

Department of INFORMATION TECHNOLOGY

**22ITL41 Database Management Systems Laboratory Record**

Name Programme

Branch Section Semester

Roll No

***Certify that this is bonafide record of work done by the above student of the* 22ITL41- DATABASE MANAGEMENT SYSTEMS LABORATORY**

***during the year* 2023 - 2024.**

Submitted for the Examination held on

**Signature of the**

**Lab Incharge Head of the Department**

**Examiner 1 Examiner 2**

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXP.**  **NO.** |  | **Name of Experiment** | **Marks (60)** | **Signature** |
| 1. |  | DATA DEFINITION LANGUAGE COMMANDS, INTEGRITY CONSTRAINTS |  |  |
| 2. |  | DATA MANIPULATION LANGUAGE, DATA CONTROL LANGUAGE COMMANDS AND TCL COMMANDS |  |  |
| 3. |  | NESTED QUERIES |  |  |
| 4. |  | JOIN OPERATIONS |  |  |
| 5. |  | VIEWS AND INDEX |  |  |
| 6. |  | STRING FUNCTIONS (Additional) |  |  |
| 7. |  | AGGREGATE FUNCTIONS (Additional) |  |  |
| 8. |  | ORDER\_BY, GROUP\_BY AND HAVING CLAUSES  (Additional) |  |  |
| 9. |  | PL/SQL STATEMENTS |  |  |
| 10. |  | CURSORS |  |  |
| 11. |  | TRIGGERS |  |  |
| 12. |  | PROCEDURES AND FUNCTIONS |  |  |
| 13. |  | MINI PROJECT |  |  |

|  |  |
| --- | --- |
| **EX NO : 1**  **29 -02 -24** | **Data Definition Language commands and Integrity Constraints** |

**AIM:**

To execute DDL commands and Integrity constraints.

**DATA DICTIONARY**

SQL> select \* from tab;

TNAME TABTYPE CLUSTERID

------------------------------ ------- ----------

CUSTOMER TABLE

MANAGE TABLE

**Table creation:**

SQL> create table student(sno number(5) primary key,sname varchar(20) not null,phone number(10) unique,gender char(1) check(gender='M' or gender='F'));

Table created.

**SCHEMA OF THE TABLE**

SQL> desc student;

Name Null? Type

----------------------------------------- -------- ----------------------------

SNO NOT NULL NUMBER(5)

SNAME NOT NULL VARCHAR2(20)

PHONE NUMBER(10)

GENDER CHAR(1)

**Insertion of data into the table:**

1. **Manually:**
2. Syntax and output:

SQL> insert into student values(10001,'Immanuvel',8825503860,'M');

1 row created.

1. Syntax and output:

SQL> insert into student values(10002,'Jothi Sree',9345527195,'M');

1 row created.

1. **Through System:**

Syntax and output:

SQL> insert into student values(&sno,'&sname',&phone,'&gender');

Enter value for sno: 10003

Enter value for sname: Kavya

Enter value for phone: 9567342768

Enter value for gender: F

old 1: insert into student values(&sno,'&sname',&phone,'&gender')

new 1: insert into student values(10003,'Kavya',9567342768,'F')

1 row created.

Syntax and output:

SQL> /

Enter value for sno: 10004

Enter value for sname: Karthikeyan

Enter value for phone: 9456784565

Enter value for gender: M

old 1: insert into student values(&sno,'&sname',&phone,'&gender')

new 1: insert into student values(10004,'Karthikeyan',9456784565,'M')

1 row created.

SQL> select \* from student;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8825503860 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

10004 Karthikeyan 9456784565 M

10005 Lalit kumar 8563478696 M

**Alter:**

Add attribute:

SQL> select \* from student;

ID NAME PHONE G

---------- -------------------- ---------- -

1 Immanuvel 8825503860 M

2 Jothi Sree R S 9345527195 F

SQL> desc student;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NOT NULL NUMBER(5)

NAME NOT NULL VARCHAR2(20)

PHONE NUMBER(10)

GENDER CHAR(1)

**Adding:**

SQL> alter table student add(College varchar(25) default 'Kongu Engineering College');

Table altered.

SQL> select \* from student;

ID NAME PHONE G COLLEGE

---------- -------------------- ---------- - -------------------------

1 Immanuvel 8825503860 M Kongu Engineering College

2 Jothi Sree R S 9345527195 F Kongu Engineering College

SQL> desc student;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NOT NULL NUMBER(5)

NAME NOT NULL VARCHAR2(20)

PHONE NUMBER(10)

GENDER CHAR(1)

COLLEGE VARCHAR2(25)

**Drop Attribute:**

SQL> alter table student drop column college;

Table altered.

SQL> select \* from student;

ID NAME PHONE G

---------- -------------------- ---------- -

1 Immanuvel 8825503860 M

2 Jothi Sree R S 9345527195 F

SQL> desc student;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NOT NULL NUMBER(5)

NAME NOT NULL VARCHAR2(20)

PHONE NUMBER(10)

GENDER CHAR(1)

**Adding constraint after creating a table:**

**Table creation**

SQL> create table addconstraint(id number(2),name varchar(20));

Table created.

SQL> desc addconstraint;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NUMBER(2)

NAME VARCHAR2(20)

**Adding constraint:**

SQL> desc addconstraint;

Name Null? Type

----------------------------------------- -------- ----------------------------

**ID NUMBER(2)**

NAME VARCHAR2(20)

SQL> alter table addconstraint add constraint pk1 primary key(id);

Table altered.

SQL> desc addconstraint;

Name Null? Type

----------------------------------------- -------- ----------------------------

**ID NOT NULL NUMBER(2)**

NAME VARCHAR2(20)

**Dropping the constraint:**

SQL> alter table addconstraint drop constraint pk1;

Table altered.

SQL> desc addconstraint;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NUMBER(2)

NAME VARCHAR2(20)

**Modify:**

Modify datatype:

SQL> alter table student modify gender varchar(1);

Table altered.

**Modify datasize:**

SQL> alter table student modify phone number(5);

alter table student modify phone number(5)

\*

ERROR at line 1:

ORA-01440: column to be modified must be empty to decrease precision or scale

SQL> alter table student modify gender varchar(3);

Table altered.

SQL> desc student;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NOT NULL NUMBER(5)

NAME NOT NULL VARCHAR2(20)

PHONE NUMBER(10)

GENDER VARCHAR2(3)

**Truncate:**

Before truncate:

SQL> select \* from student;

ID NAME PHONE G

---------- -------------------- ---------- -

1 Immanuvel 8825503860 M

2 Jothi Sree R S 9345527195 F

SQL> truncate table student;

Table truncated.

View student table:

SQL> select \* from student;

no rows selected

SQL> desc student;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NOT NULL NUMBER(5)

NAME NOT NULL VARCHAR2(25)

CGPA NOT NULL NUMBER(4,2)

PHONE NUMBER(10)

GENDER CHAR(1)

**Rename:**

SQL> desc student;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NOT NULL NUMBER(5)

NAME NOT NULL VARCHAR2(20)

PHONE NUMBER(10)

GENDER VARCHAR2(1)

SQL> rename student to customers;

Table renamed.

SQL> desc student;

ERROR:

ORA-04043: object student does not exist

SQL> desc customers;

Name Null? Type

----------------------------------------- -------- ----------------------------

ID NOT NULL NUMBER(5)

NAME NOT NULL VARCHAR2(20)

PHONE NUMBER(10)

GENDER VARCHAR2(1)

**To check the constraints:**

**1.Primary key:**

1.Syntax and output : (Repetition)

SQL> insert into student values(10005,'Madanika',9345527195,'F');

insert into student values(10005,'Madanika',9345527195,'F')

\*

ERROR at line 1:

ORA-00001: unique constraint (STUDENT.SYS\_C004038) violated

2. Syntax and output: (Missing)

SQL> insert into student values('','Madanika',9345527195,'F');

insert into student values('','Madanika',9345527195,'F')

\*

ERROR at line 1:

ORA-01400: cannot insert NULL into ("STUDENT"."STUDENT"."SNO")

**2.Unique:**

Syntax and output:

SQL> insert into student values('10005','Madanika',9345527195,'F');

insert into student values('10005','Madanika',9345527195,'F')

\*

ERROR at line 1:

ORA-00001: unique constraint (STUDENT.SYS\_C004038) violated

**3.Not null:**

Syntax and output:

SQL> insert into student values('','Madanika',9345527195,'F');

insert into student values('','Madanika',9345527195,'F')

\*

ERROR at line 1:

ORA-01400: cannot insert NULL into ("STUDENT"."STUDENT"."SNO")

**4.Check:**

Syntax and output:

SQL> insert into student values('10006','Madanika',9345527195,'O');

insert into student values('10006','Madanika',9345527195,'O')

\*

ERROR at line 1:

ORA-02290: check constraint (STUDENT.SYS\_C004037) violated

**Insertion with reference to the parent table:**

Creation of parent:

SQL> create table student(id number(5) primary key,name varchar(20) not null,phone number(10), gender char(1) check(gender='M' or gender='F'));

Table created.

Creation of Child:

SQL> create table scholarship(name varchar(20) primary key,scholar\_number number(10),id references student(id));

Table created.

Insertion into student table:

SQL> insert into student values(&id,'&name',&phone,'&gender');

Enter value for id: 1

Enter value for name: Immanuvel

Enter value for phone: 8825503860

Enter value for gender: M

old 1: insert into student values(&id,'&name',&phone,'&gender')

new 1: insert into student values(1,'Immanuvel',8825503860,'M')

1 row created.

SQL> /

Enter value for id: 2

Enter value for name: Jothi Sree R S

Enter value for phone: 9345527195

Enter value for gender: F

old 1: insert into student values(&id,'&name',&phone,'&gender')

new 1: insert into student values(2,'Jothi Sree R S',9345527195,'F')

1 row created.

Insertion into scholarship:

SQL> insert into scholarship values('&name',&scholar\_number,&id);

Enter value for name: Jothi Sree R S

Enter value for scholar\_number: 1001

Enter value for id: 2

old 1: insert into scholarship values('&name',&scholar\_number,&id)

new 1: insert into scholarship values('Jothi Sree R S',1001,2)

1 row created.

SQL> /

Enter value for name: Immanuvel

Enter value for scholar\_number: 1008

Enter value for id: 1

old 1: insert into scholarship values('&name',&scholar\_number,&id)

new 1: insert into scholarship values('Immanuvel',1008,1)

1 row created.

**When a record with id not present in parent table:**

SQL> /

Enter value for name: Kavya

Enter value for scholar\_number: 1009

Enter value for id: 3

old 1: insert into scholarship values('&name',&scholar\_number,&id)

new 1: insert into scholarship values('Kavya',1009,3)

insert into scholarship values('Kavya',1009,3)

\*

ERROR at line 1:

ORA-02291: integrity constraint (22ITR035.SYS\_C004032) violated - parent key

not found

**Deletion:**

**Delete from parent first:**

SQL> delete from student where id=1;

delete from student where id=1

\*

ERROR at line 1:

ORA-02292: integrity constraint (22ITR035.SYS\_C004032) violated - child record

Found

**Delete from child and then from parent:**

SQL> delete from scholarship where id=1;

1 row deleted.

SQL> delete from student where id=1;

1 row deleted.

After delete:

SQL> select \* from student;

ID NAME PHONE G

---------- -------------------- ---------- -

2 Jothi Sree R S 9345527195 F

SQL> select \* from scholarship;

NAME SCHOLAR\_NUMBER ID

-------------------- -------------- ----------

Jothi Sree 1001 2

**On delete cascade:**

Creation of child table:

SQL> create table scholarship(name varchar(20) primary key,scholar\_number number(10),id references student(id) on delete cascade);

Table created.

After Insertion of records:

SQL> select \* from student;

ID NAME PHONE G

---------- -------------------- ---------- -

2 Jothi Sree R S 9345527195 F

1 Immanuvel 8825503860 M

SQL> select \* from scholarship;

NAME SCHOLAR\_NUMBER ID

-------------------- -------------- ----------

Jothi Sree 1001 2

Immanuvel 1008 1

Deleting in parent deletes child too:

SQL> delete from student where id=1;

1 row deleted.

Current records:

SQL> select \* from student;

ID NAME PHONE G

---------- -------------------- ---------- -

2 Jothi Sree R S 9345527195 F

SQL> select \* from scholarship;

NAME SCHOLAR\_NUMBER ID

-------------------- -------------- ----------

Jothi Sree 1001 2

**On delete set null:**

Creation of child table:

SQL> create table scholarship(name varchar(20) primary key,scholar\_number number(10),id references student(id) on delete set null);

After insertion of records:

SQL> select \* from student;

ID NAME PHONE G

---------- -------------------- ---------- -

2 Jothi Sree R S 9345527195 F

1 Immanuvel 8825503860 M

SQL> select \* from scholarship;

NAME SCHOLAR\_NUMBER ID

-------------------- -------------- ----------

Immanuvel 1008 1

Jothi Sree 1001 2

Deleting in parent makes the key null in child:

SQL> delete from student where id=1;

1 row deleted.

Records after delete:

SQL> select \* from student;

ID NAME PHONE G

---------- -------------------- ---------- -

2 Jothi Sree R S 9345527195 F

SQL> select \* from scholarship;

NAME SCHOLAR\_NUMBER ID

-------------------- -------------- ----------

Immanuvel 1008

Jothi Sree 1001 2

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT**

Thus Data Definition Language commands and Integrity Constraints were executed.

|  |  |
| --- | --- |
| **EX NO : 2**  **14-03-24** | **Data Manipulation Language, Data control Language and TCL commands** |

**AIM:**

To execute DML, DCL and TCL commands using SQL.

**Insertion of data into the table:**

**1.Manually:**

1.Syntax and output:

SQL> insert into student values(10001,'Immanuvel',8825503860,'M');

1. row created.

2.Syntax and output:

SQL> insert into student values(10002,'Jothi Sree',9345527195,'M');

1. row created.

**2.Through System:**

Syntax and output:

SQL> insert into student values(&sno,'&sname',&phone,'&gender');

Enter value for sno: 10003

Enter value for sname: Kavya

Enter value for phone: 9567342768

Enter value for gender: F

old 1: insert into student values(&sno,'&sname',&phone,'&gender')

new 1: insert into student values(10003,'Kavya',9567342768,'F')

1 row created.

Syntax and output:

SQL> /

Enter value for sno: 10004

Enter value for sname: Karthikeyan

Enter value for phone: 9456784565

Enter value for gender: M

old 1: insert into student values(&sno,'&sname',&phone,'&gender')

new 1: insert into student values(10004,'Karthikeyan',9456784565,'M')

1 row created.

SQL> select \* from student;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8825503860 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

10004 Karthikeyan 9456784565 M

**Update:**

SQL> update student set phone=8820045673 where id=10001;

1 row updated.

