LAB REPORT ON:



DATABASE MANAGEMENT SYSTEMS LAB (IT 507)

SUBMITTED BY

**ABHISHEK ANAND**

**20030141IT012**

UNDER THE GUIDANCE OF

**Dr. Chetan J Shelke**

CERTIFICATE

ALLIANCE COLLEGE OF ENGINEERING & DESIGN

NAME: ABHISHEK ANAND COURSE: INFORMATION TECHNOLOGY

REG. No: - 20030141IT012 SEMESTER: - V

This is certified to be the bonafide work of the student in the DATABASE MANAGEMENT SYSTEMS LAB during the academic year 2022/2023.

No. of practicals certified out of in the subject of

Faculty-In-Charge Head of the Department

Submitted for the practical examination held on

Internal Examiner External Examiner

INDEX



|  |  |  |
| --- | --- | --- |
| **LAB** | **PROGRAM** | **PAGE NO.** |
| 1 | E- R Diagram |  |
| 2 | Create Table |  |
| 3 | Alter and updation of table |  |
| 4 | DDL and DML |  |
| 5 | Count, Max, Min,… |  |
| 6 | Join |  |
| 7 | UNION, UNION ALL, MINUS,… |  |
| 8 | TRIGGER |  |

**LAB 1**

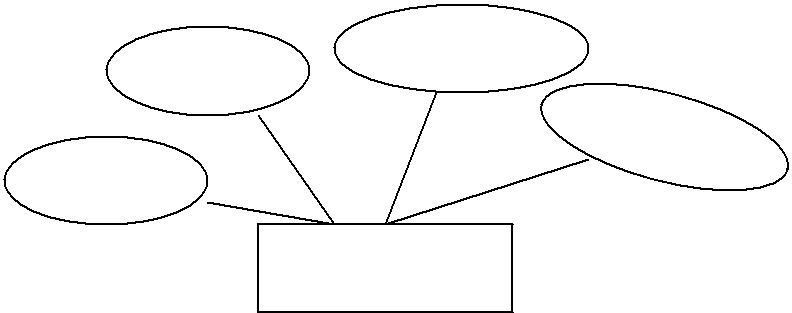
|  |  |
| --- | --- |
|  |  |

**AIM: Analyse the problem and come with the entities in it. Identify what Data has to be persisted in the databases.**

The Following are the entities:

1. Bus
2. Reservation
3. Ticket
4. Passenger
5. Cancellation

The attributes in the Entities: Bus:( Entity)



Destination

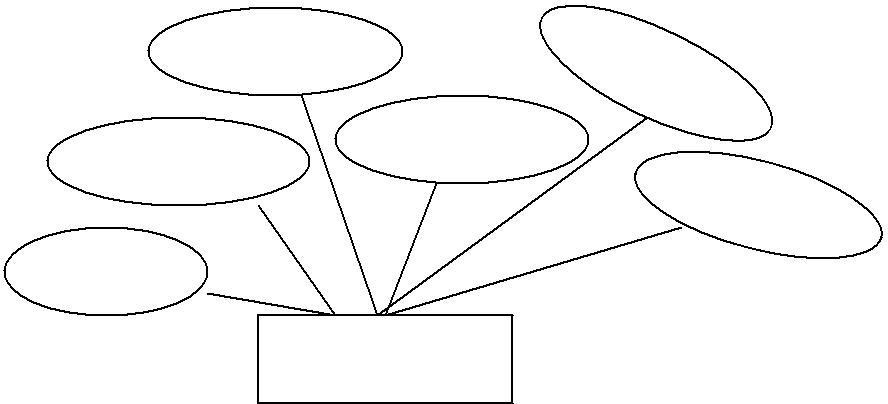
Source

Couch Type

Bus No

**Bus**

Reservation (Entity)



Contact No

Bus No

No-of-Seats

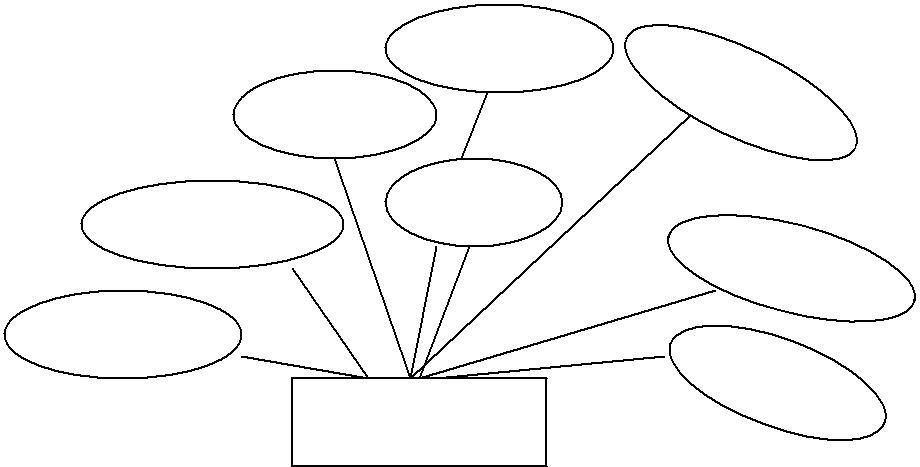
Journey date

Address

PNR NO

**Reservation**

Ticket :(Entity)



