.00 seconds CSV Export



| DEPTNO | DNAME | LOC |
|---|---|---|
| 1 | Varsha | Trichy |
| 2 | Vishali | Madurai |
| 3 | Vini | Tanjore |
| 3 | Vaishu | Tanjore |
| 1 | Vikram | Tanjore |
| 3 | Vaishnavi | Trichy |

6 rows returned in 0.00 seconds



| DEPTNO |
|---|
| 5 |

1 rows returned in 0.01 seconds CSV Export

**Result:** Thus the set operations using DML Commands was successfully performed and executed.

## Ex No 7 : VIEWS

**AIM :**

To create and manipulate various database objects of the Table using views

**Views:**

A view is the tailored presentation of data contained in one or more table and can also be said as restricted view to the data's in the tables. A view is a "virtual table" or a "stored query" which takes the output of a query and treats it as a table. The table upon which a view is created is called as base table.

**SQL Commands**

**Creating and dropping view:**

**Syntax:**

```
Create [or replace] view <view name> [column alias names] as <query> [with <options> conditions];
Drop view <view name>;
```

**Example**:

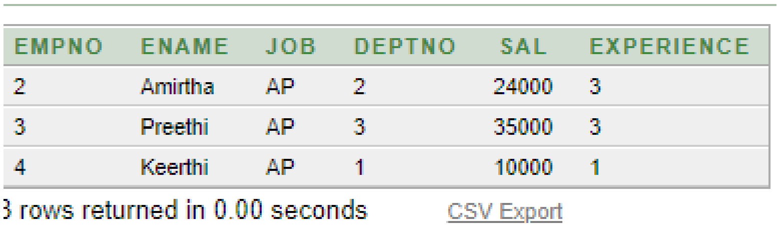
```
Create or replace view empview as select * from emp;
Drop view empview;
```

**Queries: Tables used:**

```
SQL> select * from emp;
     EMPNO ENAME    JOB        DEPTNO     SAL
---------- -------------------- ---------- ---------- ----------
```

**Q1: The organization wants to display only the details of the employees those who are AP (Horizontal Partioning)**

```
create view empview as select * from emp where job='AP';
select * from empview;
```

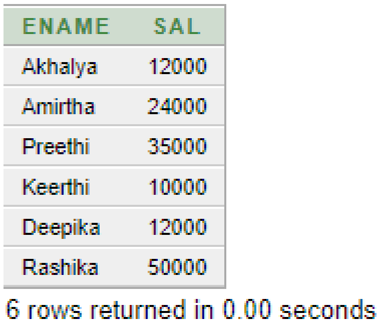


| EMPNO | ENAME | JOB | DEPTNO | SAL | EXPERIENCE |
|---|---|---|---|---|---|
| 2 | Amirtha | AP | 2 | 24000 | 3 |
| 3 | Preethi | AP | 3 | 35000 | 3 |
| 4 | Keerthi | AP | 1 | 10000 | 1 |

3 rows returned in 0.00 seconds CSV Export

**Q2: The organization wants to display only the details like empno, empname, deptno, deptname of the employees. (Vertical partitioning)**

**create view empview1 as select ename,sal from emp;**
**select * from empview1;**

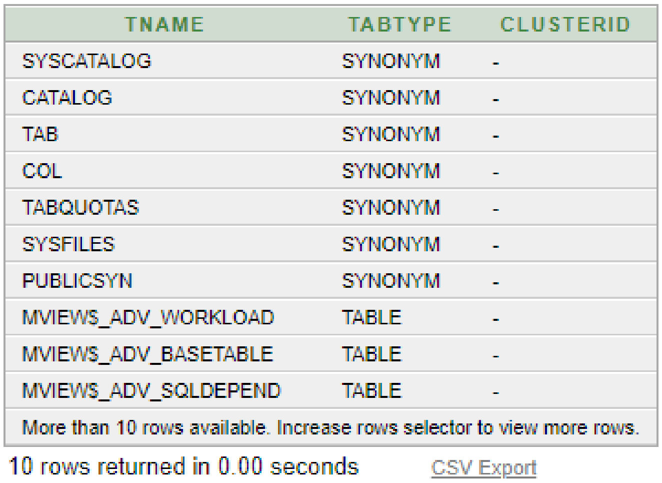


| ENAME | SAL |
|---|---|
| Akhalya | 12000 |
| Amirtha | 24000 |
| Preethi | 35000 |
| Keerthi | 10000 |
| Deepika | 12000 |
| Rashika | 50000 |