SQL> select \* from student;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8820045673 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

10004 Karthikeyan 9456784565 M

SQL> update salary set salary=salary+10000 where salary>21000.0;

1. rows updated.

SQL> select \* from salary;

ID SALARY

---------- ----------

1 40000.5

2 20000.3

4 77777.2

**Case :**

SQL> update salary set salary= case when salary<41000 then salary\*1.03 else salary\*1.10 end;

3 rows updated.

SQL> select \* from salary;

ID SALARY

---------- ----------

1 41200.52

2 20600.31

4 85554.92

**Delete:**

SQL> select \* from student;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8820045673 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

10004 Karthikeyan 9456784565 M

SQL> delete from student where sno=10004;

1 row deleted;

SQL> select \* from student;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8820045673 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

**Select:**

**\***

SQL> select \* from student;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8820045673 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

10004 Karthikeyan 9456784565 M

**where:**

SQL> select \* from student where sno=10001 ;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8820045673 M

1 row selected;

SQL> select \* from student where g=’M’ ;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8820045673 M

10002 Jothi Sree 9345527195 M

10004 Karthikeyan 9456784565 M

**Save the work in secondary memory:**

**Commit:**

Syntax:

SQL> commit;

Output:

Commit complete.

**//The work will be saved. Until now.**

**Rollback:**

Syntax:

SQL> rollback;

Output:

Rollback complete.

**Savepoint:**

Syntax:

SQL> savepoint A;

Output:

Savepoint created.

**Eg:**

**Commit:**

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8825503860 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

10004 Karthikeyan 9456784565 M

**Rollback of commit:**

1.

SQL> insert into student values(&sno,'&sname',&phone,'&g');

Enter value for sno: 10006

Enter value for sname: Hema dharunika

Enter value for phone: 9567746533

Enter value for g: F

old 1: insert into student values(&sno,'&sname',&phone,'&g')

new 1: insert into student values(10006,'Hema dharunika',9567746533,'F')

2.

SQL> /

Enter value for sno: 10007

Enter value for sname: Hema

Enter value for phone: 8995634487

Enter value for g: F

old 1: insert into student values(&sno,'&sname',&phone,'&g')

new 1: insert into student values(10007,'Hema',8995634487,'F')

1 row created.

**After insertion:**

SQL> select \* from student;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8825503860 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

10004 Karthikeyan 9456784565 M

10006 Hema dharunika 9567746533 F

10007 Hema 8995634487 F

**After rollback:**

SQL> rollback;

Rollback complete.

SQL> select \* from student;

SNO SNAME PHONE G

---------- -------------------- ---------- -

10001 Immanuvel 8825503860 M

10002 Jothi Sree 9345527195 M

10003 Kavya 9567342768 F

10004 Karthikeyan 9456784565 M

**Update:**

SQL> select \* from student;

ID NAME CGPA PHONE G

---------- ------------------------- ---------- ---------- -

1 Immanuvel 9 8825503860 M

2 Jothi Sree 8 9345527195 F

SQL> update student set cgpa=9 where id=2;

1 row updated.

SQL> select \* from student;

ID NAME CGPA PHONE G

---------- ------------------------- ---------- ---------- -

1 Immanuvel 9 8825503860 M

2 Jothi Sree 9 9345527195 F

**Commit:**

SQL> commit;

Commit complete.

**Roll back:**

SQL> rollback;

Rollback complete.

SQL> select \* from student;

ID NAME CGPA PHONE G

---------- ------------------------- ---------- ---------- -

1 Immanuvel 9 8825503860 M

2 Jothi Sree 8 9345527195 F

**Save point:**

SQL> select \* from student;

ID NAME CGPA PHONE G

---------- ------------------------- ---------- ---------- -

1 Immanuvel 9 8825503860 M

2 Jothi Sree 9 9345527195 F

SQL> savepoint A;

Savepoint created.

SQL> delete from student where id=2;

1 row deleted.

SQL> select \* from student;

ID NAME CGPA PHONE G

---------- ------------------------- ---------- ---------- -

2 Jothi Sree 9 9345527195 F

SQL> rollback to A;

Rollback complete.

SQL> select \* from student;

ID NAME CGPA PHONE G

---------- ------------------------- ---------- ---------- -

1 Immanuvel 9 8825503860 M

2 Jothi Sree 9 9345527195 F

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT**

Thus DML, DCL and TCL commands were executed using SQL.

|  |  |
| --- | --- |
| **EX NO : 3**  **22 -03 -24** | **Join Operation** |

**AIM:**

To execute various types of Join.

**Joins:**

Joins in SQL are commands which are used to combine rows from twoor more tables,based on a related column between those tables.

**Types:**

* Cross Join (or) Cartesian Product
* Inner join
* Natural Join
* Left Outer Join
* Right Outer Join
* Full Outer Join
* Self Join

**Create two tables instructor and teaches :**

SQL> create table instructor(id varchar(10) primary key,name varchar(20), deptname varchar(20),salary number(5));

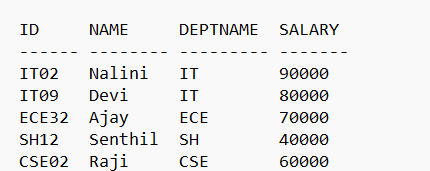
Table created.

SQL> create table teaches(id varchar(10),courseid varchar(20) primary key, secid varchar(1),semester varchar(1),year number(1));

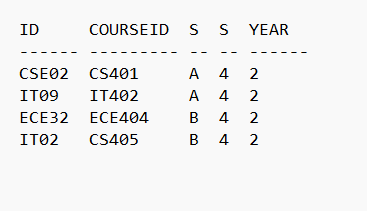
Table created.

**Display:**

SQL> select \* from instructor;



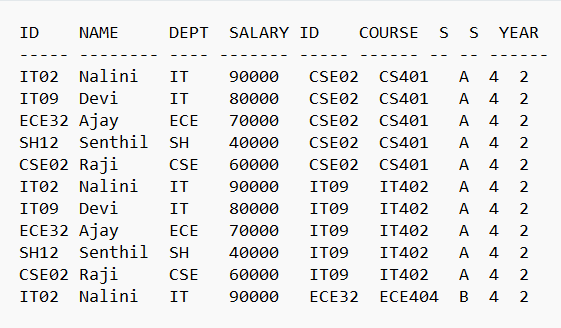
SQL> select \* from teaches;

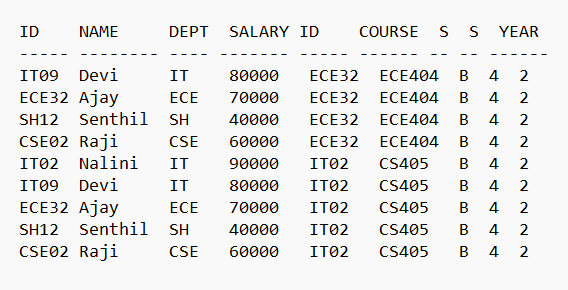


**Cross join:**

The CROSS JOIN keyword returns all records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

SQL> select \* from instructor,teaches;





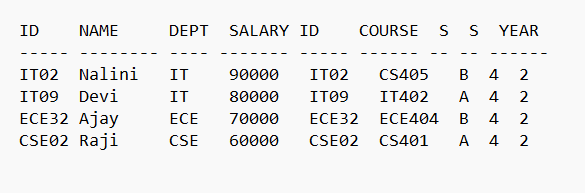
20 rows selected.

**Natural join:**

Displays pairs of rows from the two input relations that have the same value on all attributes that have the same name.

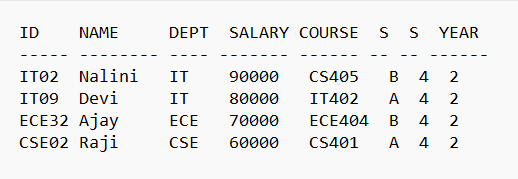
**Using condition:**

SQL> select \* from instructor i,teaches t where i.id = t.id;



**Using keyword:**

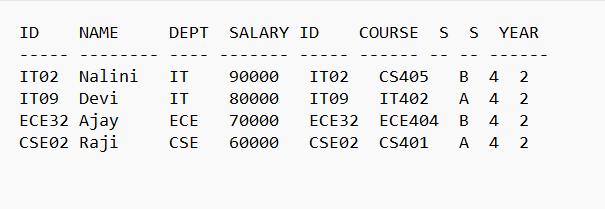
SQL> select \* from instructor natural join teaches;



**Inner join:**

Inner joins combine records from two tables whenever there are matching values in a field common to both tables.

SQL> select \* from instructor i inner join teaches t on i.id=t.id;



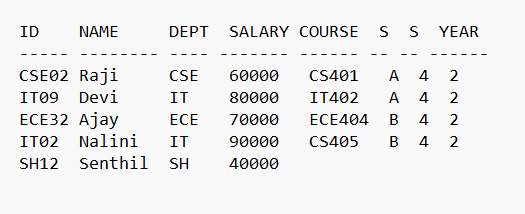
**Outer join:**

**Left:**

A left outer join returns all the rows that an inner join returns plus one row for each of the other rows in the first table that do not have a match in the second table.

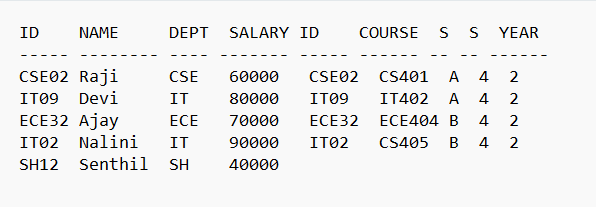
Using keyword:

SQL> select \* from instructor i natural left outer join teaches t;



Using +:

SQL> select \* from instructor i,teaches t where i.id=t.id(+);

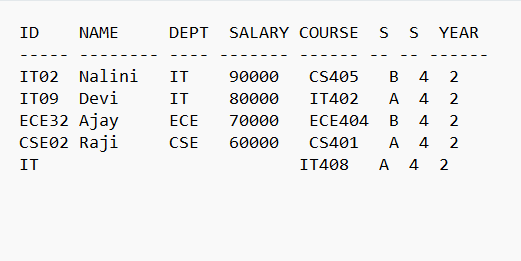


**Right:**

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is Null records from the left side, if there is no match.

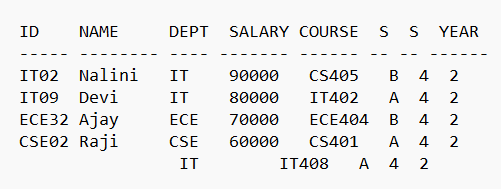
Using keyword:

SQL> select \* from instructor i natural right outer join teaches t;



**Using +:**

SQL> select \* from instructor i,teaches t where i.id(+)=t.id;

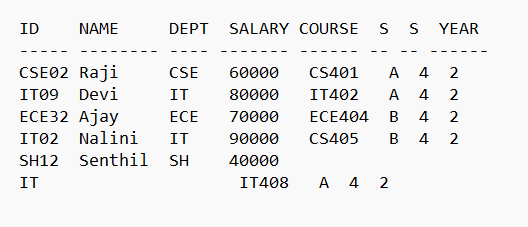


**Full outer join:**

Using keyword:

An full outer join is a method of combining tables so that the result includes unmatched rows of both tables. Joining two tables and want the result set to include unmatched rows from both tables, use a FULL OUTER JOIN clause. The matching is based on the join condition.

SQL> select \* from instructor i natural full outer join teaches t;



6 rows selected.

**Self-join:**

A self join is a regular join, but the table is joined with itself.

SQL> select s.sname as Student , l.sname as Leader from leader s,leader l where l.sid=s.lid;

STUDENT LEADER

---------- ----------

C A

B A

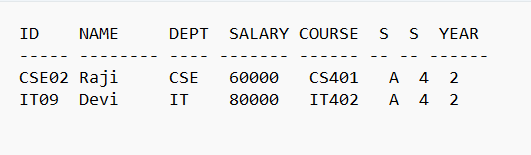
E B

D B

**Equi join:**

The EQUI JOIN in SQL performs a JOIN against a column of equality or the matching column(s) values that have the associated tables. Use an equal sign (=) as a comparison operator in our 'where' clause to refer to equality.

SQL> select \* from instructor i,teaches t where i.id=t.id and secid='A';



|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT**

Thus execution of various types of Join have been done successfully.

|  |  |
| --- | --- |
| **EX NO : 4**  **12 -04 -24** | **NESTED QUERIES** |

**AIM:**

To execute Nested Query using IN,NOT IN, Some, ALL, Exists and Not Exists keywords.

Also to execute scalar sub query in select and insert clause.

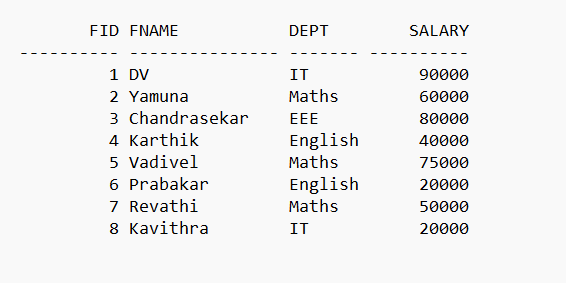
**Table creation:**

SQL> create table faculty(fid number(3) primary key,fname varchar(15) not null,dept varchar(7),salary number(6,1));

Table created.

**Insertion of some values and viewing:**

SQL> select \* from faculty;



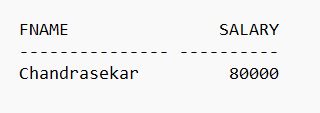
8 rows selected.

**EG for sub query:**

* **Find the faculty with second highest payment**

SQL> select fname,salary from faculty where salary=(select max(salary) from faculty where salary<(select max(salary) fro

m faculty));

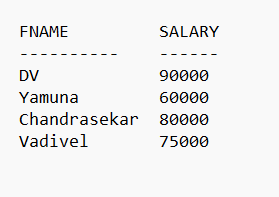


**Independent Sub- Query:**

**Using IN keyword:**

* **Find all the faculties with salary less than 50000.**

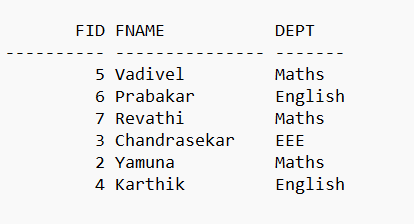
SQL> select fname,salary from faculty where salary in (select salary from faculty where salary>50000);

****

**Using NOT IN keyword:**

* **Find all the faculties other than IT department**

SQL> select fid,fname,dept from faculty where fname not in (select fname from faculty where dept='IT');

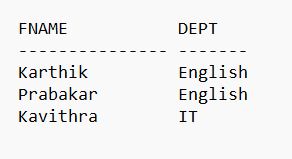


6 rows selected.