Dep- Time

Source

Age

Sex

Journey date

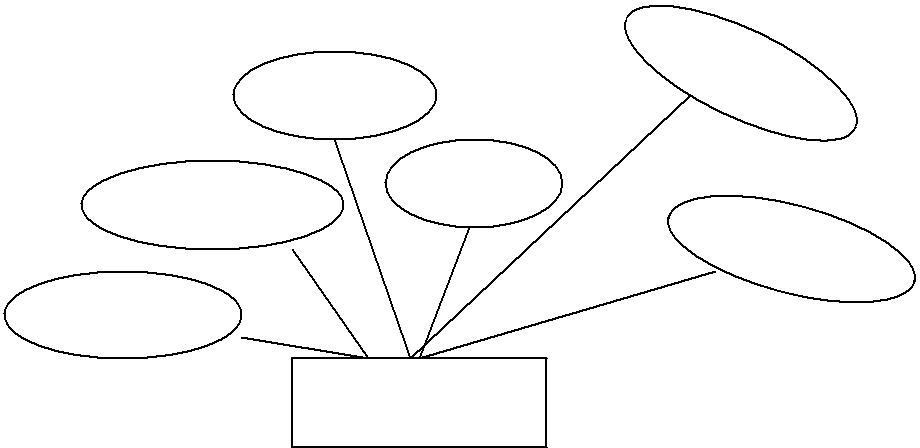
Destination

Ticket No

Bus No

**Ticket**

Passenger:



Contact NO

Age

Sex

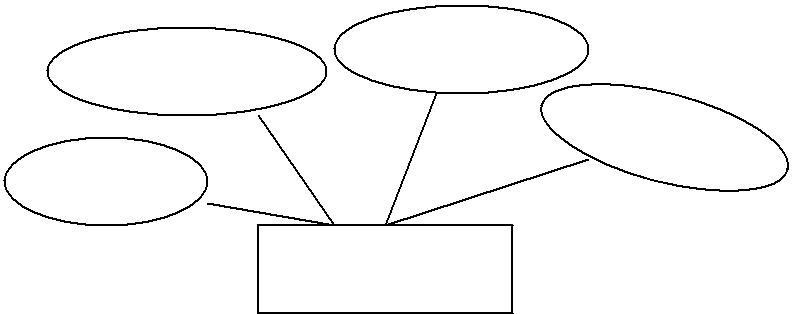
PNR NO

Name

Ticket No

**Passenger**

Cancellation (Entity)



Seat No

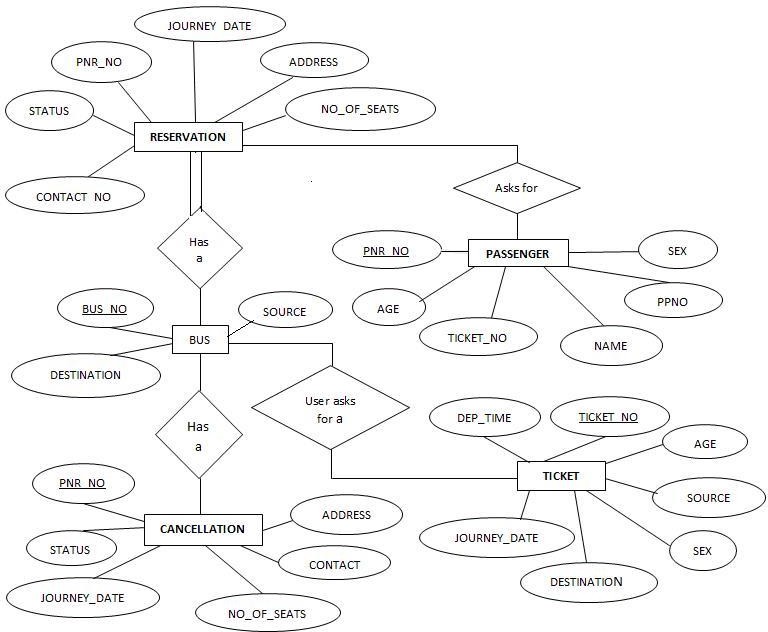
Journey date

Contact No

PNR NO

**Cancellation**

Concept design with E-R Model:



**LAB-2**

# Aim:-Create a table called Employee with the following structure.

|  |  |
| --- | --- |
| **Name** | **Type** |
| Empno | Number |
| Ename | Varchar2(10) |
| Job | Varchar2(10) |
| Mgr | Number |
| Sal | Number |

* + 1. **Add a column commission with domain to the Employee table.**
    2. Insert any five records into the table.
    3. Update the column details of job
    4. Rename the column of Employ table using alter command.
    5. Delete the employee whose Empno is 105.