6 rows returned in 0.00 seconds

**Q3: Display all the views generated.**

**select * from tab;**



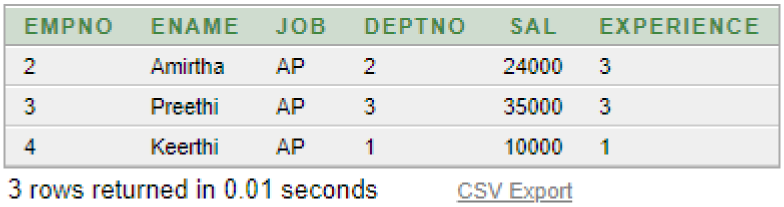
| TNAME | TABTYPE | CLUSTERID |
|---|---|---|
| SYSCATALOG | SYNONYM | - |
| CATALOG | SYNONYM | - |
| TAB | SYNONYM | - |
| COL | SYNONYM | - |
| TABQUOTAS | SYNONYM | - |
| SYSFILES | SYNONYM | - |
| PUBLICSYN | SYNONYM | - |
| MVIEWS$_ADV_WORKLOAD | TABLE | - |
| MVIEWS$_ADV_BASETABLE | TABLE | - |
| MVIEWS$_ADV_SQLDEPEND | TABLE | - |

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds CSV Export

**Q4: Execute the DML commands on the view created.**

**select * from empview;**

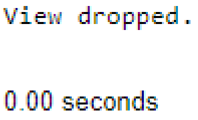


| EMPNO | ENAME | JOB | DEPTNO | SAL | EXPERIENCE |
|---|---|---|---|---|---|
| 2 | Amirtha | AP | 2 | 24000 | 3 |
| 3 | Preethi | AP | 3 | 35000 | 3 |
| 4 | Keerthi | AP | 1 | 10000 | 1 |

3 rows returned in 0.01 seconds CSV Export

**Q5: Drop a view.**

**drop view empview1;**



```
View dropped.

0.00 seconds
```

**Result:**

Thus the creation and manipulate various database objects of the Table using views was successfully executed.