**Using ALL keyword:**

* **Find all the faculties and their department who has salary less than all of the faculties of Maths department.**

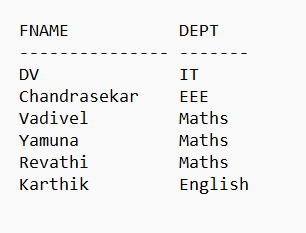
SQL> select fname,dept from faculty where salary <all(select salary from faculty where dept='Maths');



**Using SOME keyword:**

* **Find all the faculties who earn greater than atleast one of the faculty of IT department.**

SQL> select fname,dept from faculty where salary >some(select salary from faculty where dept='IT');

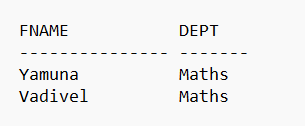


6 rows selected.

**Using Exists keyword:**

* **Find all the faculties with salary >50000 in Maths department**

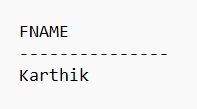
SQL> select fname,dept from faculty f where exists (select \* from faculty x where x.dept='Maths' and f.fname=x.fname and salary>50000);



**Using Not Exists keyword:**

* **Find all the faculties who work in English department with salary >30000**

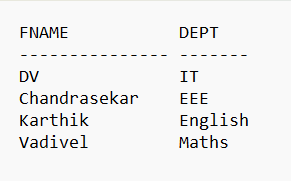
SQL> select fname from faculty f where dept='English' and not exists (select salary from faculty x where x.dept='English' and f.fname=x.fname and salary<=30000);



**Correlated sub-query:**

**=>Find all the faculties in each department with highest salary**

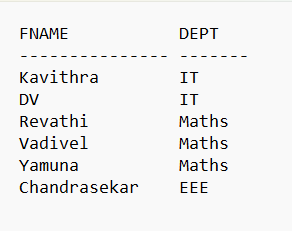
SQL> select fname,dept from faculty f where salary=(select max(salary) from faculty where f.dept=dept);



**Correlated sub-query using EXISTS operator:**

* **Select all the faculties with salary greater than 50000**

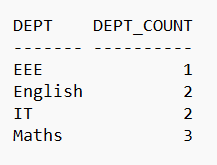
SQL> select fname,dept from faculty f where exists(select salary from faculty where f.dept=dept and salary>50000);



**Scalar Subquery using select clause:**

**=>Find the count of faculties of each department**

SQL> select dept, (select count(\*) from faculty where dept = f.dept) as dept\_count from faculty f group by dept order by dept\_count;



**Scalar subquery using insert clause:**

**=>Show of summary of records**

**Creation of a table:**

SQL> create table summary(No\_of\_faculy number(2),Amount\_spent number(7),Average\_salary number(6),Max\_salary number(6),Min\_salary number);

Table created.

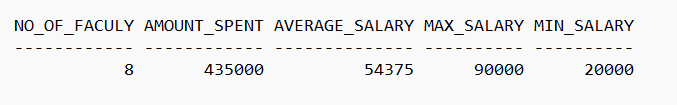
**Scalar subquery using insert clause:**

SQL> insert into summary values((select count(\*) from faculty),(select sum(salary) from faculty),(select avg(salary) from faculty),(select max(salary) from faculty),(select min(salary) from faculty));

1 row created.

**View the summary table:**

SQL> select \* from summary;



|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT**

Thus, Nested query using in,not in,some,all,exists and not exists key words is executed. Also the scalar sub query using select and insert clause are executed successfully.

|  |  |
| --- | --- |
| **EX NO : 5**  **12 -04 -24** | **VIEW AND INDEX** |

**Aim:**

To create different views and different index methods using SQL commands.

**VIEW**

Views are virtual tables created from original tables.

**INDEX**

Index are special lookup tables that need to be used by the database search engine to speed up the data retrieval

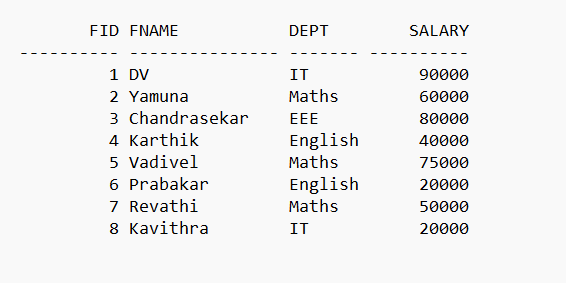
**Table creation:**

SQL> create table faculty(fid number(3) primary key,fname varchar(15) not null,dept varchar(7),salary number(6,1));

Table created.

**Insertion of some values and viewing:**

SQL> select \* from faculty;



8 rows selected.

**A. VIEW**

**CREATING A VIEW:**

**1. Creating the view by selecting the attributes from one table**

**=> Creating a view for all the faculties**

SQL> create view teacher\_view as select fid,fname from faculty;

View created.

SQL> select \* from teacher\_view;

FID FNAME

---------- ---------------

1 DV

2 Yamuna

3 Chandrasekar

4 Karthik

5 Vadivel

6 Prabakar

7 Revathi

8 Kavithra

8 rows selected.

**2. Creating the view by selecting the attributes from one table with predicate**

**=> Creating a view for all faculties of IT Department**

SQL> create view IT\_view as select fid,fname from faculty where dept='IT';

View created.

SQL> select \* from IT\_view;

FID FNAME

---------- ---------------

1 DV

8 Kavithra

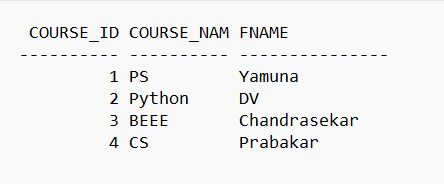
**3. Creating the view by selecting the attributes from two tables with predicate**

**=>Creating another table course and merging the attributes to create a view**

SQL> create view merged as select course\_id,course\_name,fname from faculty,course where faculty.fid=course.fid;

View created.

SQL> select \* from merged;

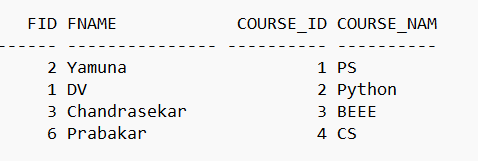


**4. CREATING THE VIEW using JOIN Clause**

SQL> create view joined\_view as select fid,fname,course\_id,course\_name from faculty natural join course;

View created.

SQL> select \* from joined\_view;



**5. CREATING THE VIEW using sub query**

SQL> create view sub\_view as select fid,fname from faculty where fname in(select fname from faculty where dept='Maths');

View created.

SQL> select \* from sub\_view;

FID FNAME

---------- ---------------

2 Yamuna

5 Vadivel

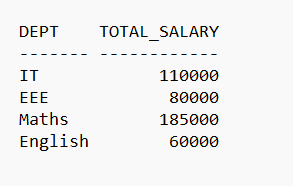
7 Revathi

**6. CREATING THE VIEW using aggregate and group by clause**

SQL> create view dept\_view(dept,total\_salary) as select dept,sum(salary) from faculty group by dept;

View created.

SQL> select \* from dept\_view;



**7. CREATING THE MATERIALIZED VIEW**

*Certain database systems allow view relations to be stored, but they make*

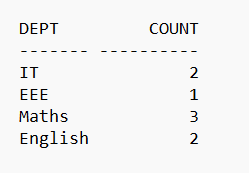
*sure that, if the actual relations used in the view definition change, the view is kept up-to-date.*

*Such views are called materialized views.*

SQL> create view count(dept,count) as select dept,count(\*) from faculty group by dept;

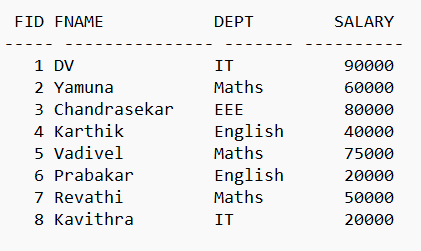
View created.

SQL> select \* from count;



**To show that the changes made on the main table reflects in the view**

SQL> select \* from faculty;

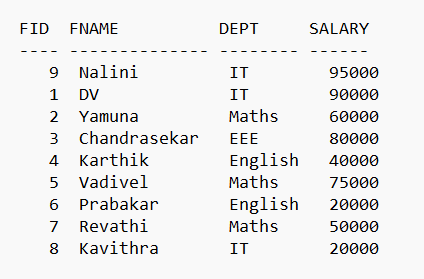


8 rows selected.

SQL> insert into faculty values(9,'Nalini','IT',95000);

1 row created.

SQL> select \* from faculty;



9 rows selected.

SQL> select \* from count;

DEPT COUNT

------- ----------

**IT 3**

EEE 1

Maths 3

English 2

**8. CREATING UPDATABLE VIEW**

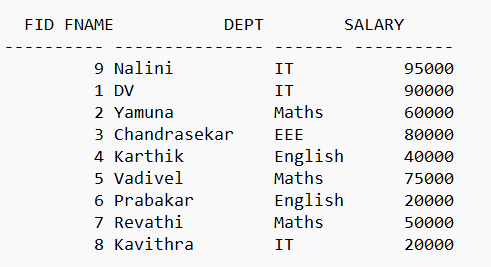
*The views that allows modification operations like INSERT,*

*UPDATE, or DELETE to be performed on the underlying tables through the view.*

SQL> create view updatable as select fid,fname,dept from faculty;

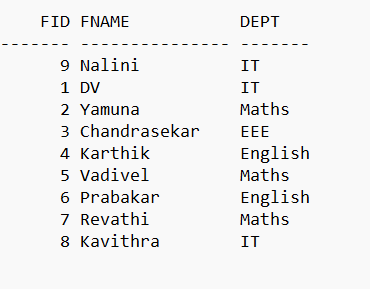
View created.

SQL> select \* from faculty;



9 rows selected.

SQL> select \* from updatable;

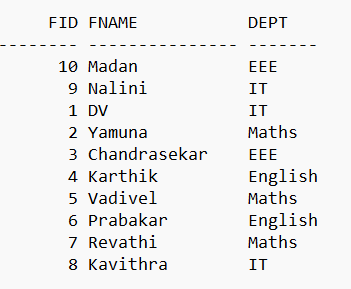


9 rows selected.

SQL> insert into updatable values(10,'Madan','EEE');

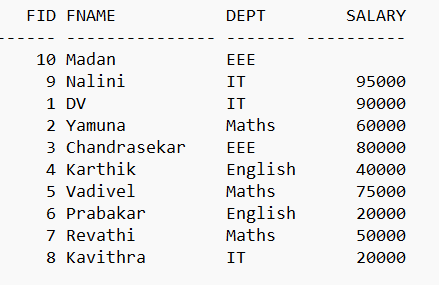
1 row created.

SQL> select \* from updatable;



10 rows selected.

SQL> select \* from faculty;



10 rows selected.

**Trying to insert a row in a view created using 2 tables using join**

SQL> select \* from joined\_view;

FID FNAME COURSE\_ID COURSE\_NAM

---------- --------------- ---------- ----------

2 Yamuna 1 PS

1 DV 2 Python

3 Chandrasekar 3 BEEE

6 Prabakar 4 CS

SQL> insert into joined\_view values(7,'Nalini',5,'C');

insert into joined\_view values(7,'Nalini',5,'C')

\*

ERROR at line 1:

ORA-01776: cannot modify more than one base table through a join view

**This is because:**

For the view to be updatable:

->The from clause has only one database relation.

->The select clause contains only attribute names of the relation and does not have any

expressions, aggregates, or distinct specification.

->Any attribute not listed in the select clause can be set to null; that is, it does not have a

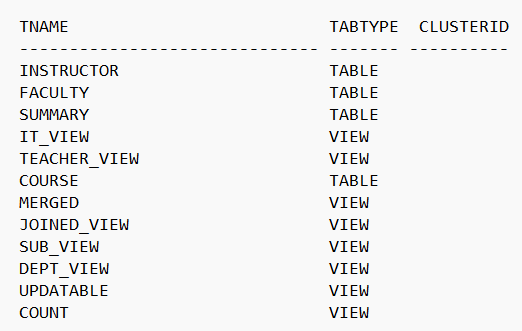
not null constraint and is not part of a primary key.

->The query does not have a group by or having clause.

->View should not be formed of subquery.

**9. Dropping the view**

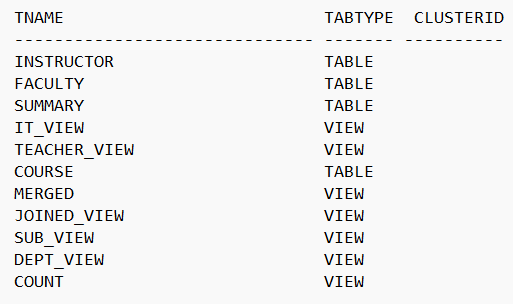
SQL> select \* from tab;



SQL> drop view updatable;

View dropped.

SQL> select \* from tab;

****

**10. Find Read only view or materialized view**

SQL> select view\_name, case when view\_type='MATERIALIZED VIEW' THEN 'Materialized View' Else 'Regular View' END AS view\_type FROM all\_views WHERE view\_name = 'IT\_VIEW';

VIEW\_NAME VIEW\_TYPE

------------------------------ -----------------

IT\_VIEW Regular View

SQL> select view\_name, case when view\_type='MATERIALIZED VIEW' THEN 'Materialized View' Else 'Regular View' END AS view\_type FROM all\_views WHERE view\_name = 'COUNT';

VIEW\_NAME VIEW\_TYPE

------------------------------ -----------------

COUNT Regular View

**B) INDEX**

*Index is used for quicker retrival of data.*

**1. Single column Index :**

SQL> create index teachers on faculty(fname);

Index created.

**2. Composite index :**

SQL> create index subjects on course(course\_id,course\_name);

Index created.

**4. Dropping an Index:**

SQL> drop index teachers;

Index dropped.

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT**

Hence views and indexes have been created successfully.

|  |  |
| --- | --- |
| **EX-NO :06**  **25.04.24** | **SQL STRING FUNCTIONS** |

**AIM:**

To perform sql string operations by using string functions.

**Table creation:**

SQL> create table details(id number(2),name varchar(10),father\_name varchar(10),city varchar(10));

Table created.

