**‘Exercise : 1**

**AIM** : Execute DDL,DML,DCL and TCL commands on below given relational schema.

EMP(Empno,Ename,Job,Salary,Mgr,Comm,Hiredate,Deptno).

Description **:**

SQL(Structured Query Language) is a standard language for storing, manipulating and retrieving data in databases.

The SQL uses four different languages for the commands

They are:

1. DDL – Data Definition Language.
2. DML –Data Manipulation Language.
3. DCL- Data Control Language.
4. TCL - Transaction Control Language.

**Data Definition Language (DDL)**

Data definition language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database List of DDL commands :

* 1. CREATE
  2. DROP
  3. ALTER
  4. TRUNCATE
  5. RENAME

1. **CREATE :**

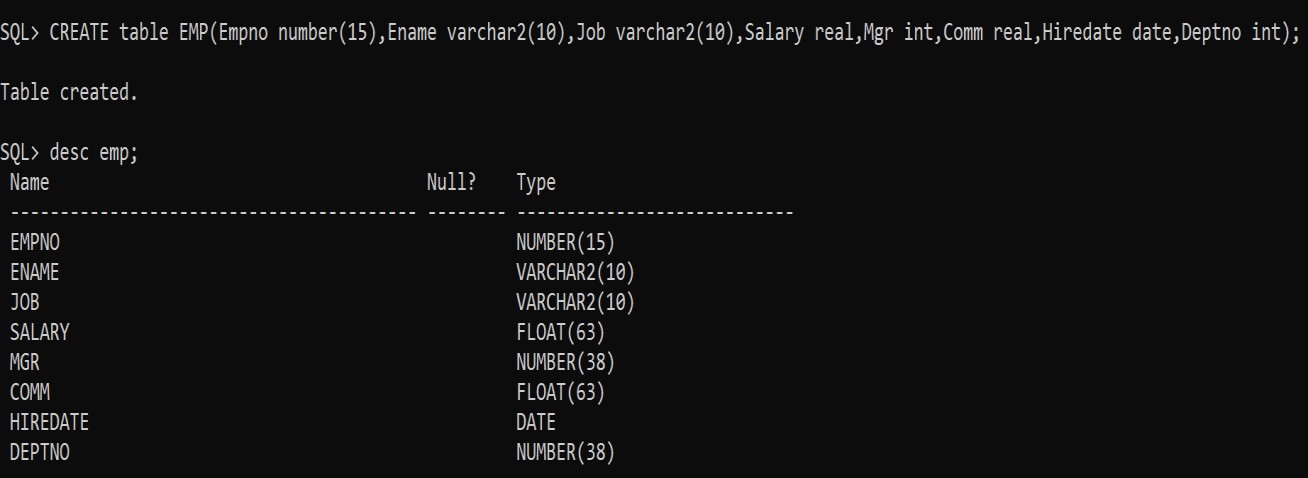
This command is used to create the database or its objects like table,index,function,views,store procedure and triggers.

**Syntax :**

CREATE table table\_name( column 1 domain type 1 , column 2 domain type 2 , …);

**Example :**

CREATE table EMP(Empno number(15),Ename varchar2(10),Job varchar2(10),Salary real,Mgr int,Comm real,Hiredate date,Deptno int); desc emp;



1. **DROP :**

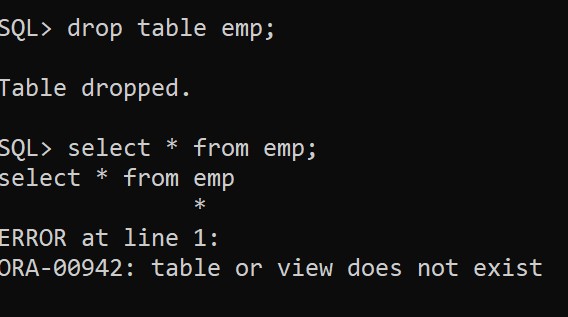
This command is used to delete objects from database.

**Syntax :**

DROP table table\_name; **Example :**

DROP table EMP;

Select \* from emp;



1. **ALTER :**

This is used to alter the structure of the database.