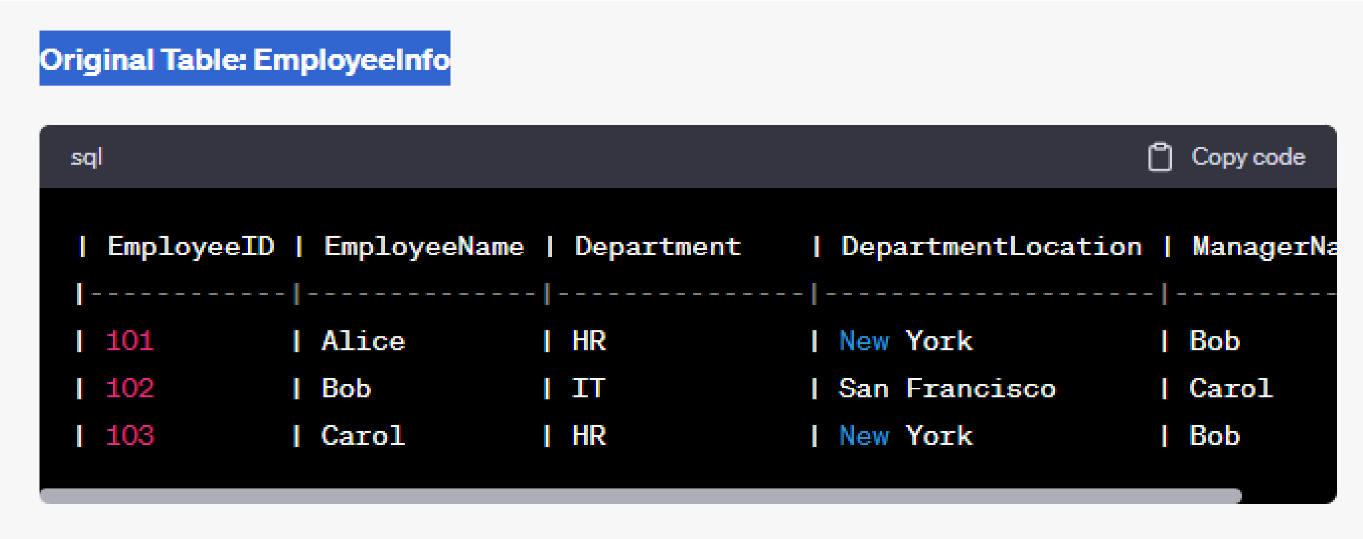
# Ex No 8: Normalization

Aim : To implement normalization using SQL

**Normalization**

Normalization is a process in database design that involves organizing and structuring relational database tables to minimize redundancy and improve data integrity. The goal of normalization is to eliminate data anomalies, such as insertion, update, and deletion anomalies, that can occur when data is not properly organized. Normalization is typically divided into several normal forms, including 1NF, 2NF, 3NF, and higher normal forms.

1. **First Normal Form (1NF):**
   - A table is in 1NF if it contains only atomic (indivisible) values, and there are no repeating groups or arrays within the table.
   - Each column should contain only one piece of information, and each row should be uniquely identifiable by a primary key.
   - Repeating groups should be moved into separate tables.
2. **Second Normal Form (2NF):**
   - A table is in 2NF if it's already in 1NF and all non-key attributes are fully functionally dependent on the entire primary key.
   - In other words, all attributes that are not part of the primary key should depend on the entire primary key, not just part of it.
3. **Third Normal Form (3NF):**
   - A table is in 3NF if it's already in 2NF and all non-key attributes are transitively dependent only on the primary key.
   - This means that non-key attributes should not depend on other non-key attributes.



Original Table: EmployeeInfo

```
| EmployeeID | EmployeeName | Department     | DepartmentLocation  | ManagerNa
|------------|--------------|----------------|---------------------|----------
| 101        | Alice        | HR             | New York            | Bob
| 102        | Bob          | IT             | San Francisco       | Carol
| 103        | Carol        | HR             | New York            | Bob
```

This table is not normalized because there's a repeating group for the manager's name and the manager's name depends on another non-key attribute. Let's proceed with the normalization steps.

**Step 1: First Normal Form (1NF)** In this step, we'll remove repeating groups by creating a separate table for departments.

Create the Departments

```
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50),
    DepartmentLocation VARCHAR(50)
);
```

**Modify the employee information table**

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(50),
    DepartmentID INT,
    ManagerID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

**Step 2: Second Normal Form (2NF)** In this step, we'll address partial dependency. The manager's name depends on the manager's ID, which is not part of the EmployeeInfo table.

Create the Managers

```
CREATE TABLE Managers (
   ManagerID INT PRIMARY KEY,
   ManagerName VARCHAR(50)
);
```