**SOLUTION:**

SQL> create table employee(empno number, ename varchar2(10),job varchar2(10),mgrnumber,sal number);

Table created.

SQL> desc employee;

Name Null? Type

EMPNO

NUMBER

ENAME VARCHAR2(10)

JOB VARCHAR2(10)

MGR NUMBER

SAL NUMBER

1. **Add a column commission with domain to the Employee table.**

SQL> alter table employee add (commission number); Table altered.

SQL> desc employee;

Name Null? Type

EMPNO NUMBER

ENAME VARCHAR2(10)

JOB VARCHAR2(10)

MGR NUMBER

SAL NUMBER

COMMISSION NUMBER

1. **Insert any five records into the table.**

SQL> insert into employee values(&empno,'&ename','&job', &mgr,&sal,'&commission'); Enter value for empno: 101

Enter value for ename: abhi

Enter value for job: manager

Enter value for mgr: 1234

Enter value for sal: 10000

Enter value for commission:70

old 1: insert into employee values(&empno,'&ename','&job',&mgr,&sal,'&commission') new 1: insert into employee values(101,'abhi','manager',1234,10000,'70')

1 row created.

SQL> /

Enter value for empno: 102 Enter value for ename: rohith Enter value for job: analyst Enter value for mgr: 2345 Enter value for sal: 9000

Enter value for commission: 65

old 1: insert into employee values(&empno,'&ename','&job',&mgr,&sal,'&commission') new 1: insert into employee values(102,'rohith','analyst',2345,9000,'65')

1 row created.

SQL> /Enter value for empno: 103 Enter value for ename: david Enter value for job: analyst Enter value for mgr: 3456 Enter value for sal: 9000

Enter value for commission: 65

old 1: insert into employee values(&empno,'&ename','&job',&mgr,&sal,'&commission') new 1: insert into employee values(103,'david','analyst',3456,9000,'65')

1 row created.

SQL> /Enter value for empno: 104 Enter value for ename: rahul Enter value for job: clerk Enter value for mgr: 4567 Enter value for sal: 7000

Enter value for commission: 55

old 1: insert into employee values(&empno,'&ename','&job',&mgr,&sal,'&commission') new 1: insert into employee values(104,'rahul','clerk',4567,7000,'55')

1 row created.

SQL> / Enter value for empno: 105 Enter value for ename: pramod Enter value for job: salesman Enter value for mgr: 5678 Enter value for sal:5000

Enter value for commission: 50

old 1: insert into employee values(&empno,'&ename','&job',&mgr,&sal,'&commission') new 1: insert into employee values(105,'pramod','salesman',5678,5000,'50')

1 row created.

SQL> select \* from employee;

EMPNOENAME JOB MGRSALCOMMISSION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101 abhi | Manager | 1234 | 10000 | 70 |
| 102 rohith | Analyst | 2345 | 9000 | 65 |
| 103 david | Analyst | 3456 | 9000 | 65 |
| 104 rahul | Clerk | 4567 | 7000 | 55 |
| 105 pramod | salesman | 5678 | 5000 | 50 |

1. **Update the column details ofjob**

SQL> update employee set job='trainee' where empno=103; 1 row updated.

SQL> select \* from employee;

EMPNOENAME JOBMGR SALCOMMISSION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101 abhi | manager | 1234 | 10000 | 70 |
| 102 rohith | analyst | 2345 | 9000 | 65 |
| 103 david | trainee | 3456 | 9000 | 65 |
| 104 rahul | clerk | 4567 | 7000 | 55 |
| 105 pramod | salesman | 5678 | 5000 | 50 |

1. **Rename the column of Employ table using alter command.**

SQL> alter table employee rename column mgr to manager\_no; Table altered.

SQL> desc employee;

Name Null? Type

------------------- EMPNO NUMBER

ENAME VARCHAR2(10)

JOB VARCHAR2(10)

MANAGER\_NO NUMBER

SAL NUMBER

COMMISSION NUMBER

1. **Delete the employee whose Empno is105.**

SQL> delete employee where empno=105; 1 row deleted.