**Syntax :**

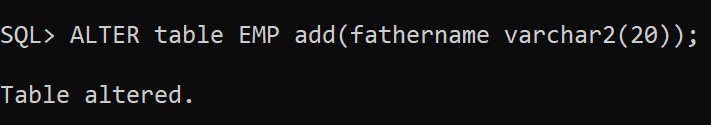
ALTER TABLE - ADD COLUMN :

ALTER table table\_name add column\_name domain type ; ALTER TABLE – DROP COLUMN :

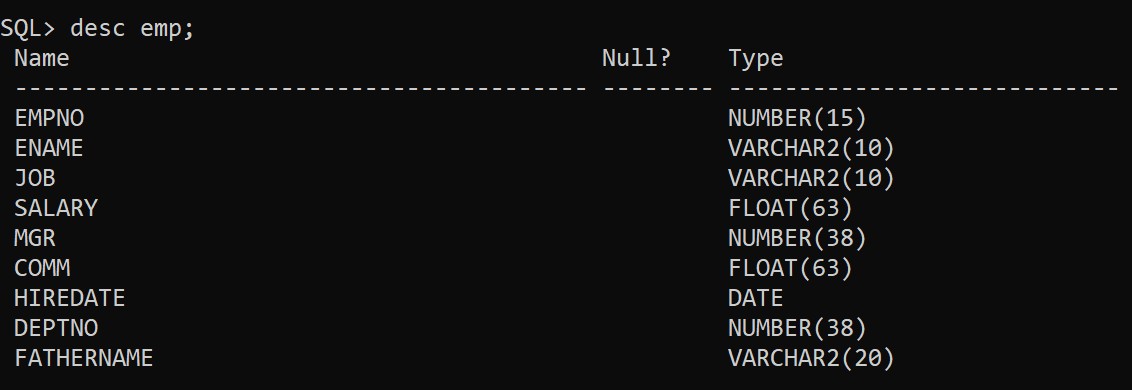
ALTER table table\_name DROP column column\_name; ALTER TABLE – MODIFY COLUMN :

ALTER table table\_name MODIFY column\_name datatype ; **Example :**

ALTER table EMP add(fathername varchar2(20)) ;

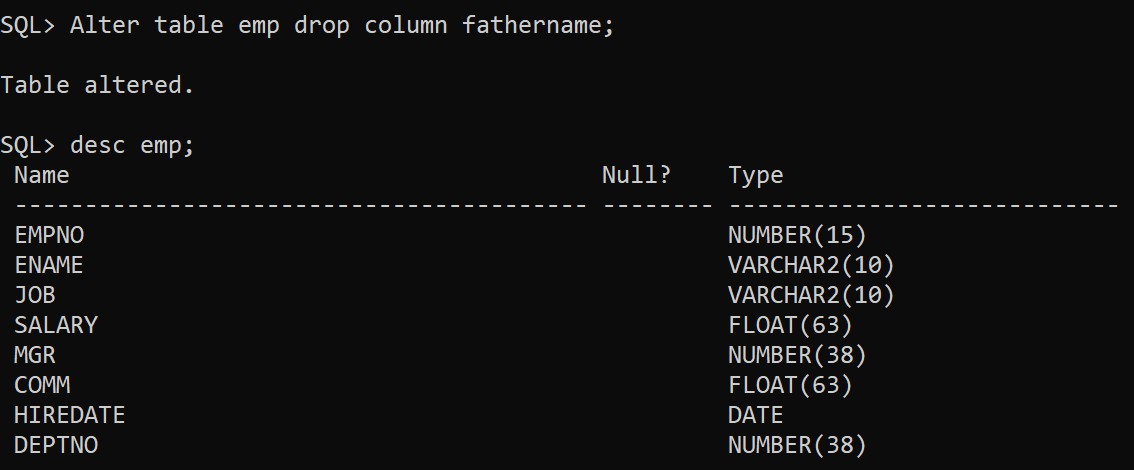


Desc emp;