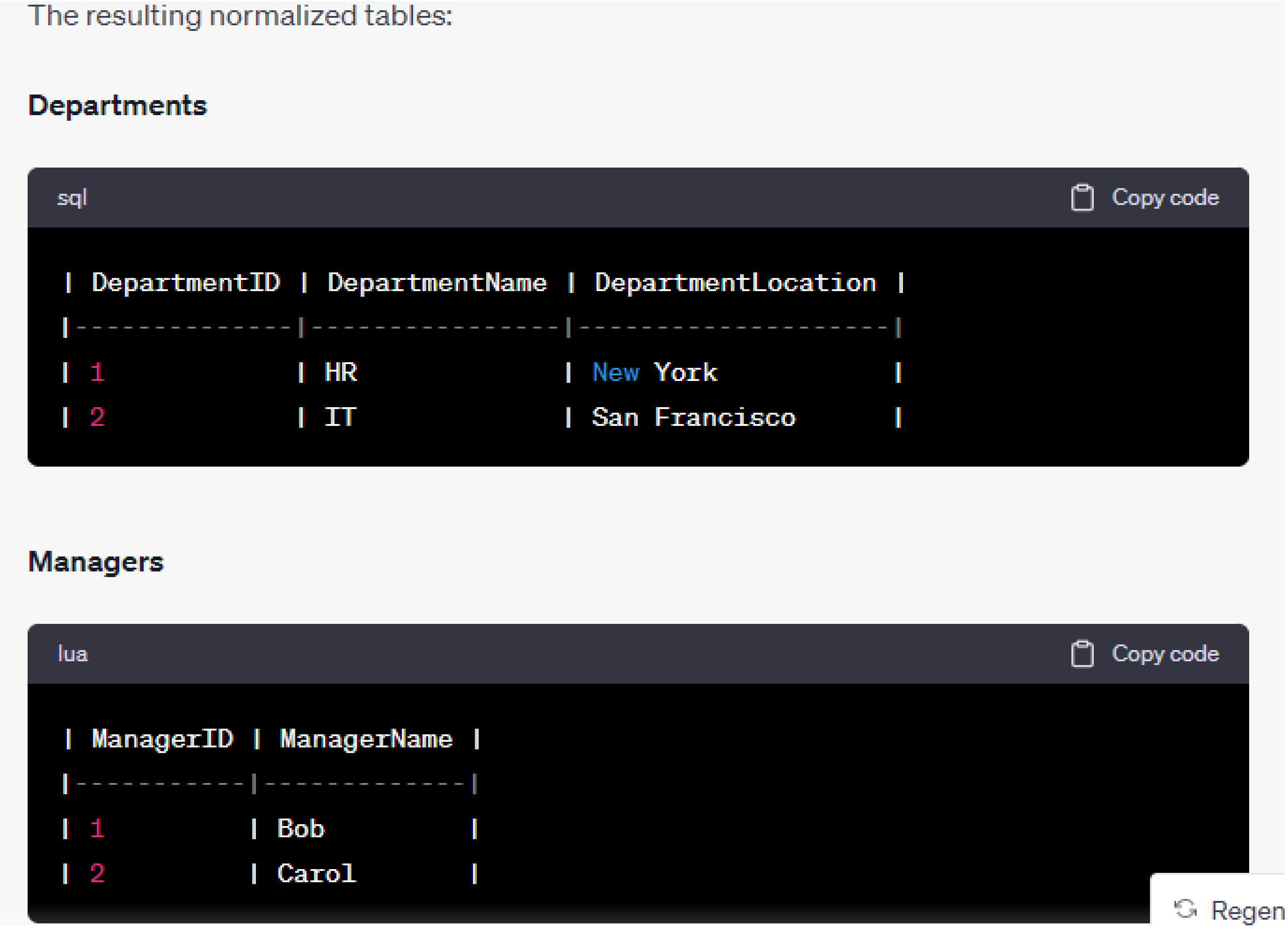
Modify the Employees table to reference the Managers table

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   EmployeeName VARCHAR(50),
   DepartmentID INT,
   ManagerID INT,
   FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID),
   FOREIGN KEY (ManagerID) REFERENCES Managers(ManagerID)
);
```

**Step 3: Third Normal Form (3NF)** In this step, we'll eliminate transitive dependency. The manager's name depends on the manager's ID, which is now properly separated.

No further modification is needed for this example.

The resulting normalized tables:



The resulting normalized tables:

**Departments**

```
| DepartmentID | DepartmentName | DepartmentLocation |
|--------------|----------------|--------------------|
| 1            | HR             | New York           |
| 2            | IT             | San Francisco      |
```

**Managers**