SQL> select \* from employee;

EMPNOENAME JOB MANAGER\_NO SALCOMMISSION

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101 abhi | manager | 1234 | 10000 | 70 |
| 102 rohith | analyst | 2345 | 9000 | 65 |
| 103 david | trainee | 3456 | 9000 | 65 |
| 104 rahul | clerk | 4567 | 7000 | 55 |

**LAB-3**

**Aim:-Create department table with the following structure.**

|  |  |
| --- | --- |
| **Name** | **Type** |
| Deptno | Number |
| Deptname | Varchar2(10) |
| Location | Varchar2(10) |

* + 1. Add column designation to the department table.
    2. Insert values into the table.
    3. List the records of dept table grouped by deptno.
    4. Update the record where deptno is 9.
    5. Delete any column data from the table.

**SOLUTION:**

SQL> create table department (deptno number, deptname varchar2(10), location varchar2(10)); Table created.

|  |  |  |
| --- | --- | --- |
| SQL> desc department; Name | Null? | Type |
| ------------------- DEPTNO DEPTNAME |  | NUMBER VARCHAR2(10) |
| LOCATION |  | VARCHAR2(10) |

1. **Add column designation to the department table.**

SQL> alter table department add(designation varchar2(10)); Table altered.

SQL> desc department;

Name Null? Type

------------------- DEPTNO NU

MBER

DEPTNAME VARCHAR2(10)

LOCATION VARCHAR2(10)

DESIGNATION VARCHAR2(10)

1. **Insert values into thetable.**

SQL> insert into department values(&deptno,'&deptname','&location','&designation'); Enter value for deptno: 9

Enter value for deptname: accounting Enter value for location: hyderabad Enter value for designation: manager

old 1: insert into department values(&deptno,'&deptname','&location','&designation') new 1: insert into department values(9,'accounting','hyderabad','manager')

1 row created. SQL> /

Enter value for deptno: 10

Enter value for deptname: research Enter value for location: chennai Enter value for designation: professor

old 1: insert into department values(&deptno,'&deptname','&location','&designation') new 1: insert into department values(10,'research','chennai','professor')

1 row created. SQL> /

Enter value for deptno: 11 Enter value for deptname:sales

Enter value for location: banglore Enter value for designation: salesman

old 1: insert into department values(&deptno,'&deptname','&location','&designation') new 1: insert into department values(11,'sales','banglore','salesman')

1 row created. SQL> /

Enter value for deptno: 12

Enter value for deptname: operations Enter value for location: mumbai Enter value for designation: operator

old 1: insert into department values(&deptno,'&deptname','&location','&designation') new

1: insert into department values(12,'operations','mumbai','operator') 1 row created.

SQL> insert into department values(&deptno,'&deptname','&location','&designation'); Enter value for deptno: 9

Enter value for deptname: accounting Enter value for location: chennai Enter value for designation: manager

old 1: insert into department values (&deptno, '&deptname', '&location', '&designation')

new 1: insert into department values (9, 'accounting', 'chennai', 'manager')

1 row created.

SQL> select \* from department ;

DEPTNO DEPTNAME LOCATION DESIGNATION

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | accounting | hyderabad | manager |
| 10 | research | chennai | professor |
| 11 | sales | banglore | salesman |
| 12 | operations | mumbai | operator |
| 9 | accounting | chennai | manager |

**a. List the records of dept table grouped bydeptno.**

SQL> select deptno,deptname from department group by deptno,deptname;

DEPTNO DEPTNAME

9 accounting

12 operations

1. research
2. sales
3. **Update the record where deptno is9.**

SQL> update department set designation='accountant' where deptno=9; 2 rows updated.

SQL> select \* from department;

DEPTNO DEPTNAME LOCATION DESIGNATION

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | Accounting | hyderabad | accountant |
| 10 | Research | chennai | professor |
| 11 | Sales | banglore | salesman |
| 12 | Operations | mumbai | operator |
| 9 | Accounting | chennai | accountant |

1. **Delete any column data from the table.**

SQL> alter table department drop(designation); Table altered.

SQL> select \* from department; DEPTNO DEPTNAMELOCATION

|  |  |  |
| --- | --- | --- |
| 9 | accounting | hyderabad |
| 10 | research | chennai |
| 11 | sales | bangalore |
| 12 | operations | mumbai |
| 9 | accounting | Chennai |

**LAB-4**

**Aim:-**

**a. Create a user and grant all permissions to the user.**

* 1. **Insert the any three records in the employee table and use rollback. Check the result.**
  2. **Add primary key constraint and not null constraint to the employee table.**
  3. **Insert null values to the employee table and verify the result.**

**SOLUTION:**

1. **create a user and grant all permissions to the user.**

CONNECT <USER-NAME>/<PASSWORD>@<DATABASE NAME>;

--Create user query

CREATE USER <USER NAME> IDENTIFIED BY <PASSWORD>;

--Provide roles

GRANT CONNECT, RESOURCE, DBA TO <USER NAME>;

--Assigning privileges

GRANT CREATE SESSION GRANT ANY PRIVILEGE TO

<USER NAME>; GRANT UNLIMITED TABLESPACE TO

<USER NAME>;

--Provide access to tables.