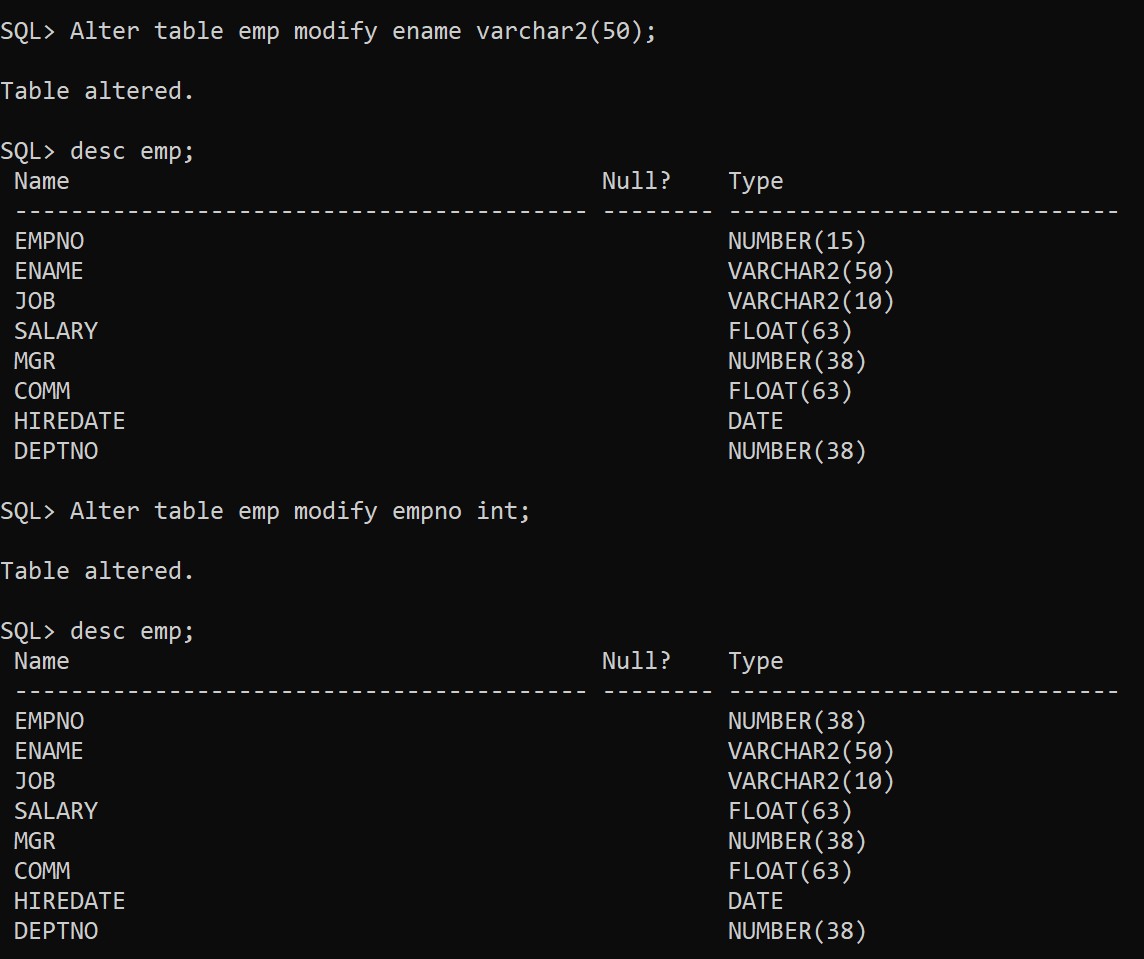
Alter table emp drop column fathername;

Desc emp;



Alter table emp modify name varchar2(20);

Alter table emp modify empno int;



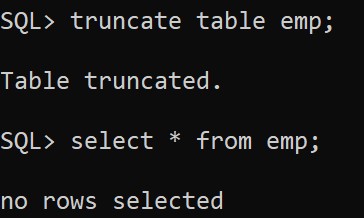
**4.TRUNCATE :**

This is used to remove all records from a table , including all spaces allocated for the records are removed. **Syntax :**

TRUNCATE table table\_name **Example:**

TRUNCATE table EMP;

Select \* from emp;



**5. RENAME**

Rename will be in two situations.

1. To change the name of the table.
2. To change the name of the column.

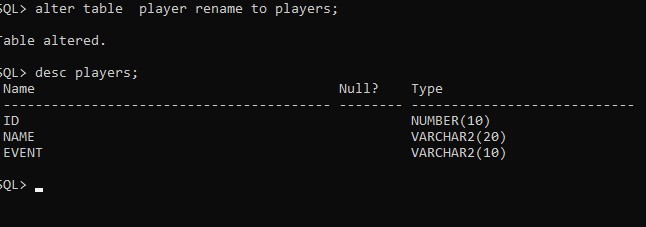
**Syntax**

* 1. alter table tablename rename to players.