```
| ManagerID | ManagerName |
|-----------|-------------|
| 1         | Bob         |
| 2         | Carol       |
```

**Result : Thus normalization concept has been implemented using SQL**

## Ex No 9: Procedures and Functions

**AIM:** To develop procedures and function for various operations.

A procedure is a block that can take parameters (sometimes referred to as arguments) and be invoked. Procedures promote reusability and maintainability. Once validated, they can be used in number of applications. If the definition changes, only the procedure are affected, this greatly simplifies maintenance.

**KEYWORDS AND THEIR PURPOSES**

**REPLACE:** It recreates the procedure if it already exists.

PROCEDURE: It is the name of the procedure to be created.

**ARGUMENT:** It is the name of the argument to the procedure. Parenthesis can be omitted if no arguments are present.

**IN:** Specifies that a value for the argument must be specified when calling the procedure ie., used to pass values to a sub-program. This is the default parameter.

**OUT:** Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie. used to return values to a caller of the sub-program.

**INOUT:** Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it's calling environment after execution.

**RETURN:** It is the data type of the function's return value because every function must return a value, this clause is required.

**PROCEDURE Syntax :**

```
create or replace procedure <procedure name> (argument {in,out,inout} datatype ) {is,as}
variable declaration; constant
declaration;
begin
PL/SQL subprogram body;
exception
exception PL/SQL block;
end;
```

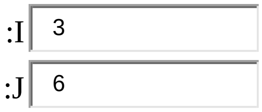
**Syntax:**

```
create or replace function <function name> (argument in datatype,……) return datatype {is,as}
variable declaration;
constant declaration;
begin
PL/SQL subprogram body;
exception
exception PL/SQL block;
end;
```

**1. PL/SQL program to add two number.**

```
declare
i integer;
j integer;
k integer;
begin
i:=:i;
j:=:j;
k:=i+j;
dbms_output.put_line(k);
end;
/
```

Input:



```
:I 3
:J 6
```

Output:

```
9

Statement processed.
```

**2. PL/SQL program to find factorial of a number.**

```
declare
n number;
fac number:=1;
i number;
begin
n:=:n;
```

```
for i in 1..n
loop
fac:=fac*i;
end loop;
dbms_output.put_line('factorial='||fac);
end;
/
```

Input:

```
:N 5
```

Output:

```
factorial=120

Statement processed.
```

## **3.PL/SQL program for Fibonacci series**

```
declare
fir number:=0;
sec number:=1;
third number;
n number:=:n;
i number;
begin
dbms_output.put_line('fibonacci series is:')
dbms_output.put_line(fir);
dbms_output.put_line(sec);
for i in 2..n
loop
third:=fir+sec;
fir:=sec;
sec:=third;
dbms_output.put_line(third);
end loop;
end;
/
```

Input:

```
:N 6
```

Output:

```
fibonacci series is:
0
1
1
2
3
5
8

Statement processed.
```

**4. PL/SQL program to swap two number**

```
declare
a number;
b number;
temp number;
begin
a:=5;
b:=10;
dbms_output.put_line('before swapping:');
dbms_output.put_line('a='||a||'b='||b);
temp:=a;
a:=b;
b:=temp;
dbms_output.put_line('after swapping:');
dbms_output.put_line('a='||a||'b='||b);
end;
/
```

Output:

```
before swapping:
a=5b=10
after swapping:
a=10b=5

Statement processed.
```

**5. create a procedure for student mark details.**

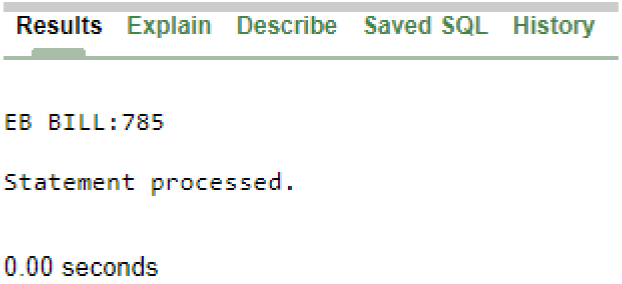
```
create or replace procedure markdetails1
(
id in number,
m1 in number,
m2 in number,
m3 in number,
m4 in number,
m5 in number,
total out number,
per out number,
grade out varchar2
)
is
gradeA number(5,2):=90;
gradeB number(5,2):=80;
gradeC number(5,2):=70;
gradeD number(5,2):=60;
begin
total:=m1+m2+m3+m4+m5;
per:=total/500*100;