**Insertion of some values and viewing:**

SQL> select \* from details;

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

1 Immanuvel Rajappan Krishnagiri

2 Jothi Sree Ramasamy Gobi

3 Kavya Senthil Chennimalai

4 Abishek Aarumugam Thiruchengodu

5 Karthi Ranga Salem

**Finding the ascii of the character:**

SQL> select ascii('a') as ascii from dual;

ASCII

----------

97

**Find the ascii of the first character of the name in a table**

SQL> select ascii(name) from details;

ASCII(NAME)

-----------

73

74

75

65

75

**Find the char of a ascii:**

SQL> select chr(97) from dual;

C

-

a

SQL> select chr(78) from dual;

C

-

N

**Concat a substring into string**

SQL> select concat('Immanuvel','Rajappan') as Name from dual;

NAME

-----------------

ImmanuvelRajappan

**Concat a substring into string in a table**

SQL> select concat(name,' 2026 Passout') as student from details;

STUDENT

-----------------------

Immanuvel 2026 Passout

Jothi Sree 2026 Passout

Kavya 2026 Passout

Abishek 2026 Passout

Karthi 2026 Passout

**Length:**

SQL> select length(name) from details;

LENGTH(NAME)

------------

9

10

5

7

6

**Replacing:**

SQL> select replace(city,'Krishnagiri','Banglore') as city from details;

CITY

----------------------------

Banglore

Gobi

Chennimalai

Thiruchengodu

Salem

**Find the position of a character in a string:**

SQL> select name,instr(name,'i') as I\_position from details;

NAME I\_POSITION

---------- ----------

Immanuvel 0

Jothi Sree 5

Kavya 0

Abishek 3

Karthi 6

**Reversing the string:**

SQL> SELECT REVERSE('Immanuvel') as REVERSE from dual;

REVERSE

---------

levunammI

SQL> SELECT id,REVERSE(name) as reversed from details;

ID REVERSED

---------- ----------

1 levunammI

2 eerS ihtoJ

3 ayvaK

4 kehsibA

5 ihtraK

**Translate:**

SQL> select translate('Monday','Monday','Sunday') from dual;

TRANSL

------

Sunday

SQL> select translate('3\*[2+1]/{8-4}', '[]{}', '()()') as valid from dual;

VALID

-------------

3\*(2+1)/(8-4)

SQL> select translate(city,'abcde','vwxyz') as city from details;

CITY

---------------

Krishnvgiri

Gowi

Chznnimvlvi

Thiruxhzngoyu

Svlzm

**Trim:**

SQL> select trim(' Immanuvel R ') as trimmed from dual;

TRIMMED

-----------

Immanuvel R

**RPAD:**

SQL> select rpad(name,6) as RPADDED from details;

RPADDE

------

Immanu

Jothi

Kavya

Abishe

Karthi

**Upper:**

SQL> select upper(name) as caps from details;

CAPS

----------

IMMANUVEL

JOTHI SREE

KAVYA

ABISHEK

KARTHI

**Lower:**

SQL> select lower(name) as lows from details;

LOWS

----------

immanuvel

jothi sree

kavya

abishek

karthi

**Like:**

SQL> select \* from details where name like 'I%';

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

1 Immanuvel Rajappan Krishnagiri

SQL> select \* from details where city like 'Che%';

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

3 Kavya Senthil Chennimalai

SQL> select \* from details where city like 'G\_\_i';

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

2 Jothi Sree Ramasamy Gobi

SQL> select \* from details where city like '%a%';

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

1 Immanuvel Rajappan Krishnagiri

3 Kavya Senthil Chennimalai

5 Karthi Ranga Salem

SQL> select \* from details where name like 'K%i';

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

5 Karthi Ranga Salem

SQL> select \* from details where name like 'A\_%';

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

4 Abishek Aarumugam Thiruchengodu

SQL> select \* from details where name like '%e\_';

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

1 Immanuvel Rajappan Krishnagiri

2 Jothi Sree Ramasamy Gobi

4 Abishek Aarumugam Thiruchengodu

SQL> select \* from details where father\_name like 'Senthil';

ID NAME FATHER\_NAM CITY

---------- ---------- ---------- ---------------

3 Kavya Senthil Chennimalai

SQL> SELECT NAME ||' specialized in '|| work AS specilization FROM workers;

SPECILIZATION

------------------------------------

Immanuvel specialized in Full stack

Jothi specialized in Analytics

Karthi specialized in Architech

Abishek specialized in DBA

SQL> select \* from emp;

ID NAME EMAIL

---------- ---------- --------------------

1 Jothi jothi@gmail.com

2 Immanuvel immanuvel@gmail.com

3 Karthi [karthi18@gmail.com](mailto:karthi18@gmail.com)

**Domain:**

SQL> SELECT NAME, SUBSTR(EMAIL, INSTR(EMAIL, '@') + 1) AS email\_domain FROM emp;

NAME EMAIL\_DOMAIN

---------- --------------------

Jothi gmail.com

Immanuvel gmail.com

Karthi gmail.com

**Format:**

SQL> SELECT name,email FROM emp WHERE REGEXP\_LIKE(EMAIL, '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');

NAME EMAIL

---------- --------------------

Jothi jothi@gmail.com

Immanuvel immanuvel@gmail.com

Karthi [karthi18@gmail.com](mailto:karthi18@gmail.com)

**Change domain:**

SQL> SELECT REPLACE(EMAIL, SUBSTR(EMAIL, INSTR(EMAIL, '@')), '@kongu.edu') AS new\_email FROM emp;

NEW\_EMAIL

--------------------------------------------------------------------------------

jothi@kongu.edu

immanuvel@kongu.edu

karthi18@kongu.edu

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT**

Thus sql string operations are performed by using string functions

|  |  |
| --- | --- |
| **EX NO : 07**  **25 -04 -24** | **Set Operations & Aggregate Functions** |

**AIM:**

To perform aggregation on the relational database and group the records based on the conditions.

**AGGREGATE FUNCTION**

* An aggregate function in SQL performs a calculation on multiple values and returns a single value.
* SQL provides many aggregate functions that include avg, count, sum, min, max, etc.
* An aggregate function ignores NULL values when it performs the calculation, except for the count function.

**Table creation and viewing:**

SQL> select \* from workers;

NAME WORK SALARY

---------- ---------- ----------

Immanuvel Full stack 10000

Jothi Analytics 8000

Karthi Architech 9000

Abishek DBA 5000

SQL> select \* from parttime;

NAME WORK SALARY

---------- ---------- ----------

Immanuvel UI design 1000

Karthi Painting 800

Kavya Python 500

Kaviya Flute 1000

**Set operations:**

SET operators are special type of operators which are used to combine the result of two queries.

**Union:**

SQL> select name from workers union select name from parttime;

NAME

----------

Abishek

Immanuvel

Jothi

Karthi

Kaviya

Kavya

6 rows selected.

**Union all:**

SQL> select name from workers union all select name from parttime;

NAME

----------

Immanuvel

Jothi

Karthi

Abishek

Immanuvel

Karthi

Kavya

Kaviya

8 rows selected.

**Intersect:**

SQL> select name from workers intersect select name from parttime;

NAME

----------

Immanuvel

Karthi

**Minus:**

SQL> select name from workers minus select name from parttime;

NAME

----------

Abishek

Jothi

**Aggregate functions**

SQL> select avg(salary) from workers;

AVG(SALARY)

-----------

8000

SQL> select max(salary) from workers;

MAX(SALARY)

-----------

10000

SQL> select min(salary) from workers;

MIN(SALARY)

-----------

5000

SQL> select sum(salary) from workers;

SUM(SALARY)

-----------

32000

SQL> select count(salary) from workers;

COUNT(SALARY)

-------------

4

**Group by**

SQL> select work,avg(salary) from workers group by work;

WORK AVG(SALARY)

---------- -----------

Full stack 8500

Analytics 6000

DBA 5500

Architech 9000

SQL> select work,max(salary) from workers group by work;

WORK MAX(SALARY)

---------- -----------

Full stack 10000

Analytics 8000

DBA 6000

Architech 9000

SQL> select work,min(salary) from workers group by work;

WORK MIN(SALARY)

---------- -----------

Full stack 7000

Analytics 4000

DBA 5000

Architech 9000

**Having clause:**

SQL> select work,avg(salary) from workers group by work having avg(salary)>1000;

WORK AVG(SALARY)

---------- -----------

Full stack 8500

Analytics 6000

DBA 5500

Architech 9000

SQL> select distinct work from workers;

WORK

----------

Full stack

Analytics

DBA

Architech

**Order by:**

SQL> select name,salary from workers order by salary;

NAME SALARY

---------- ----------

Kowshika 4000

Abishek 5000

Kavya 6000

Lalit 7000

Jothi 8000

Karthi 9000

Immanuvel 10000

7 rows selected.

SQL> select name,salary from workers order by salary asc;

NAME SALARY

---------- ----------

Kowshika 4000

Abishek 5000

Kavya 6000

Lalit 7000

Jothi 8000

Karthi 9000

Immanuvel 10000

7 rows selected.

SQL> select name,salary from workers order by salary desc;

NAME SALARY

---------- ----------

Immanuvel 10000

Karthi 9000

Jothi 8000

Lalit 7000

Kavya 6000

Abishek 5000

Kowshika 4000

7 rows selected.

SQL> select name,salary from workers order by name asc , salary desc;

NAME SALARY

---------- ----------

Abishek 5000

Immanuvel 10000

Jothi 8000

Karthi 9000

SQL> select work,avg(salary) from workers group by work order by avg(salary) desc;

WORK AVG(SALARY)

---------- -----------

Architech 9000

Full stack 8500

Analytics 6000

DBA 5500

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT**

Hence different aggregate functions have been successfully executed.

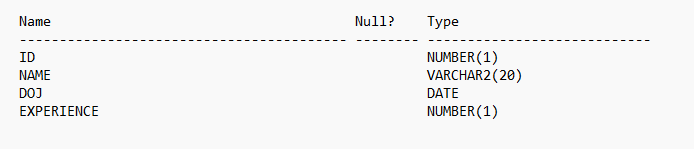
|  |  |
| --- | --- |
| **EX NO : 08**  **03 -05 -24** | **Date And Time Functions** |

**Aim:**

To perform date and time functions in SQL.

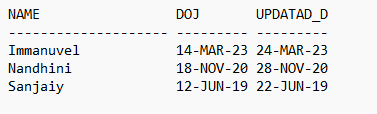
**STAFF TABLE:**

SQL> desc Staff;



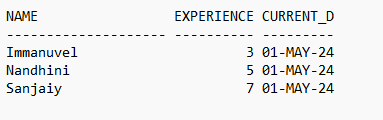
**1. ADD DATE:**

SQL> select Name,DOJ,DOJ+Interval '10'day as Updatad\_DOJ from Staff;



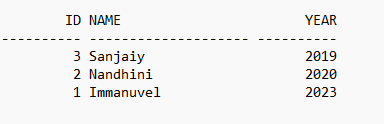
**2.CUR DATE:**

SQL> Select Name,Experience,Current\_Date AS Current\_Date from Staff;



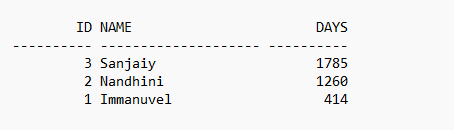
**3.EXTRACT:**

SQL> select ID,Name,extract(Year from DOJ) AS Year from Staff order by Year;



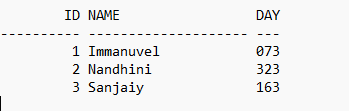
**4.CALCULATE INTERVAL BETWEEN TWO DATES:**

SQL> select ID,Name,trunc(sysdate)-trunc(DOJ) as Days from Staff order by days desc;



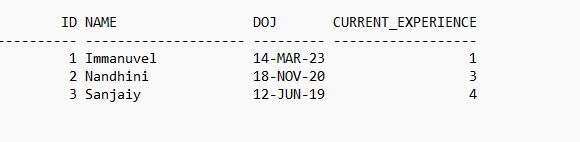
**5.DAY OF THE YEAR:**

SQL> select ID,Name,to\_char(DOJ,'DDD') as DAY from Staff;



**6.CALCULATE EXPERIENCE:**

SQL> select ID,Name,DOJ,trunc(Months\_between(sysdate,DOJ)/12) as Current\_Experience from Staff;



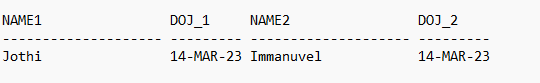
**7.STAFF WITH LONG TENURE:**

SQL> select ID,Name,DOJ from Staff where Doj =(select min(DOJ) from Staff);



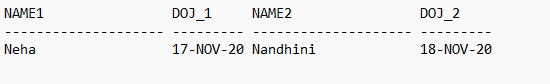
**8.STAFF JOINED ON SAME DAY:**

SQL> select s1.Name as name1,s1.DOJ AS DOJ\_1,s2.Name as name2,s2.DOJ AS DOJ\_2 from Staff s1 join Staff s2 on s1.DOJ = s2.DOJ and s1.ID>s2.ID;



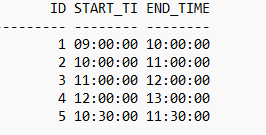
**9.HIRED ON ALTERNATE DAYS:**

SQL> select s1.Name as name1,s1.DOJ AS DOJ\_1,s2.Name as name2,s2.DOJ AS DOJ\_2 from Staff s1 join Staff s2 on ABS(s1.DOJ-s2.DOJ)=1 and s1.ID>s2.ID;



**10.ADDHOUR:**

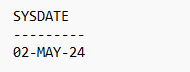
SQL> select id,To\_char(start\_time,'hh24:mi:ss') as start\_time,to\_char(start\_time+interval '1' hour,'hh24:mi:ss') as End\_time from time;



**USING DUAL:**

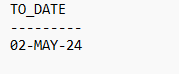
**1. CURRENT DATE AND TIME:**

sql> select sysdate from dual;



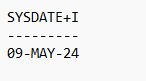
**2.SPECIFIC DATE:**

sql> select to\_date('2024-05-02', 'yyyy-mm-dd') from dual;



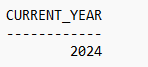
**3.DATE ARITHMETIC:**

sql> select sysdate + interval '7' day from dual;



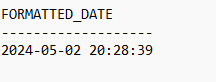
**4.EXTRACTING COMPONENTS FROM A DATE:**

sql> select extract(year from sysdate) as current\_year from dual;



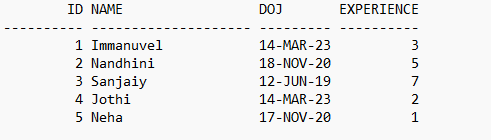
**5.FORMATTING DATES:**

sql> select to\_char(sysdate, 'yyyy-mm-dd hh24:mi:ss') as formatted\_date from dual;



**6.DATE COMPARISON:**

sql> select \* from staff where doj < sysdate;



**7.TO CHAR:**

SQL> select to\_char(sysdate,'DD/MM/YYYY') as NEW\_DATE from dual;

NEW\_DATE

----------

03/05/2024

**8.LAST DAY:**

SQL> select last\_day(sysdate) as LAST\_DATE from dual;