GRANT SELECT, UPDATE, INSERT, DELETE ON <TABLE NAME> TO <USER NAME>;

1. **Insert the any three records in the employee table and use rollback. Check the result.**

SQL> SELECT \* FROM EMPLOYEE;

EMPNO ENAME JOB MANAGER\_NO SAL COMMISSION

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 101 | abhi | manager | 1234 | 1100 | 70 |
| 102 | rohith | analyst | 2345 | 9000 | 65 |
| 103 | david | trainee | 3456 | 9000 | 65 |
| 104 | rahul | clerk | 4567 | 7000 | 55 |

SQL> insert into employee values (&empno, '&ename', '&job', &manager\_no, &sal, &commission); Enter value for empno: 105

Enter value for ename: aravind Enter value for job: salesman Enter value for manager\_no: 5678 Enter value for sal: 5000

Enter value for commission: 50

old 1: insert into employee values(&empno,'&ename','&job',&manager\_no,&sal,&commission) new 1: insert into employee values(105,'aravind','salesman',5678,5000,50)

1 row created.

SQL> rollback; Rollback complete. SQL> SELECT \* FROM EMPLOYEE;

EMPNO ENAME JOB MANAGER\_NO SAL COMMISSION

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **101** | **abhi** | **manager** |  | **1234** | **1100** |  | **70** |
| **102** | **rohith** | **analyst** |  | **2345** | **9000** |  | **65** |
| **103** | **david** | **trainee** |  | **3456** | **9000** |  | **65** |
| **104** | **rahul** | **clerk** |  | **4567** | **7000** |  | **55** |

1. **Add primary key constraint and not null constraint to the employee table.** SQL> alter table employee modify (empno number primary key, ename varchar2(10) not null); Table altered.

SQL> desc employee;

Name Null? Type

- EMPNO NOTNULL NUMBER

ENAME NOT NULL VARCHAR2(10) JOB VARCHAR2(10) MANAGER\_NO NUMBER SALNUMBER

COMMISSION NUMBER

1. **Insert null values to the employee table and verify the result.**

SQL> desc employee;

Name Null? Type

------------------- EMPNO NOTNULL NUMBER

ENAME NOTNULL VARCHAR2(10)

JOB NOTNULL VARCHAR2(10) MANAGER\_NO SAL NUMBER NOTNULL NUMBER

COMMISSION NUMBER

SQL> insert into employee values(&empno,'&ename','&job',&manager\_no,&sal,&commission); Enter value for empno: 105

Enter value for ename: mohith Enter value for job: salesman Enter value for manager\_no: 5678 Enter value for sal: null

Enter value for commission: 50 old 1: insert into employee

values(&empno,'&ename','&job',&manager\_no,&sal,&commission) new 1: insert into employee values(105,'mohith','salesman',5678,null,50)

insert into employee values(105,'mohith','salesman',5678,null,50)

\*

**a. Create a user and grant all permissions to the user. Insert values in the department table and use commit.**

**Add constraints like unique and not null to the department table. Insert repeated values and null values into the table.**

**SOLUTION:**

**create a user and grant all permissions to the user.**

CONNECT <USER-NAME>/<PASSWORD>@<DATABASE NAME>;

--Create user query

CREATE USER <USER NAME> IDENTIFIED BY <PASSWORD>;

--Provide roles

GRANT CONNECT,RESOURCE,DBA TO <USER NAME>;

--Assigning privileges

GRANT CREATE SESSION GRANT ANY PRIVILEGE TO <USER NAME>; GRANT UNLIMITED TABLESPACE TO <USER NAME>;

--Provide access to tables.

GRANT SELECT, UPDATE, INSERT, DELETE ON <TABLE NAME> TO

<USER NAME>;

**Insert values in the department table and use commit.**

SQL> insert into department values(&deptno,'&deptname','&location'); Enter value for deptno: 13

Enter value for deptname: sales Enter value for location: delhi

old 1: insert into department values(&deptno,'&deptname','&location') new 1: insert into department values(13,'sales','delhi')

1 row created.

SQL> commit; Commit complete.

SQL> select \* from department; DEPTNO DEPTNAMELOCATION

accounting hyderabad research chennai

sales banglore

operations mumbai

9 accounting chennai

13 sales delhi 6 rows selected.

1. **Add constraints like unique and not null to the departmenttable.**

SQL> alter table department modify(deptnonumberunique); Table altered.

SQL> alter table department modify(location varchar2(10) notnull); Table altered.