if per>=gradeA then
grade:='A';
elSif per>=gradeB and per<gradeA then
grade:='B';
elSif per>=gradeC and per<gradeB then
grade:='c';
elSif per>=gradeD and per<gradeC then
grade:='D';
else
grade:='F';
end if;
end;
/
Declare
total1 number;
per1 number(9,2);
grade1 varchar2(1);
begin
markdetails1(124,75,80,85,90,95, total=>total1, per=>per1, grade=>grade1);
dbms_output.put_line('total marks'||total1);
dbms_output.put_line('per'||per1||'%');
dbms_output.put_line('grade'||grade1);
end;
/
```

6**. create or replace procedure calc_eb_bill(p_units in number, p_rate_per_unit in number:=3.50,p_fixed_charges in number:=50.00,p_tax_rate in number:=0.05,p_eb_bill out number)**

```
is
begin
declare
energy_charge number(10,2):=p_units*p_rate_per_unit;
begin
declare
tax_amount number(10,2):=energy_charge*p_tax_rate;
begin
```

```
p_eb_bill:=energy_charge+tax_amount+p_fixed_charges;
end;
end;
end;
/
declare
eb_bill number(10,2);
begin
calc_eb_bill(200,p_eb_bill=>eb_bill);
dbms_output.put_line('EB BILL:'||eb_bill);
end;
/
```

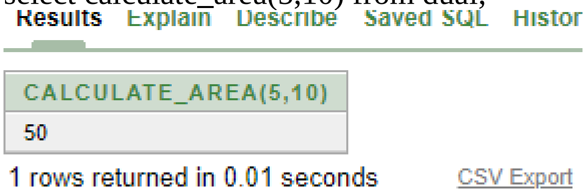


```
EB BILL:785

Statement processed.

0.00 seconds
```

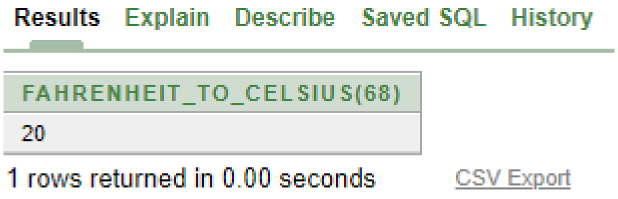
```
7. create or replace function calculate_area(length in number,width in number)
return number
is
begin
return length*width;
end;
/
select calculate_area(5,10) from dual;
```



| CALCULATE_AREA(5,10) |
|---|
| 50 |

1 rows returned in 0.01 seconds