LAST\_DATE

---------

31-MAY-24

**9.MONTHS BETWEEN:**

SQL> select round(months\_between(sysdate,date '2004-07-12')) MONTH\_DIFFERENCE from dual;

MONTH\_DIFFERENCE

----------------

238

**10.** **ADD\_MONTHS:**

SQL> select add\_months(sysdate,7) NEWDATE from dual;

NEWDATE

---------

03-DEC-24

**11.** **FROM\_TZ:**

SQL> select from\_tz(timestamp '2020-05-01 19:35:10','-07:00') NEWVALUE from dual;

NEWVALUE

---------------------------------------------------------------------------

01-MAY-20 07.35.10.000000000 PM -07:00

**12.** **NEW\_TIME:**

SQL> select new\_time(sysdate,'pst','ast') TIME\_IN\_AST from dual;

TIME\_IN\_A

---------

03-MAY-24

**13.** **SESSIONTIMEZONE:**

SQL> select sessiontimezone from dual;

SESSIONTIMEZONE

---------------------------------------------------------------------------

+05:30

**14.** **SYSTIMESTAMP:**

SYSTIMESTAMP

---------------------------------------------------------------------------

03-MAY-24 03.03.18.886000 PM +05:30

**15.** **TRUNC:**

SQL> select trunc(sysdate,'MM') MONTH from dual;

MONTH

---------

01-MAY-24

**16.** **TZ\_OFFSET:**

SQL> select TZ\_OFFSET('Indian/Christmas') as OFFSET from dual;

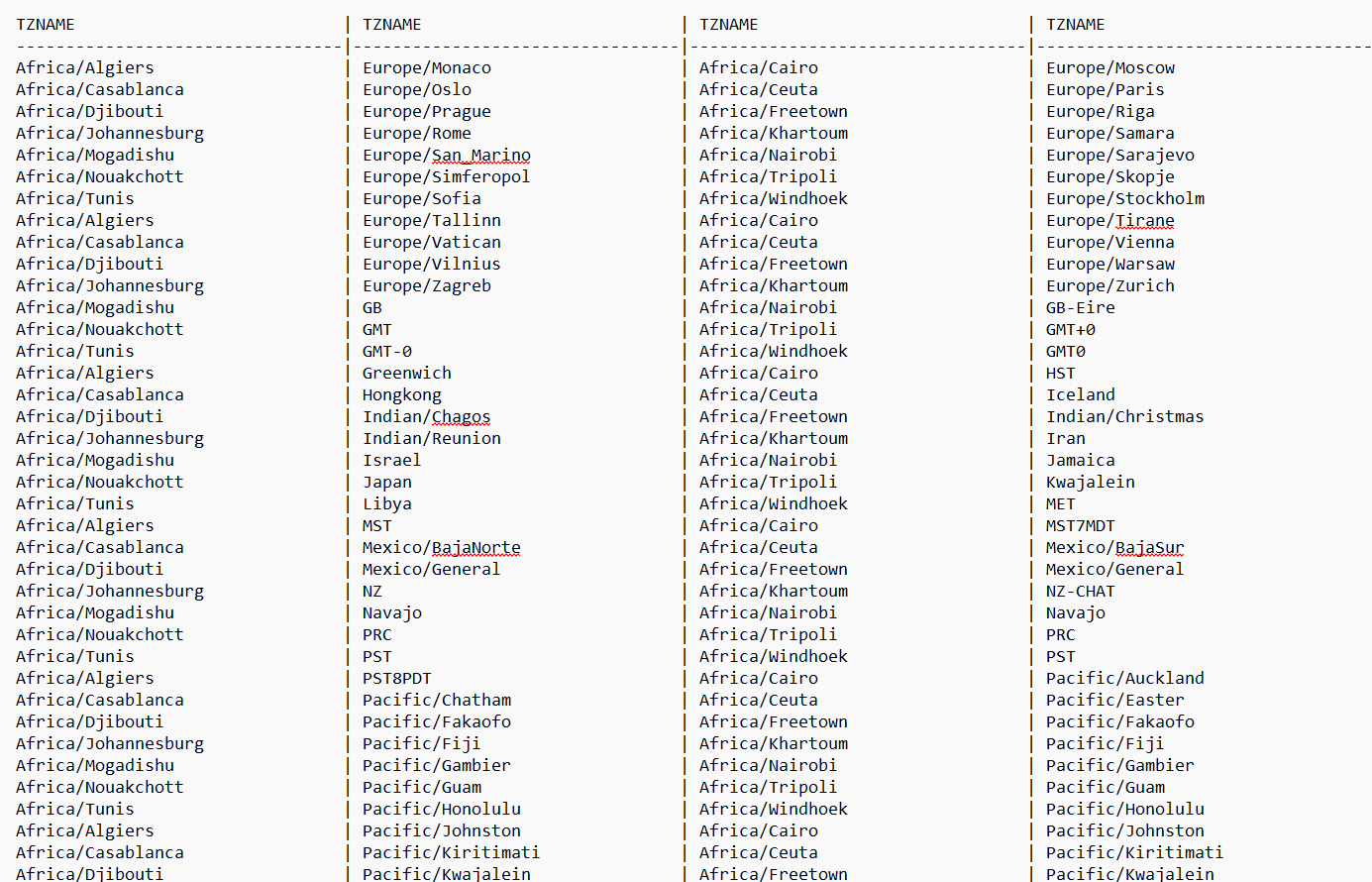
OFFSET

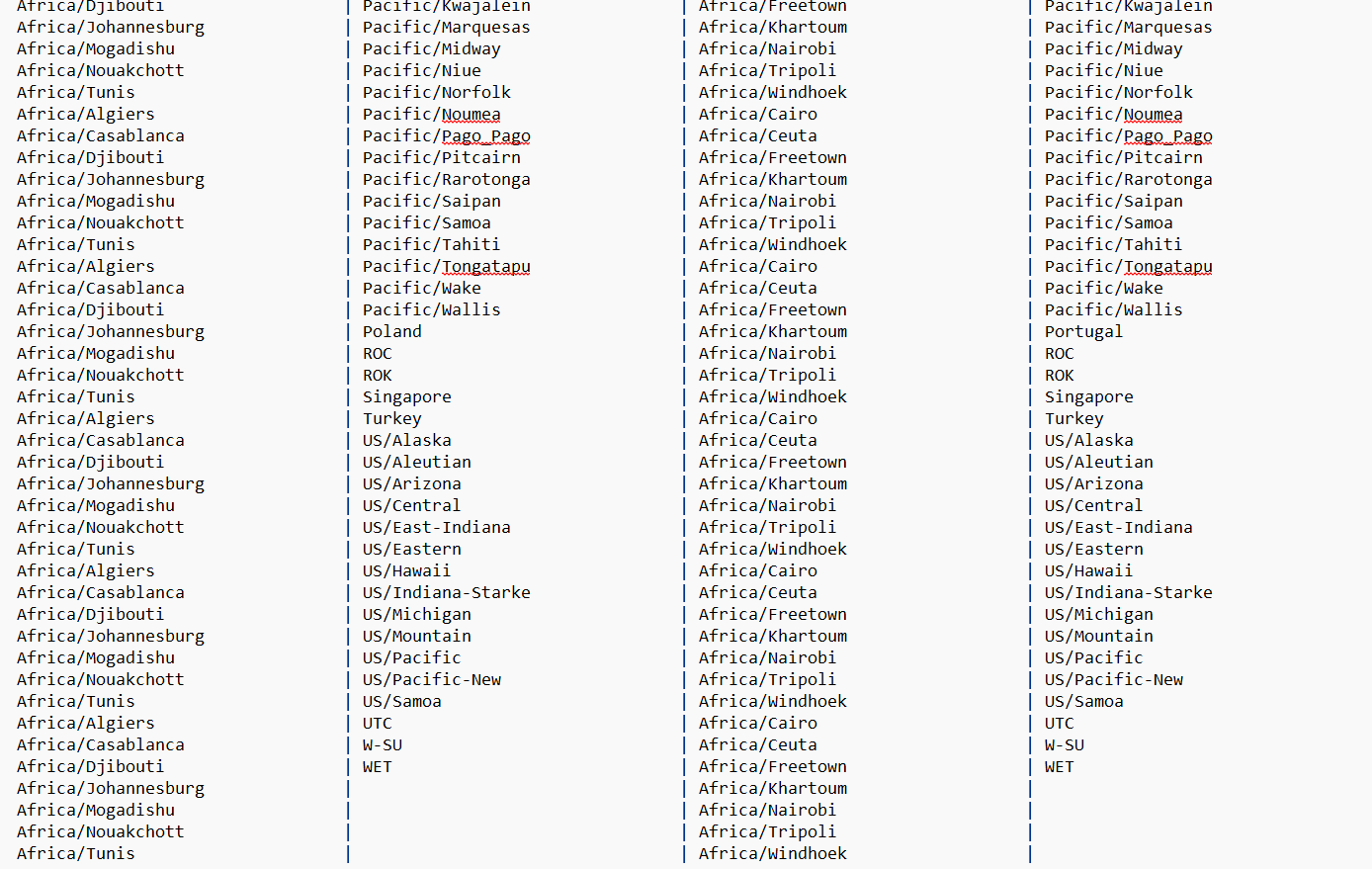
-------

+07:00

**17.** **To retrieve list of distinct time Zone:**

select distinct tzname from v$timezone\_names order by tzname;





|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT**

Thus, SQL date and time functions has been executed successfully.

|  |  |
| --- | --- |
| **EX NO : 9**  **08-05-24** | **PL/SQL Programs** |

**Aim:**

To execute PL/SQL statements in various programs.

**PL/SQL:** PL/SQL is a combination of SQL along with the procedural features of programming

languages. PL/SQL is one of the key programming languages embedded in the Oracle Database

along with SQL.

**Features of PL/SQL:**

* PL/SQL is tightly integrated with SQL.
* It offers extensive error checking.
* It offers numerous data types.
* It offers a variety of programming structures.
* It supports structured programming through functions and procedures.

**Program:**

set serveroutput on

1. **Program to print Hello world**

SQL> declare

msg varchar(30):='Kongu Engineering College!';

begin

dbms\_output.put\_line(msg);

end;

/

Kongu Engineering College!

PL/SQL procedure successfully completed.

**2. Program to print sum of two numbers**

SQL> declare

a integer:=10;

b integer:=20;

c integer;

d real;

begin

c:=a+b;

dbms\_output.put\_line('Value of c is: '||c);

d:=20.0/10.0;

dbms\_output.put\_line('Value of d is: '||d);

end;

/

Value of c is: 30

Value of d is: 2

PL/SQL procedure successfully completed.

**3. Simple IF-THEN Statement**

SQL> declare

n number;

begin

n:=&n;

if n>18 then

dbms\_output.put\_line('Eligible to Vote');

end if;

end;

/

Enter value for n: 19

old 4: n:=&n;

new 4: n:=19;

Eligible to Vote

PL/SQL procedure successfully completed.

**4. Simple IF-THEN-ELSE Statement**

SQL> declare

n number;

begin

n:=&n;

if n>18 then

dbms\_output.put\_line('Eligible to vote');

else

dbms\_output.put\_line('Not eligible to vote');

end if;

end;

/

Enter value for n: 17

old 4: n:=&n;

new 4: n:=17;

Not eligible to vote

PL/SQL procedure successfully completed.

**5. Nested IF-THEN-ELSE Statements**

SQL> declare

n number;

begin

n:=&n;

if n>18 then

dbms\_output.put\_line('Eligible to vote');

else

if n=18 then

dbms\_output.put\_line('Eligible to vote');

else

dbms\_output.put\_line('Not eligible to vote');

end if;

end if;

end;

/

Enter value for n: 18

old 4: n:=&n;

new 4: n:=18;

Eligible to vote

PL/SQL procedure successfully completed.

**6. IF-THEN-ELSIF Statement**

SQL> declare

n number;

begin

n:=&n;

if n>18 then

dbms\_output.put\_line('Eligible to vote');

elsif n=18 then

dbms\_output.put\_line('Eligible to vote');

else

dbms\_output.put\_line('Not eligible to vote');

end if;

end;

/

Enter value for n: 19

old 4: n:=&n;

new 4: n:=19;

Eligible to vote

PL/SQL procedure successfully completed.

**7. Extended IF-THEN Statement**

SQL> declare

cgpa number;

begin

cgpa:=10;

if cgpa=10 then

dbms\_output.put\_line('O grade');

elsif cgpa=9 then

dbms\_output.put\_line('A+ grade');

elsif cgpa=8 then

dbms\_output.put\_line('A grade');

elsif cgpa=7 then

dbms\_output.put\_line('B+ grade');

elsif cgpa=6 then

dbms\_output.put\_line('B grade');

else

dbms\_output.put\_line('No such grade');

end if;

end;

/

O grade

PL/SQL procedure successfully completed.

**8. Simple CASE Statement**

SQL> declare

cgpa number;

begin

cgpa:=9;

case cgpa

when 10 then dbms\_output.put\_line('O grade');

when 9 then dbms\_output.put\_line('A+ grade');

when 8 then dbms\_output.put\_line('A grade');

when 7 then dbms\_output.put\_line('B+ grade');

when 6 then dbms\_output.put\_line('B grade');

else dbms\_output.put\_line('No such grade');

end case;

end;

/

A+ grade

PL/SQL procedure successfully completed.

**9. Searched CASE Statement**

SQL> declare

cgpa number;

begin

cgpa:=7;

case

when cgpa=10 then dbms\_output.put\_line('O grade');

when cgpa=9 then dbms\_output.put\_line('A+ grade');

when cgpa=8 then dbms\_output.put\_line('A grade');

when cgpa=7 then dbms\_output.put\_line('B+ grade');

when cgpa=6 then dbms\_output.put\_line('B grade');

else dbms\_output.put\_line('No such grade');

end case;

end;

/

B+ grade

PL/SQL procedure successfully completed.

**10. EXCEPTION Instead of ELSE Clause in CASE Statement**

SQL> declare

cgpa number;

begin

cgpa:=5;

case

when cgpa=10 then dbms\_output.put\_line('O grade');

when cgpa=9 then dbms\_output.put\_line('A+ grade');

when cgpa=8 then dbms\_output.put\_line('A grade');

when cgpa=7 then dbms\_output.put\_line('B+ grade');

when cgpa=6 then dbms\_output.put\_line('B grade');

else dbms\_output.put\_line('No such grade');

end case;

exception

when case\_not\_found then

dbms\_output.put\_line('No such Grade');

end;

/

No such grade

PL/SQL procedure successfully completed.