SQL> DESC DEPARTMENT;

Name Null? Type

DEPTNO NUMBER

DEPTNAME

VARCHAR2(10)

LOCATION NOT NULL VARCHAR2(10)

1. **Insert repeated values and null values into thetable.**

SQL> insert into department values(&deptno,'&deptname','&location'); Enter value for deptno: 10 Enter value for deptname: research Enter value for location:

old 1: insert into department values(&deptno,'&deptname','&location') new 1: insert into department values(10,'research','')

insert into department values(10,'research','')

SQL> insert into department values(&deptno,'&deptname','&location'); Enter value for deptno: 10 Enter value for deptname: research Enter value for location: hyderabad

old 1: insert into department values(&deptno,'&deptname','&location') new 1: insert into department values(10,'research','hyderabad')

**LAB-5**

**AIM :-**

**Queries using aggregate functions(COUNT,AVG,MIN,MAX,SU M),Group by,Orderby,Having.**

|  |  |  |  |
| --- | --- | --- | --- |
| **E\_id** | **E\_name** | **Age** | **Salary** |
| 101 | Anu | 22 | 9000 |
| 102 | Shane | 29 | 8000 |
| 103 | Rohan | 34 | 6000 |
| 104 | Scott | 44 | 10000 |
| 105 | Tiger | 35 | 8000 |
| 106 | Alex | 27 | 7000 |
| 107 | Abhi | 29 | 8000 |

1. **Create Employee table containing allRecords.**

SQL> create table emp(eidnumber,ename varchar2(10),age, number, salary

number); Table created. SQL> desc emp;

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Null? | Type | |
|  |  |  |  |
| EID | -- NUMBER | |
| ENAME AGE |  | VARCHAR2(10  )  NUMBER | |
| SALARY |  | NUMBER | |

# Count number of employee names from employee table.

SQL> select count(ename) from emp; COUNT(ENAME)

-

7

# Find the Maximum age from employee table.

SQL> select max(age) from emp;

MAX(AGE)

44

# Find the Minimum age from employee table.

SQL> select min(age)

from emp;

MIN(AGE)

22

Display the Sum of age employeetable. SQL> select sum(age) from emp; SUM(AGE)

220

Display the Average of age from Employeetable. SQL> select avg(age) from emp; AVG(AGE)

31.4285714

Create a View for age in employeetable.

SQL> create or replace view A as select age from emp where age<30; View created. Displayviews

SQL> select \* from A; AGE

22 29

27

29

# Find grouped salaries of employees.(group by clause)

SQL> select salary from emp group by salary; SALARY

9000

10000

8000

6000

7000

# (vi).Find salaries of employee in Ascending Order.(order by clause)

SQL> select ename,salary from emp order by salary;

|  |  |
| --- | --- |
| ENAME | SALARY |
| - | - |
| rohan | 6000 |
| alex | 7000 |
| shane | 8000 |
| abhi | 8000 |
| tiger | 8000 |
| anu | 9000 |
| scott | 10000 |

1. rows selected.

**(Vii) Find salaries of employee in Descending Order.**

SQL> select ename,salary from emp order by salary desc;

ENAME SALARY

- scott 10000

anu 9000

|  |  |
| --- | --- |
| Shane | 8000 |
| Abhi | 8000 |
| Tiger | 8000 |
| Alex | 7000 |
| Rohan | 6000 |

1. rows selected.

# (viii) Having Clause.

SQL> select ename,salary from emp where age<29 group by ename,salary having salary<10000;

ENAME SALARY

-

alex 7000

anu 9000

**LAB-6**

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1

EXPERIMENT 6

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

AIM

SQL commands to perform Join operations.

REQUIREMENTS

paiza.io (Online SQL), Database.

THEORY

A JOIN clause is used to combine rows from two or more tables, based on a related column between

them.

o(INNER) JOIN: Returns records that have matching values in both tables.

oLEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the

right table.

oRIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the

left table.

oFULL (OUTER) JOIN: Return all records when there is a match in either left or right tab

**Aim:-** SQL commands to perform Join operations.

THEORY: A join clause is used to combine rows from two or more tables, based on a related column between them.

* (INNER) JOIN: Returns records that have matching values in both tables.
* LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table.
* RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table.
* FULL (OUTER) JOIN: Return all records when there is a match in either left or right table.