```
create or replace function fahrenheit_to_celsius(temp_f in number)
return number
is
temp_c number;
begin
temp_c:=(temp_f-32)*5/9;
return temp_c;
end;
/
select fahrenheit_to_celsius(68) from dual;
```



| FAHRENHEIT_TO_CELSIUS(68) |
|---|
| 20 |

1 rows returned in 0.00 seconds

# Ex No 10 Trigger Creation

**Aim :** To create Trigger.

## TRIGGER

A Trigger is a stored procedure that defines an action that the database automatically take when some database-related event such as Insert, Update or Delete occur.

## TRIGGER VS. PROCEDURE VS CURSOR

| TRIGGER | PROCEDURES | CURSORS |
|---|---|---|
| These are named PL/SQL blocks. | These are named PL/SQL blocks. | These are named PL/SQL blocks. |
| These are invoked automatically. | User as per need invokes these. | These can be created both explicitly and implicitly. |
| These can't take parameters. | These can take parameters. | These can take parameters. |
| These are stored in database. | These are stored in database. | These are not stored in database. |

## TYPES OF TRIGGERS

The various types of triggers are as follows,

- **Before**: It fires the trigger before executing the trigger statement.
- **After**: It fires the trigger after executing the trigger statement.
- **For each row**: It specifies that the trigger fires once per row.
- **For each statement**: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

## VARIABLES USED IN TRIGGERS

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

**Row Level Trigger vs. Statement Level Trigger:**

| Row Level Trigger | Statement Level Trigger |
|---|---|
| These are fired for each row affected by the DML statement. | These are fired once for the statement instead of the no of rows modified by it. |
| These are used for generating/checking the values begin inserted or updated. | These are used for generated the summary information. |

**Before trigger vs. after trigger**

| Before Triggers | After Triggers |
|---|---|
| Before triggers are fired before the DML statement is actually executed. | After triggers are fired after the DML statement has |

**Sytax:**

Create or replace trigger <trg_name> Before /After Insert/Update/Delete
[of column_name, column_name….]
on table_name> [for each row] [when condition]
begin
---statement end;

**Q1: Create a trigger that insert current user into a username column of an existing**

**Program:**

```
SQL> create table itstudent4(name varchar2(15),username varchar2(15)); Table
created.
SQL> create or replace trigger itstudent4 before insert on itstudent4 for each row
  2  declare
  3  name varchar2(20);
  4  begin
  5  select user into name from dual;
  6  :new.username:=name;
  7  end;
  8   /
```

Trigger created.

**Outpu**t:

```
SQL> insert into itstudent4 values('&name','&username'); Enter value
for name: akbar
Enter value for username: ranjani
old   1: insert into itstudent4 values('&name','&username')
new   1: insert into itstudent4 values('akbar','ranjani')
1 row created. SQL> /
Enter value for name: suji
Enter value for username: priya
old   1: insert into itstudent4 values('&name','&username')
new   1: insert into itstudent4 values('suji','priya')
1 row created.

SQL> select * from itstudent4;
NAME        USERNAME
---------------          -------------          akbar
               SCOTT
suji           SCOTT
```

**Q2: Create a Simple Trigger that does not allow Insert Update and Delete Operations on the Table**

**Program**

**Table used:**

```
SQL> select * from itempls;
ENAME         EID   SALARY
---------- --------- --------
xxx            11    10000
 yyy           12    10500
 zzz           13    15500
```

**Trigger:**

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each row
  2  begin
  3  raise_application_error(-20010,'You cannot do manipulation');
  4  end;
  5   /
Trigger created.
```

**Output:**

```
SQL> insert into itempls values('aaa',14,34000);
insert into itempls values('aaa',14,34000)
            *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'
            *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
SQL> update itempls set eid=15 where ename='yyy';
update itempls set eid=15 where ename='yyy'
       *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
```

**Q3: Create a Trigger that raises an User Defined Error Message and does not allow updating and Insertion**

**Program: Table used:**

```
SQL> select * from itempls;
ENAME         EID   SALARY
---------- --------- -------- xxx   11
10000 yyy   12    10500 zzz   13
                  15500
```

**Trigger:**

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
  2  declare
  3  triggsal itempls.salary%type;
  4  begin
  5  select salary into triggsal from itempls where eid=12;
  6  if(:new.salary>triggsal or :new.salary<triggsal) then
  7  raise_application_error(-20100,'Salary has not been changed');
  8  end if;
  9  end;
 10  /
Trigger created.
```

**Output:**

```
        SQL> insert into itempls values ('bbb',16,45000);
        insert into itempls values ('bbb',16,45000)
               *
               ERROR at line 1:
ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
        SQL> update itempls set eid=18 where ename='zzz';
        update itempls set eid=18 where ename='zzz'
               * ERROR at line 1:
ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
```

**Q4: develop a query to Drop the Created Trigger**

**Ans:**SQL> drop trigger ittrigg; Trigger dropped.

**Result:**

Thus the creation of triggers for various events such as insertion, updation, etc., was performed and executed successfully.