**11. Program for simple LOOP**

SQL> declare

i number:=0;

begin

loop

i:=i+1;

exit when i>=10;

end loop;

dbms\_output.put\_line('value of i:'||i);

end;

/

value of i:10

PL/SQL procedure successfully completed.

**12. Program using WHILE LOOP:**

SQL> declare

i number:=0;

j number:=0;

begin

while i<=20

loop

j:=j+i;

i:=i+2;

dbms\_output.put\_line('The Value of j is '||j);

end loop;

end;

/

The Value of j is 0

The Value of j is 2

The Value of j is 6

The Value of j is 12

The Value of j is 20

The Value of j is 30

The Value of j is 42

The Value of j is 56

The Value of j is 72

The Value of j is 90

The Value of j is 110

PL/SQL procedure successfully completed.

**13. WHILE-LOOP Statement**

SQL> declare

n1 number:=10;

n2 number:=1;

begin

while n2<n1 loop

dbms\_output.put\_line('Value of n2:'||n2);

n2:=n2+1;

end loop;

end;

/

Value of n2:1

Value of n2:2

Value of n2:3

Value of n2:4

Value of n2:5

Value of n2:6

Value of n2:7

Value of n2:8

Value of n2:9

PL/SQL procedure successfully completed.

**14. FOR-LOOP Statement**

SQL> begin

for i in 1..20

loop

dbms\_output.put\_line(to\_char(i));

end loop;

end;

/

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

PL/SQL procedure successfully completed.

**15. Reverse FOR-LOOP Statement**

SQL> begin

for i in reverse 1..10

loop

dbms\_output.put\_line(to\_char(i));

end loop;

end;

/

10

9

8

7

6

5

4

3

2

1

PL/SQL procedure successfully completed.

16. Simple GOTO Statement

SQL> declare

prime varchar2(30);

num pls\_integer:=7;

begin

for j in 2..round(sqrt(num))

loop

if num mod j=0 then

prime:='is not a prime number';

goto display;

end if;

end loop;

prime:='is a prime number';

<<display>>

dbms\_output.put\_line(to\_char(num) || prime);

end;

/

7is not a prime number

PL/SQL procedure successfully completed.

**17. GOTO Statement to Branch to an Enclosing Block**

SQL> declare

v\_name varchar(15);

emp\_id number(6) := 1;

begin

<<get\_name>>

begin

select name into v\_name from instructor where id = emp\_id;

dbms\_output.put\_line(v\_name);

emp\_id := emp\_id + 2;

if emp\_id <= 15 then

goto get\_name;

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('No instructor found in ID ' || emp\_id);

end;

end;

/

Vijay

Surya

Sanjay

Saran

Karthi

Sathis

No instructor found in ID 13

PL/SQL procedure successfully completed.

**18. Do...While Statement:**

SQL> declare

num number:=0;

begin

loop

dbms\_output.put(num||',');

num:=num+2;

exit when num>=10;

end loop;

dbms\_output.put\_line('Even Numbers: '||num);

end;

/

0,2,4,6,8,Even Numbers: 10

PL/SQL procedure successfully completed.

**19. Factorial value**

SQL> declare

n number:=10;

fact number:=1;

begin

for i in 1..n loop

fact:=fact\*i;

end loop;

dbms\_output.put\_line('Factorial numbers are: '|| n ||' is '||fact);

end;

/

Factorial numbers are: 10 is 3628800

PL/SQL procedure successfully completed.

**20. Prime Number Generation**

SQL> declare

v\_limit number := 20;

isprime boolean;

begin

for i in 2..v\_limit loop

isprime := true;

for j in 2..sqrt(i) loop

if mod(i, j) = 0 then

isprime := false;

exit;

end if;

end loop;

if isprime then

dbms\_output.put\_line(i || ' is a prime number');

end if;

end loop;

end;

/

2 is a prime number

3 is a prime number

5 is a prime number

7 is a prime number

11 is a prime number

13 is a prime number

17 is a prime number

19 is a prime numbe

PL/SQL procedure successfully completed.

**21. Fibonacci Series**

SQL> declare

v\_start number:=5;

a number:=0;

b number:=1;

c number;

begin

dbms\_output.put\_line('Fibbonacci series: ');

dbms\_output.put\_line(a);

dbms\_output.put\_line(b);

for i in 3..v\_start loop

c:=a+b;

dbms\_output.put\_line(c);

a:=b;

b:=c;

end loop;

end;

/

Fibbonacci series:

0

1

1

2

3

PL/SQL procedure successfully completed.

**22. Checking Palindrome**

SQL> declare

original varchar(20) := 'appa';

result varchar(100);

begin

for i in reverse 1..length(original) loop

result := result || substr(original, i, 1);

end loop;

if original = result then

dbms\_output.put\_line(original || ' is palindrome');

else

dbms\_output.put\_line(original || ' is not a palindrome');

end if;

end;

/

appa is palindrome

PL/SQL procedure successfully completed.

**23. Swap two numbers**

SQL> declare

n1 number := 45;

n2 number := 03;

temp number;

begin

dbms\_output.put\_line('Before Swapping:');

dbms\_output.put\_line('n1: ' || n1);

dbms\_output.put\_line('n2: ' || n2);

temp := n1;

n1 := n2;

n2 := temp;

dbms\_output.put\_line('After Swapping:');

dbms\_output.put\_line('n1: ' || n1);

dbms\_output.put\_line('n2: ' || n2);

end;

/

Before Swapping:

n1: 45

n2: 3

After Swapping:

n1: 3

n2: 45

PL/SQL procedure successfully completed.

**24. PL/SQL block for updating single row into a table.**

SQL> CREATE TABLE empdet (

eno NUMBER,

ename VARCHAR2(100),

deptno NUMBER,

basic NUMBER,

HRA NUMBER,

DA NUMBER,

PF NUMBER,

netpay NUMBER

);

Table created.

SQL> declare

v\_eno empdet.eno%type;

v\_ename empdet.ename%type;

v\_deptno empdet.deptno%type;

v\_basic empdet.basic%type;

v\_hra empdet.HRA%type;

v\_da empdet.DA%type;

v\_pf empdet.PF%type;

v\_netpay empdet.netpay%type;

begin

v\_eno:=&v\_eno;

v\_ename:='&v\_ename';

v\_deptno:=&v\_deptno;101

v\_basic:=&v\_basic;

v\_hra:=(v\_basic\*50)/100;

v\_da:=(v\_basic\*20)/100;

v\_pf:=(v\_basic\*7)/100;

v\_netpay:=v\_basic+v\_hra+v\_da-v\_pf;

insert into empdet(eno,ename,deptno,basic,HRA,DA,PF,netpay) values(v\_eno,v\_ename,v\_deptno,v\_basic,v\_hra,v\_da,v\_pf,v\_netpay);

dbms\_output.put\_line('Row inserted successfully.');

exception

when others then

dbms\_output.put\_line('Error:'|| SQLERRM);

end;

/

Enter value for v\_eno: 101

old 11: v\_eno:=&v\_eno;

new 11: v\_eno:=101;

Enter value for v\_ename: John

old 12: v\_ename:='&v\_ename';

new 12: v\_ename:='John';

Enter value for v\_deptno: 10

old 13: v\_deptno:=&v\_deptno;

new 13: v\_deptno:=10;

Enter value for v\_basic: 50000

old 14: v\_basic:=&v\_basic;

new 14: v\_basic:=50000;

Row inserted successfully.

PL/SQL procedure successfully completed.

SQL> /

Enter value for v\_eno: 102

old 11: v\_eno:=&v\_eno;

new 11: v\_eno:=102;

Enter value for v\_ename: Pranav

old 12: v\_ename:='&v\_ename';

new 12: v\_ename:='Pranav';

Enter value for v\_deptno: 11

old 13: v\_deptno:=&v\_deptno;

new 13: v\_deptno:=11;

Enter value for v\_basic: 60000

old 14: v\_basic:=&v\_basic;

new 14: v\_basic:=60000;

Row inserted successfully.

PL/SQL procedure successfully completed.

SQL> /

Enter value for v\_eno: 103

old 11: v\_eno:=&v\_eno;

new 11: v\_eno:=103;

Enter value for v\_ename: Ram

old 12: v\_ename:='&v\_ename';

new 12: v\_ename:='Ram';

Enter value for v\_deptno: 12

old 13: v\_deptno:=&v\_deptno;

new 13: v\_deptno:=12;

Enter value for v\_basic: 40000

old 14: v\_basic:=&v\_basic;

new 14: v\_basic:=40000;

Row inserted successfully.

PL/SQL procedure successfully completed.

**25. PL/SQL block for updating multiple rows into a table.**

SQL> declare

new\_name varchar(15):='Sam';

begin

update empdet set ename=new\_name where eno=101;

commit;

end;

/

PL/SQL procedure successfully completed.

SQL> declare

new\_salary number:=80000;

begin

update empdet set basic=new\_salary where basic<=50000;

commit;

end;

/

PL/SQL procedure successfully completed.

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**Result:**

To PL/SQL statements in various programs are executed successfully.

|  |  |
| --- | --- |
| **EX NO:10**  **05-05-24** | **CURSORS** |

**Aim:**

To study and write the PL/SQL programs using cursors.

**Table creation:**

SQL> create table employee(ename varchar(10),eid number(3),dob date,basic number(8,2),da number (8,2),total number(5));

Table created.

SQL> select \* from employee;

ENAME EID DOB BASIC DA TOTAL

---------- ---------- --------- ---------- ---------- ----------

Kavin 1 06-OCT-90 25000

Arun 2 08-SEP-90 17000

Sankar 3 01-JAN-89 12000

Radha 4 10-APR-82 35000

Kevin 5 02-FEB-89 30000

**1. Program Using Implicit Cursor to Increase basic Of Each Customer By 500**

DECLARE

total\_rows number(2);

BEGIN

UPDATE employee SET basic= basic + 1000;

IF sql%notfound THEN

dbms\_output.put\_line('no customers selected');

ELSIF sql%found THEN

total\_rows := sql%rowcount;

dbms\_output.put\_line( total\_rows || ' Employees selected ');

END IF;

END;

/

5 Employees selected

PL/SQL procedure successfully completed.

ENAME EID DOB BASIC DA TOTAL

---------- ---------- --------- ---------- ---------- ----------

Kavin 1 06-OCT-90 26000

Arun 2 08-SEP-90 18000

Sankar 3 01-JAN-89 13000

Radha 4 10-APR-82 36000

Kevin 5 02-FEB-89 31000

**2. Program Using Explicit Cursor to Fetch the Employee Details from the Table**

DECLARE

e\_id employee.eid%type;

e\_name employee.ename%type;

e\_dob employee.dob%type;

CURSOR e\_employee is

SELECT eid, ename, dob FROM employee;

BEGIN

OPEN e\_employee;

LOOP

FETCH e\_employee into e\_id, e\_name, e\_dob;

EXIT WHEN e\_employee %notfound;

dbms\_output.put\_line(e\_id || ' ' || e\_name || ' ' || e\_dob);

END LOOP;

CLOSE e\_employee;

END;

/

1 Kavin 06-OCT-90

2 Arun 08-SEP-90

3 Sankar 01-JAN-89

4 Radha 10-APR-82

5 Kevin 02-FEB-89

PL/SQL procedure successfully completed.

**3. Program using explicit cursor to calculate da & total.**

DECLARE

v\_salary employee.basic%TYPE;

v\_da NUMBER;

v\_total\_salary NUMBER;

CURSOR c\_employee IS

SELECT basic, total

FROM employee;

BEGIN

FOR emp\_rec IN c\_employee LOOP

v\_salary := emp\_rec.basic;

IF v\_salary < 15000 THEN

v\_da := 0.25 \* v\_salary;

ELSIF v\_salary <= 30000 THEN

v\_da := 0.35 \* v\_salary;

ELSE

v\_da := 0.45 \* v\_salary;

END IF;

v\_total\_salary := v\_salary + v\_da;

UPDATE employee

SET da = v\_da,

total = v\_total\_salary

WHERE basic = v\_salary;

END LOOP;

COMMIT;

dbms\_output.put\_line('DA and Total Salary updated successfully.');

END;

/

DA and Total Salary updated successfully.

PL/SQL procedure successfully completed.

SQL> select \* from employee;

ENAME EID DOB BASIC DA TOTAL

---------- ---------- --------- ---------- ---------- -----------------

Kavin 1 06-OCT-90 26000 9100 35100

Arun 2 08-SEP-90 18000 6300 24300

Sankar 3 01-JAN-89 13000 3250 16250

Radha 4 10-APR-82 36000 16200 52200

Kevin 5 02-FEB-89 31000 13950 44950

**4. Create an explicit cursor to calculate results of students**

stud(name char(10),rno number(4),m1 number(10), m2 number(10),m3 umber(10),tot

number(16),avg number(5,2), result char(4));

CREATE TABLE students (

name VARCHAR2(5),

rno NUMBER(4),

m1 NUMBER(3),

m2 NUMBER(3),

m3 NUMBER(3),

tot NUMBER(16),

avg NUMBER(5,2),

result VARCHAR2(4)

);

INSERT INTO students (name, rno, m1, m2, m3) VALUES ('John', 101, 80, 75, 85);

INSERT INTO students (name, rno, m1, m2, m3) VALUES ('Alice', 102, 90, 85, 95);

INSERT INTO students (name, rno, m1, m2, m3) VALUES ('Bob', 103, 70, 65, 75);

SQL> select \* from student;

NAME RNO M1 M2 M3 TOT AVG RESU

----- ---------- ---------- ---------- ---------- ---------- ---------- ----

John 101 80 75 85

Alice 102 90 85 95

Bob 103 70 65 75

DECLARE

CURSOR c\_students IS

SELECT name, rno, m1, m2, m3

FROM student;

v\_total\_marks NUMBER(16);

v\_avg\_marks NUMBER(5,2);

v\_result CHAR(4);

BEGIN

FOR student\_rec IN c\_students LOOP

v\_total\_marks := student\_rec.m1 + student\_rec.m2 + student\_rec.m3;

v\_avg\_marks := v\_total\_marks / 3;

IF v\_avg\_marks >= 60 THEN

v\_result := 'Pass';

ELSE

v\_result := 'Fail';

END IF;

-- Update student's total, average, and result

UPDATE student

SET tot = v\_total\_marks,

avg = v\_avg\_marks,

result = v\_result

WHERE rno = student\_rec.rno;

END LOOP;

COMMIT;

dbms\_output.put\_line('Results calculated and updated successfully.');

END;

/

Results calculated and updated successfully.

PL/SQL procedure successfully completed.