1. Find customers name, rating given and commission obtained by common salesnumber

select c.cname, c.rating, s.comm from customers as c inner join

salespeople as s on c.snum = s.snum;

2. Display amt of order, order date and name of the salesperson who made the sale left join

select o.amt, o.odate, s.sname from orders as o left join salespeople as s on o.snum=s.snum;

3. Display all Order number, Order Amount, and Order Date commissions of the sales person

select o.onum, O.amt, o.odate, s.comm from orders as o left join salespeople as s on o.snum=s.snum;

4. Display all salespeople details and rating of purchase by the customers

select c.rating, s.snum, s.sname, s.city, s.comm from customers as c right join salespeople as s on c.snum=s.snum;

5. Display all customers and order details using full join

select \* from customers as c left join orders as o on c.snum= o.snum union

select \* from customers as c right join orders as o on c.snum= o.snum;

**LAB-7**

**Aim:-**

**SET Operations**

SQL supports few Set operations which can be performed on the table data.

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

**Syntax**

SELECT column\_name FROM table1 UNION

SELECT column\_name FROM table2;

**The First table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**The Second table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

Union SQL query will be: SELECT \* FROM First UNION

SELECT \* FROM Second;

The resultset table will look like:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

1. **Union All**

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

**Syntax:**

SELECT column\_name FROM table1 UNION ALL

SELECT column\_name FROM table2;

**Example:** Using the above First and Second table. Union All query will be like:

SELECT \* FROM First UNION ALL

SELECT \* FROM Second;

The result set table will look like:

|  |  |
| --- | --- |
| **-ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

1. **Intersect**

* It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
* In the Intersect operation, the number of datatype and columns must be the same.
* It has no duplicates and it arranges the data in ascending order by default.

**Syntax**

SELECT column\_name FROM table1 INTERSECT

SELECT column\_name FROM table2;

**Example:**

**Using the above First and Second table.**

Intersect query will be:

SELECT \* FROM First INTERSECT

SELECT \* FROM Second;

The result set table will look like:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |

1. **Minus**

* It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
* It has no duplicates and data arranged in ascending order by default.

**Syntax:**

SELECT column\_name FROM table1 MINUS

SELECT column\_name FROM table2;

**Example**

**Using the above First and Second table.**

Minus query will be: SELECT \* FROM First MINUS

SELECT \* FROM Second;

The result set table will look like:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |

**LAB-8**

**Aim:- Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:**

**CUSTOMERS table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Alive | 24 | Khammam | 2000 |
| 2 | Bob | 27 | Kadappa | 3000 |
| 3 | Catri | 25 | Guntur | 4000 |
| 4 | Dena | 28 | Hyderabad | 5000 |
| 5 | Eeshwar | 27 | Kurnool | 6000 |
| 6 | Farooq | 28 | Nellur | 7000 |

CREATE OR REPLACE TRIGGER display\_salary\_changes BEFORE DELETE OR INSERT OR UPDATE ON customers FOR EACH ROW

WHEN (NEW.ID > 0) DECLARE

sal\_diff number; BEGIN

sal\_diff := :NEW .salary - :OLD .salary; dbms\_output.put\_line('Old salary: ' || :OLD.salary); dbms\_output.put\_line('New salary: ' || :NEW.salary); dbms\_output.put\_line('Salary difference: ' || sal\_diff);

END;

/

# Trigger created.

Here following two points are important and should be noted carefully:

OLD and NEW references are not available for table level triggers, rather you can use them for record leveltriggers.

If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

Above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using DELETE operation on the table.

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table:

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (7, 'Kriti', 22, 'HP', 7500.00 );

When a record is created in CUSTOMERS table, above create trigger display\_salary\_changeswill be fired and it will display the following result:

Old salary:

New salary: 7500 Salary difference:

# Convert employee name into uppercase whenever an employee record is inserted or updated. Trigger to fire before the insert orupdate.

SQL> create table Employee(

* 1. ID VARCHAR2(4 BYTE) NOTNULL,
  2. First\_Name VARCHAR2(10BYTE),
  3. Last\_Name VARCHAR2(10BYTE),
  4. Start\_Date DATE,
  5. End\_Date DATE,
  6. Salary NUMBER(8,2),
  7. City VARCHAR2(10BYTE),
  8. Description VARCHAR2(15

BYTE) 10)

11 /

Table created.

SQL> CREATE OR REPLACE TRIGGER employee\_ insert\_ update

BEFORE INSERT OR UPDATE ON employee FOR EACH ROW 4DECLARE

5 dup\_ flag INTEGER; 6BEGIN

--Force all employee names to upper case.

:NEW. first\_ name :=UPPER(:NEW. first\_name); 9END;

10 /

Trigger created.

SQL> insert into Employee(ID, First\_Name, Last\_Name, Start\_Date, End\_Date, Salary, City, Description)

2 values('01','Jason', 'Martin', to\_date('19960725','YYYYMMDD'), to\_date('20060725','YYYYMMDD'), 1234.56, 'Toronto', 'Programmer')

3 /

1. row created.

SQL> insert into Employee(ID, First\_Name, Last\_Name, Start\_Date, End\_Date, Salary, City, Description)

1. values('02','Alison', 'Mathews', to\_date('19760321','YYYYMMDD'), to\_date('19860221','YYYYMMDD'), 6661.78, 'Vancouver','Tester')

3 /

1. row created.

SQL> insert into Employee(ID, First\_Name, Last\_Name, Start\_Date, End\_Date, Salary, City, Description)

1. values('03','James', 'Smith', to\_date('19781212','YYYYMMDD'), to\_date('19900315','YYYYMMDD'), 6544.78, 'Vancouver','Tester')

3 /

1. row created.

SQL> insert into Employee(ID, First\_Name, Last\_Name, Start\_Date, End\_Date, Salary, City, Description)

1. values('04','Celia', 'Rice', to\_date('19821024','YYYYMMDD'), to\_date('19990421','YYYYMMDD'), 2344.78,'Vancouver','Manager')

3 /

1 row created.

SQL> insert into Employee(ID, First\_Name, Last\_Name, Start\_Date, End\_Date, Salary, City, Description)

2 values('05','Robert', 'Black', to\_date('19840115','YYYYMMDD'), to\_date('19980808','YYYYMMDD'), 2334.78, 'Vancouver','Tester')

3 /

1 row created.

SQL> insert into Employee(ID, First\_Name, Last\_Name, Start\_Date, End\_Date, Salary, City, Description)

2 values('06','Linda', 'Green', to\_date('19870730','YYYYMMDD'), to\_date('19960104','YYYYMMDD'), 4322.78,'New York','Tester')

3 /

1 row created.

SQL> insert into Employee(ID, First\_Name, Last\_Name, Start\_Date, End\_Date, Salary, City, Description)

2 values('07','David', 'Larry', to\_date('19901231','YYYYMMDD'), to\_date('19980212','YYYYMMDD'), 7897.78,'New York', 'Manager')

3 /

1 row created.

SQL> insert into Employee(ID, First\_Name, Last\_Name, Start\_Date, End\_Date, Salary, City, Description)

2 values('08','James', 'Cat', to\_date('19960917','YYYYMMDD'),

to\_date('20020415','YYYYMMDD'),

1232.78,'Vancouver','Tester') 3 /

1 row created. SQL> select

\* from

Employee 2

/ID FIRST\_NAME LAST\_NAMESTART\_DATEND\_DATE SALARYCITY

DESCRIPTION

-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 01 | JASON | Martin | 25-JUL-96 | 25-JUL-06 1234.56 | Toronto  Programme  r |
| 02 | ALISON | Mathews | 21-MAR-76 | 21-FEB-86 6661.78 | Vancouver Tester |
| 03 | JAMES | Smith | 12-DEC-78 | 15-MAR-90  6544.78 | Vancouver Tester |
| 04 | CELIA | Rice | 24-OCT-82 | 21-APR-99 2344.78 | Vancouver Manager |
| 05 | ROBER  T | Black | 15-JAN-84 | 08-AUG-98 2334.78 | Vancouver Tester |
| 06 | LINDA | Green | 30-JUL-87 | 04-JAN-96 4322.78 | New YorkTester |
| 07 | DAVID | Larry | 31-DEC-90 | 12-FEB-98 7897.78 | New York Manager |
| 08 | JAMES | Cat | 17-SEP-96 | 15-APR-02 1232.78 | Vancouver Tester |

8 rows selected.

SQL> drop table Employee 2 / Table dropped.

# Trigger before deleting a record from emp table. Trigger will insert the row to be deleted into another table and also record the user who has deleted therecord.

SQL> CREATE OR REPLACE TRIGGERemployee\_before\_delete

1. BEFOREDELETE
2. ON employee
3. FOR EACHROW
4. DECLARE
5. v\_usernamevarchar2(10);
6. BEGIN
7. -- Find username of person performing the DELETE on thetable
8. SELECT user INTOv\_username
9. FROMdual;
10. -- Insert record into audit table
11. INSERT INTO employee\_audit (id, salary, delete\_date,deleted\_by)
12. VALUES (:old.id,:old.salary, sysdate, v\_username);
13. END;

15 /

Trigger created.

SQL> delete from employee; 8 rows deleted. SQL> select \* from employee\_audit;

ID SALARY DELETE \_DADELETED\_BY

---- ---------- --------- --------------- 01 1234.56 09-SEP-06JAVA2S 02 6661.78 09-SEP-06JAVA2S

03 6544.78 09-SEP-06JAVA2S

04 2344.78 09-SEP-06JAVA2S

05 2334.78 09-SEP-06JAVA2S

06 4322.78 09-SEP-06JAVA2S

07 7897.78 09-SEP-06JAVA2S

08 1232.78 09-SEP-06JAVA2S

8 rows selected.

SQL>drop tableemployee\_audit; Table dropped.