NAME RNO M1 M2 M3 TOT AVG RESU

----- ---------- ---------- ---------- ---------- ---------- ---------- ----

John 101 80 75 85 240 80 Pass

Alice 102 90 85 95 270 90 Pass

Bob 103 70 65 75 210 70 Pass

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**Result:**

Thus, PL/SQL programs using cursor was implemented successfully.

|  |  |
| --- | --- |
| **EX NO : 11**  **31-05 -24** | **Trigger** |

**AIM:** To create and access the database using triggers.

**TRIGGER:**

A database trigger is a stored procedure that is fired when an insert, update or delete statement is issued against the associated table. Database triggers can be used for the following purposes.

* To generate data automatically.
* To enforce integrity constraints.
* To customize security authorizations.
* To maintain replicate tables.

**TO CREATE A TABLE REVISED**

SQL> create table revised(empid number(10),name varchar(10),salary number(10));

Table created.

**TO DISPLAY THE CONTENTS OF THE TABLE REVISED**

SQL> SQL> select \* from revised;

EMPID NAME SALARY

---------- ---------- ----------

1 Immanuvel 90000

2 Jothi 70000

3 Lalit 35000

4 Abishek 50000

5 Karthi 55000

6 Lipiga 60000

6 rows selected.

**TO CREATE TRIGGER AND UPDATE THE SALARY VALUE** SQL> create or replace trigger t1

2 after insert on revised

3 begin

4 update revised set salary=75000 where salary>50000; 5end;

6 /

Trigger created.

**TO INSERT VALUES INTO THE TABLE REVISED**

SQL> insert into revised values(&empid,'&name',&salary); Enter value for empid: 7

Enter value for name:Kavya

Enter value for salary: 70000

old 1: insert into revised values(&empid,'&name',&salary) new 1: insert into revised values(6,'Kavya',7000)

1 row created.

**TO DISPLAY THE TABLE REVISED AFTER UPDATING**

SQL> select \* from revised;

EMPID NAME SALARY

---------- ---------- ----------

1 Immanuvel 75000

2 Jothi 75000

3 Lalit 35000

4 Abishek 50000

5 Karthi 75000

6 Lipiga 75000

7 Kavya 75000

7 rows selected.

**TO DROP THE TRIGGER CREATED**

SQL> drop trigger t1;

Trigger dropped.

**TO CREATE A TABLE NEWBACKUP**

SQL> create table newbackup(empid number(10),name varchar(10),salary number(10)); Table created.

**TO CREATE A TABLE OLDBACKUP**

SQL> create table oldbackup(empid number(10),name varchar(10),salary number(10)); Table created.

**TO CREATE ANOTHER TRIGGER**

SQL> create or replace trigger t2

after insert or update on revised

for each row

begin if inserting

then insert into newbackup values(:new.empid,:new.name,:new.salary);

elsif updating then insert into oldbackup values(:old.empid,:old.name,:old.salary); end if;

end;

/Trigger created.

**TO INSERT VALUES INTO THE TABLE REVISED**

SQL> insert into revised values(&empid,'&name',&salary);

Enter value for empid: 8

Enter value for name: Giridharan

Enter value for salary: 85000

old 1: insert into revised values(&empid,'&name',&salary)

new 1: insert into revised values(8,'Giridharan',85000)

1 row created.

**TO DISPLAY THE TABLE REVISED AFTER UPDATING**

SQL> select \* from revised;

EMPID NAME SALARY

---------- ---------- ----------

1 Immanuvel 75000

2 Jothi 75000

3 Lalit 35000

4 Abishek 50000

5 Karthi 75000

6 Lipiga 75000

7 Kavya 75000

8 Giridharan 85000

8 rows selected.

**TO DISPLAY THE TABLE NEWBACKUP**

SQL> select \* from newbackup;

EMPID NAME SALARY

---------- ---------- ----------

8 Giridharan 85000

**TO UPDATE THE FIELD SALARY IN THE TABLE REVISED**

SQL> update revised set salary=100000 where salary>70000;

2 rows updated.

**TO DISPLAY THE TABLE REVISED**

SQL> select \* from revised;

EMPID NAME SALARY

---------- ---------- ----------

1 Immanuvel 100000

2 Jothi 100000

3 Lalit 35000

4 Abishek 50000

5 Karthi 100000

6 Lipiga 100000

7 Kavya 100000

8 Giridharan 100000

8 rows selected.

**TO DISPLAY THE TABLE OLDBACKUP**

SQL> select \* from oldbackup;

EMPID NAME SALARY

---------- ---------- ----------

1 Immanuvel 75000

2 Jothi 75000

5 Karthi 75000

6 Lipiga 75000

7 Kavya 75000

8 Giridharan 85000

6 rows selected.

**TO DISPLAY THE TABLE NEWBACKUP**

SQL> select \* from newbackup;

EMPID NAME SALARY

---------- ---------- ----------

1. Giridharan 85000

**PROGRAMS:**

**1 .CREATE STATEMENT LEVEL BEFORE INSERT TRIGGER :**

SQL> create or replace trigger t3

2 before insert on workers

3 begin

4 update workers set salary=9000 where salary<8000;

5 end;

6 /

Trigger created.

SQL> select \* from workers;

NAME WORK SALARY

---------- ---------- ----------

Immanuvel Full stack 10000

Jothi Analytics 8000

Karthi Architech 9000

Abishek DBA 7000

Lipiga Full stack 10000

Madanika DBA 7000

6 rows selected.

SQL> insert into workers values('Lalit','Software',4000);

1 row created.

SQL> select \* from workers;

NAME WORK SALARY

---------- ---------- ----------

Immanuvel Full stack 10000

Jothi Analytics 8000

Karthi Architech 9000

Abishek DBA 9000

Lipiga Full stack 10000

Madanika DBA 9000

Lalit Software 4000

**2.Create a Row level after update trigger to insert the new values into another table also**

SQL> select \* from workers;

NAME WORK SALARY

---------- ---------- ----------

Immanuvel Full stack 10000

Jothi Analytics 8000

Karthi Architech 9000

Abishek DBA 9000

Lipiga Full stack 10000

Madanika DBA 9000

Lalit Software 4000

7 rows selected.

SQL> select \* from updatingprogram;

no rows selected

SQL> create or replace trigger t5

2 after update on workers for each row

3 begin if updating

4 then insert into updatingprogram values(:old.name,:old.work,:old.salary);

5 end if;

6 end;

7 /

Trigger created.

SQL> select \* from updatingprogram;

no rows selected

SQL> update workers set work='DBA' where name='Karthi';

1 row updated.

SQL> select \* from workers;

NAME WORK SALARY

---------- ---------- ----------

Immanuvel Full stack 10000

Jothi Analytics 8000

Karthi DBA 9000

Abishek DBA 9000

Lipiga Full stack 10000

Madanika DBA 9000

Lalit Software 4000

7 rows selected.

SQL> select \* from updatingprogram;

NAME WORK SALARY

---------- --------------- ----------

Karthi Architech 9000

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT:**

Thus various types of trigger are created for the database.

|  |  |
| --- | --- |
| **EX NO : 12**  **17 -06 -24** | **Procedure and Function** |

**AIM:** To implement the procedure and functions using PL/SQL

**1. Create a function to find factorial of a number.**

create or replace function factorial(n number) return number is

fact number := 1;

begin

if n < 0 then

return null;

elsif n = 0 then

return 1;

else

for i in 1..n loop

fact := fact \* i;

end loop;

return fact;

end if;

end;

/

Function created.

SQL> select factorial(6) from dual;

FACTORIAL(6)

------------

720

**2. Create a procedure to calculate the area of the square**

create or replace function area\_of\_square(a number) return number is

area number:=0;

begin

area:=a\*a;

return area;

end;

/

Function created.

SQL> select area\_of\_square(10) from dual;

AREA\_OF\_SQUARE(10)

------------------

100

**3. Write a PL/SQL function called POW that takes two numbers as argument and return the**

**value of the first number raised to the power of the second.**

create or replace function pow(base number,exponent number) return number is

result number:=1;

begin

for i in 1..exponent loop

result:=result\*base;

end loop;

return result;

end;

/

Function created.

Function created.

SQL> select pow(3,4) from dual;

POW(3,4)

----------

81

**4. Write a PL/SQL function ODDEVEN to return value TRUE if the number passed to it is**

**EVEN else will return FALSE**

create or replace function oddeven(n number) return varchar is

begin

if mod(n,2)=0 then

return 'even';

else

return 'odd';

end if;

end;

/

SQL> select oddeven(18) from dual;

ODDEVEN(18)

------------------------

Even

SQL> select oddeven(7) from dual;

ODDEVEN(7)

------------------------

Odd

**5. Write a PL/SQL procedure called MULTI\_TABLE that takes two numbers as parameter and**

**displays the multiplication of the first parameter till the second parameter.**

SQL> create or replace procedure multi\_table(n in number, m in number) is

2 result number;

3 begin

4 for i in n..m loop

5 result := n \* i;

6 dbms\_output.put\_line(n || ' \* ' || i || ' = ' || result);

7 end loop;

8 end;

9 /

Procedure created.

SQL> begin

2 multi\_table(1,5);

3 end;

4 /

1 \* 1 = 1

1 \* 2 = 2

1 \* 3 = 3

1 \* 4 = 4

1 \* 5 = 5

PL/SQL procedure successfully completed.

**6. Create a function to find the maximum salary from the table customer.**

SQL> select \* from customer;

ID NAME ADDRESS SALARY AGE

---------- -------------------- -------------------- ---------- ----------

1 Immanuvel Vriddhachalam 70000 24

2 Jothi Sree panruti 65000 24

3 Karthikeyan vellore 58000 30

4 Kavya chennai 67000 29

5 Abishek salem 45000 27

6 vel cuddalore 40000 26

6 rows selected.

create or replace function maxsalary return number is

maximum\_salary number;

begin

select max(salary) into maximum\_salary from customers;

return maximum\_salary;

end;

/

SQL> create or replace function maxsalary return number is

2 maximum\_salary number;

3 begin

4 select max(salary) into maximum\_salary from customer;

5 return maximum\_salary;

6 end;

7 /

Function created.

SQL> select maxsalary() as maximum\_salary from dual;

MAXIMUM\_SALARY

--------------

70000

**7. Create a procedure to find the maximum and minimum salary from the table customer.**

SQL> create or replace procedure max\_and\_min\_salary is

2 max\_salary number;

3 min\_salary number;

4 begin

5 select max(salary), min(salary) into max\_salary, min\_salary from customer;

6 dbms\_output.put\_line('Maximum Salary: ' || max\_salary);

7 dbms\_output.put\_line('Minimum Salary: ' || min\_salary);

8 end;

9 /

Procedure created.

SQL> begin

2 max\_and\_min\_salary;

3 end;

4 /

Maximum Salary: 70000

Minimum Salary: 40000

PL/SQL procedure successfully completed.

**8. Program for procedure – selected record’s price is incremented by 500 , executing the**

**procedure created and displaying the updated table**

SQL> create or replace procedure increment\_price as

2 begin

3 update customer

4 set salary=salary+500

5 end;

6 /

Warning: Procedure created with compilation errors.

SQL> create or replace procedure increment\_price as

2 begin

3 update customer set salary=salary+500;

4 end;

5 /

Procedure created.

SQL> execute increment\_price;

PL/SQL procedure successfully completed.

SQL> select \* from customer;

ID NAME ADDRESS SALARY AGE

---------- -------------------- -------------------- ---------- ----------

1 Immanuvel Vriddhachalam 70500 24

2 Jothi Sree panruti 65500 24

3 Karthikeyan vellore 58500 30

4 Kavya chennai 67500 29

5 Abishek salem 45500 27

6 vel cuddalore 40500 26

6 rows selected.

**9. Create a function to return the sum of the salary in the customer table whose age is greater than 25.**

create or replace function sumofsalary return number is

total\_salary number:=0;

begin

select sum(salary) into total\_salary from customer

where age>25;

return total\_salary;

end;

/

Function created.

SQL> select sumofsalary as total\_salary from dual;

TOTAL\_SALARY

------------

212000

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT:**

Thus the procedure and functions are executed successfully.

|  |  |
| --- | --- |
| **EX NO : 13**  **14 -06- 24** | **Mini Project** |

**AIM:**

To develop a web application of Payment management system and to store the data in MYSQL.

**Coding:**

**Frontend:**

Index.html

<!DOCTYPE html>

<html>

<head>

<title>Diwali Payments</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

background-color: #f0f0f0;

}

.container {

max-width: 800px;

margin: 0 auto;

background-color: #fff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

}

h1, h2 {

color: #333;

}

form {

margin-bottom: 20px;

}

label {

display: inline-block;

width: 100px;

font-weight: bold;

}

input[type="text"], input[type="number"], input[type="submit"], input[type="radio"] {

padding: 8px;

margin-bottom: 10px;

border-radius: 4px;

border: 1px solid #ccc;

}

input[type="submit"] {

background-color: #4caf50;

color: white;

border: none;

cursor: pointer;

}

input[type="submit"]:hover {

background-color: #45a049;

}

input[type="radio"] {

margin-right: 10px;

}

table {

width: 100%;

border-collapse: collapse;

margin-bottom: 20px;

}

table, th, td {

border: 1px solid #ddd;

}

th, td {

padding: 12px;

text-align: left;

}

th {

background-color: #f2f2f2;

}

.hidden {

display: none;

}

.navigation-buttons {

margin-top: 20px;

}

.navigation-buttons button {

padding: 10px 20px;

margin-right: 10px;

border-radius: 4px;

border: none;

background-color: #007bff;

color: #fff;

cursor: pointer;

}

.navigation-buttons button:hover {

background-color: #0056b3;

}

.result-section {

margin-top: 20px;

}

.result-section h2 {

margin-bottom: 10px;

}

.result-table {

width: 100%;

border-collapse: collapse;

}

.result-table, .result-table th, .result-table td {

border: 1px solid #ddd;

}

.result-table th, .result-table td {

padding: 12px;

text-align: left;

}

.result-table th {

background-color: #f2f2f2;

}

</style>

</head>

<body>

<h1>Diwali Payments</h1>

<div>

<h2>Add User</h2>

<form id="addUserForm">

<label for="c\_name">Name:</label>

<input type="text" id="c\_name" name="c\_name" required>

<br>

<label for="c\_vill">Village:</label>

<input type="text" id="c\_vill" name="c\_vill" required>

<br>

<label for="c\_category">Category:</label>

<input type="text" id="c\_category" name="c\_category" required>

<br>

<label for="phone">Phone:</label>

<input type="text" id="phone" name="phone" required>

<br>

<input type="submit" value="Add User">

</form>

<div id="addUserResult" class="hidden"></div>

</div>

<div>

<h2>Add Payment</h2>

<form id="addPaymentForm">

<label for="c\_id">User ID:</label>

<input type="number" id="c\_id" name="c\_id" required>

<br>

<label for="p\_month">Payment Month:</label>

<input type="text" id="p\_month" name="p\_month" required>

<br>

<input type="submit" value="Add Payment">

</form>

<div id="addPaymentResult" class="hidden"></div>

</div>

<div>

<h2>Find User</h2>

<form id="findUserForm">

<label for="userId">Enter User ID:</label>

<input type="number" id="userId" name="userId" required>

<input type="submit" value="Find User">

</form>

<div id="userResult" class="hidden"></div>

</div>

<div>

<h2>Find Payments by User ID</h2>

<form id="findPaymentsForm">

<label for="userIdPayments">Enter User ID:</label>

<input type="number" id="userIdPayments" name="userIdPayments" required>

<input type="submit" value="Find Payments">

</form>

<div id="paymentsResult" class="hidden"></div>

</div>

<div>

<h2>View Payments by Month</h2>

<form id="viewPaymentsByMonthForm">

<label for="p\_month">Enter Month:</label>

<input type="text" id="p\_month" name="p\_month" required>

<input type="submit" value="View Payments">

</form>

<div id="paymentsByMonthResult" class="hidden"></div>

</div>

<div class="hidden">

<h2>Total Amount Paid</h2>

<form id="totalAmountPaidForm">

<label for="userIdTotal">Enter User ID:</label>

<input type="number" id="userIdTotal" name="userIdTotal" required>

<input type="submit" value="Get Total Amount Paid">

</form>

<div id="totalAmountResult" class="hidden"></div>

</div>

<div>

<h2>Delete User</h2>

<form id="deleteUserForm">

<label for="userIdDelete">Enter User ID to Delete:</label>

<input type="number" id="userIdDelete" name="userIdDelete" required>

<input type="submit" value="Delete User">

</form>

<div id="deleteResult" class="hidden">

<p id="deletedUserDetails"></p>

<p id="paymentCount"></p>

</div>

</div>

<div class="hidden">

<h2>All Users</h2>

<button id="loadUsers" type="button" style="background-color: #4caf50; color: white; border: none; cursor: pointer;">Load Users</button>

<div id="usersList" class="hidden"></div>

</div>

<div>

<h2>Update User</h2>

<form id="searchUserForm">

<label for="searchUserId">Enter User ID:</label>

<input type="number" id="searchUserId" name="searchUserId" required>

<input type="submit" value="Search User">

</form>

<div id="updateUserFormContainer" class="hidden">

<form id="updateUserForm">

<input type="hidden" id="updateUserId" name="c\_id">

<label for="update\_c\_name">Name:</label>

<input type="text" id="update\_c\_name" name="c\_name" required>

<br>

<label for="update\_c\_vill">Village:</label>

<input type="text" id="update\_c\_vill" name="c\_vill" required>

<br>

<label for="update\_c\_category">Category:</label>

<input type="text" id="update\_c\_category" name="c\_category" required>

<br>

<label for="update\_phone">Phone:</label>

<input type="text" id="update\_phone" name="phone" required>

<br>

<input type="submit" value="Update User">

</form>

</div>

</div>

<script>

document.addEventListener('DOMContentLoaded', function() {

// JavaScript code to handle form submissions

// Helper function to create a table

function createTable(data, columns) {

let table = '<table><thead><tr>';

columns.forEach(col => table += `<th>${col}</th>`);

table += '</tr></thead><tbody>';

data.forEach(row => {

table += '<tr>';

columns.forEach(col => table += `<td>${row[col]}</td>`);

table += '</tr>';

});

table += '</tbody></table>';

return table;

}

// Handle add user form submission

document.getElementById('addUserForm').addEventListener('submit', function (event) {

event.preventDefault();

const formData = new FormData(this);

const data = {};

formData.forEach((value, key) => {

data[key] = value;

});

fetch('/add\_user', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(data)

})

.then(response => response.json())

.then(result => {

const addUserResult = document.getElementById('addUserResult');

addUserResult.classList.remove('hidden');

addUserResult.innerHTML = `User added successfully: ${JSON.stringify(result.data)}`;

})

.catch(error => console.error('Error:', error));

});

// Handle add payment form submission

document.getElementById('addPaymentForm').addEventListener('submit', function (event) {

event.preventDefault();

const formData = new FormData(this);

const data = {};

formData.forEach((value, key) => {

data[key] = value;

});

fetch('/add\_payments', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(data)

})

.then(response => response.json())

.then(result => {

const addPaymentResult = document.getElementById('addPaymentResult');

addPaymentResult.classList.remove('hidden');

addPaymentResult.innerHTML = `Payment added successfully: ${JSON.stringify(result.data)}`;

})

.catch(error => console.error('Error:', error));

});

// Handle find user form submission

document.getElementById('findUserForm').addEventListener('submit', function (event) {

event.preventDefault();

const userId = document.getElementById('userId').value;

fetch(`/find\_user?userId=${userId}`)

.then(response => response.json())

.then(result => {

const userResult = document.getElementById('userResult');

userResult.classList.remove('hidden');

if (result.length > 0) {

userResult.innerHTML = createTable(result, ['c\_id', 'c\_name', 'c\_vill', 'c\_category', 'phone']);

} else {

userResult.innerHTML = 'No user found';

}

})

.catch(error => console.error('Error:', error));

});

// Handle find payments form submission

document.getElementById('findPaymentsForm').addEventListener('submit', function (event) {

event.preventDefault();

const userIdPayments = document.getElementById('userIdPayments').value;

fetch(`/find\_payments?userIdPayments=${userIdPayments}`)

.then(response => response.json())

.then(result => {

const paymentsResult = document.getElementById('paymentsResult');

paymentsResult.classList.remove('hidden');

if (result.length > 0) {

paymentsResult.innerHTML = createTable(result, ['p\_id', 'c\_id', 'c\_name', 'p\_month']);

} else {

paymentsResult.innerHTML = 'No payments found';

}

})

.catch(error => console.error('Error:', error));

});

// Handle view payments by month form submission

document.getElementById('viewPaymentsByMonthForm').addEventListener('submit', function (event) {

event.preventDefault();

const p\_month = document.getElementById('p\_month').value;

fetch(`/view\_payments\_by\_month?p\_month=${p\_month}`)

.then(response => response.json())

.then(result => {

const paymentsByMonthResult = document.getElementById('paymentsByMonthResult');

paymentsByMonthResult.classList.remove('hidden');

if (result.length > 0) {

paymentsByMonthResult.innerHTML = createTable(result, ['p\_id', 'c\_id', 'c\_name', 'p\_month']);

} else {

paymentsByMonthResult.innerHTML = 'No payments found for the specified month';

}

})

.catch(error => console.error('Error:', error));

});

// Handle delete user form submission

document.getElementById('deleteUserForm').addEventListener('submit', function (event) {

event.preventDefault();

const userId = document.getElementById('userIdDelete').value;

fetch(`/delete\_user?userId=${userId}`, {

method: 'DELETE'

})

.then(response => response.json())

.then(result => {

const deleteResult = document.getElementById('deleteResult');

const deletedUserDetails = document.getElementById('deletedUserDetails');

const paymentCount = document.getElementById('paymentCount');

deleteResult.classList.remove('hidden');

if (result.userDetails) {

deletedUserDetails.innerHTML = `User Details: ${JSON.stringify(result.userDetails)}`;

} else {

deletedUserDetails.innerHTML = 'User not found';

}

paymentCount.innerHTML = `Total Payments Made: ${result.paymentCount}`;

})

.catch(error => console.error('Error:', error));

});

// Handle search user form submission

document.getElementById('searchUserForm').addEventListener('submit', function (event) {

event.preventDefault();

const searchUserId = document.getElementById('searchUserId').value;

fetch(`/find\_user?userId=${searchUserId}`)

.then(response => response.json())

.then(result => {

if (result.length > 0) {

const user = result[0];

document.getElementById('updateUserId').value = user.c\_id;

document.getElementById('update\_c\_name').value = user.c\_name;

document.getElementById('update\_c\_vill').value = user.c\_vill;

document.getElementById('update\_c\_category').value = user.c\_category;

document.getElementById('update\_phone').value = user.phone;

document.getElementById('updateUserFormContainer').classList.remove('hidden');

} else {

alert('User not found');

}

})

.catch(error => console.error('Error:', error));

});

// Handle update user form submission

document.getElementById('updateUserForm').addEventListener('submit', function (event) {

event.preventDefault();

const formData = new FormData(this);

const data = {};

formData.forEach((value, key) => {

data[key] = value;

});

fetch('/update\_user', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(data)

})

.then(response => response.json())

.then(result => {

alert('User updated successfully');

document.getElementById('updateUserFormContainer').classList.add('hidden');

})

.catch(error => console.error('Error:', error));

});

});

</script>

</body>

</html>

**Backend:**

Server.js

const express = require('express');

const bodyParser = require('body-parser');

const mysql = require('mysql');

const path = require('path');

const app = express();

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json()); // Add this line to parse JSON request bodies

app.use(express.static(path.join(\_\_dirname, 'public')));

const db\_config = {

host: 'localhost',

user: 'root',

password: 'Immanuvel@12',

database: 'diwali'

};

const pool = mysql.createPool(db\_config);

app.get('/', (req, res) => {

res.sendFile(path.join(\_\_dirname, 'public/templates/index.html'));

});

app.post('/add\_user', (req, res) => {

const { c\_name, c\_vill, c\_category, phone } = req.body;

pool.getConnection((err, connection) => {

if (err) throw err;

const sql = 'INSERT INTO customers (c\_name, c\_vill, c\_category, phone) VALUES (?, ?, ?, ?)';

connection.query(sql, [c\_name, c\_vill, c\_category, phone], (err, results) => {

connection.release(); // Release the connection

if (err) throw err;

res.json({ message: 'User added successfully', data: { id: results.insertId, c\_name, c\_vill, c\_category, phone } });

});

});

});

app.post('/add\_payments', (req, res) => {

const { c\_id, p\_month } = req.body;

pool.getConnection((err, connection) => {

if (err) throw err;

const sql = 'INSERT INTO payments (c\_id, p\_month) VALUES (?, ?)';

connection.query(sql, [c\_id, p\_month], (err, results) => {

connection.release(); // Release the connection

if (err) throw err;

res.json({ message: 'Payment added successfully', data: { c\_id, p\_month } });

});

});

});

app.get('/find\_user', (req, res) => {

const userId = req.query.userId;

pool.getConnection((err, connection) => {

if (err) throw err;

const sql = 'SELECT \* FROM customers WHERE c\_id = ?';

connection.query(sql, [userId], (err, results) => {

connection.release(); // Release the connection

if (err) throw err;

res.json(results);

});

});

});

app.get('/find\_payments', (req, res) => {

const userId = req.query.userIdPayments;

pool.getConnection((err, connection) => {

if (err) throw err;

const sql = 'SELECT payments.\*, customers.c\_name FROM payments JOIN customers ON payments.c\_id = customers.c\_id WHERE payments.c\_id = ?';

connection.query(sql, [userId], (err, results) => {

connection.release(); // Release the connection

if (err) throw err;

res.json(results);});});

});

app.get('/view\_payments\_by\_month', (req, res) => {

const month = req.query.p\_month;

pool.getConnection((err, connection) => {

if (err) throw err;

const sql = 'SELECT payments.\*, customers.c\_name FROM payments JOIN customers ON payments.c\_id = customers.c\_id WHERE payments.p\_month = ?';

connection.query(sql, [month], (err, results) => {

connection.release(); // Release the connection

if (err) throw err;

res.json(results);});});

});

app.get('/total\_amount\_paid', (req, res) => {

const userId = req.query.userId;

pool.getConnection((err, connection) => {

if (err) throw err;

const sql = `

SELECT payments.p\_id, category.ct\_price

FROM payments

JOIN category ON payments.p\_month = category.ct\_name

WHERE payments.c\_id = ?

`;

connection.query(sql, [userId], (err, results) => {

connection.release(); // Release the connection

if (err) {

console.error("Error executing query:", err);

throw err;

}

let totalAmount = 0;

results.forEach((row) => {

const categoryPrice = row.ct\_price;

totalAmount += categoryPrice;

});

res.json({ total\_amount: totalAmount });});});

});

app.put('/update\_user', (req, res) => {

const { c\_id, c\_name, c\_vill, c\_category, phone } = req.body;

const sql = 'UPDATE customers SET c\_name = ?, c\_vill = ?, c\_category = ?, phone = ? WHERE c\_id = ?';

connection.query(sql, [c\_name, c\_vill, c\_category, phone, c\_id], (err, result) => {

if (err) throw err;

res.json({ message: 'User updated successfully' });});

});

app.delete('/delete\_user', (req, res) => {

const userId = req.query.userId;

pool.getConnection((err, connection) => {

if (err) throw err;

const userDetailsQuery = 'SELECT \* FROM customers WHERE c\_id = ?';

connection.query(userDetailsQuery, [userId], (err, userDetails) => {

if (err) {

connection.release();

throw err;

}

if (userDetails.length === 0) {

connection.release();

return res.json({ message: 'User not found' });

}

const paymentsCountQuery = 'SELECT COUNT(\*) AS payment\_count FROM payments WHERE c\_id = ?';

connection.query(paymentsCountQuery, [userId], (err, paymentsCountResult) => {

if (err) {

connection.release();

throw err;

}

const paymentCount = paymentsCountResult[0].payment\_count;

const deletePaymentsQuery = 'DELETE FROM payments WHERE c\_id = ?';

connection.query(deletePaymentsQuery, [userId], (err) => {

if (err) {

connection.release();

throw err;

}

const deleteUserQuery = 'DELETE FROM customers WHERE c\_id = ?';

connection.query(deleteUserQuery, [userId], (err) => {

connection.release();

if (err) throw err;

res.json({

message: 'User and associated payments deleted successfully',

userDetails: userDetails[0],

paymentCount: paymentCount

});});});});});});});

app.post('/update\_user', (req, res) => {

const { c\_id, c\_name, c\_vill, c\_category, phone } = req.body;

const sql = 'UPDATE customers SET c\_name = ?, c\_vill = ?, c\_category = ?, phone = ? WHERE c\_id = ?';

pool.query(sql, [c\_name, c\_vill, c\_category, phone, c\_id], (error, results) => {

if (error) return res.status(500).json({ error: error.message });

if (results.affectedRows > 0) {

res.json({ success: true });

} else {

res.status(404).json({ error: 'User not found' });

}

});

});

const PORT = 3000;

app.listen(PORT, () => {

console.log(`Server running on port http://localhost:${PORT}`);

});

|  |  |  |
| --- | --- | --- |
| CONTENTS | MARKS ALLOTED | MARKS OBTAINED |
| Aim, Algorithm, SQL, PL/SQL | 30 |  |
| Execution and Result | 20 |  |
| Viva | 10 |  |
| Total | 60 |  |

**RESULT:**

Thus the web application using mysql for payment management has been implemented successfully.