**Example**

alter table player rename to players; Table altered. desc players;

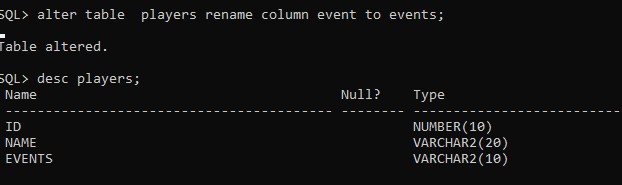
Output



* 1. alter table tablename column<old-column> to <new-coloumn>

**Example** alter table players rename column Event to Events; table altered. desc players;

Output



**Data Manipulating Language (DML) :**

The SQL commands that deals with the manipulation of data present in the data .

DML is the component of SQL statement that controls access to data and to the database.

List of DML commands :

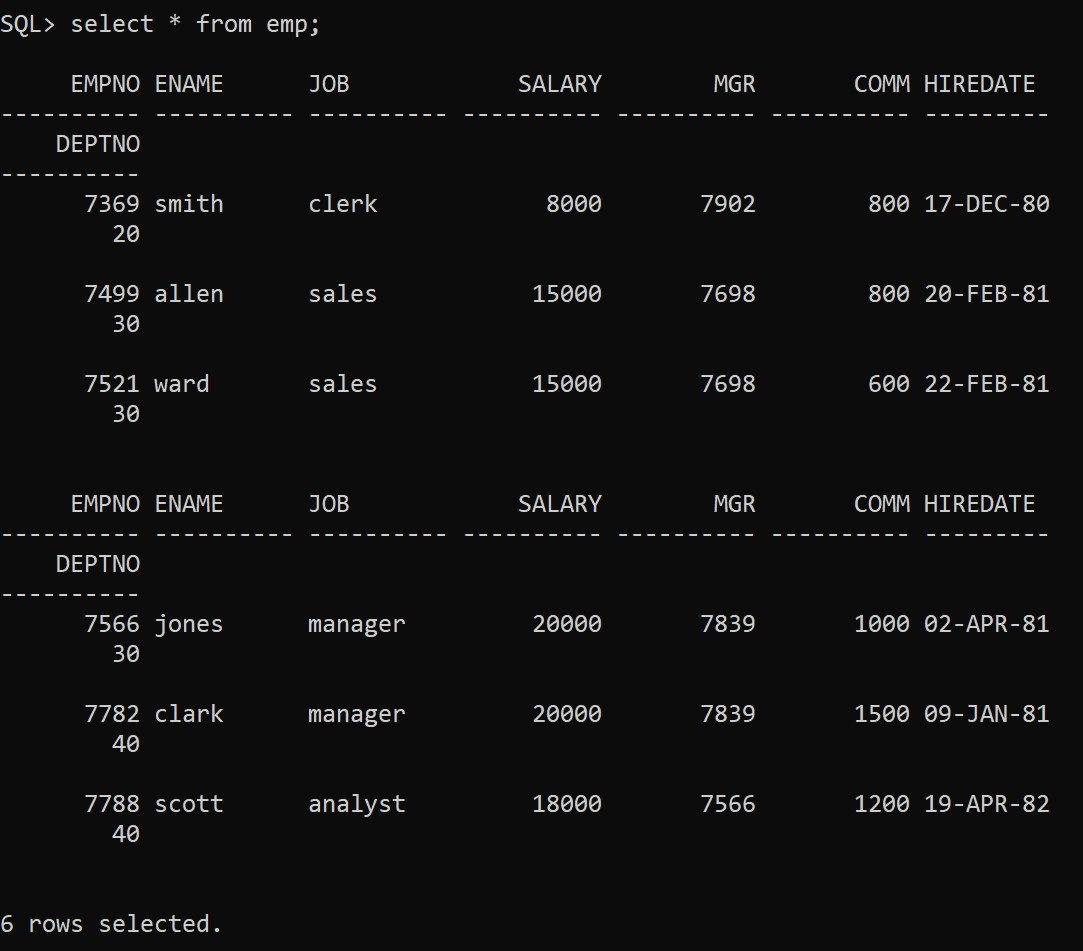
* 1. SELECT – it is used to retrieve data from the database.
  2. INSERT – it is used to insert data into a table.
  3. UPDATE – it is used to update existing data within a table.
  4. DELETE – it is used to delete records from a database table.

1. **SELECT :** It is used to select data from a database. The data returned is stored in a result table, called the result set.

**Syntax :**

SELECT \* FROM table\_name ; **Example :**

Select \* from EMP;

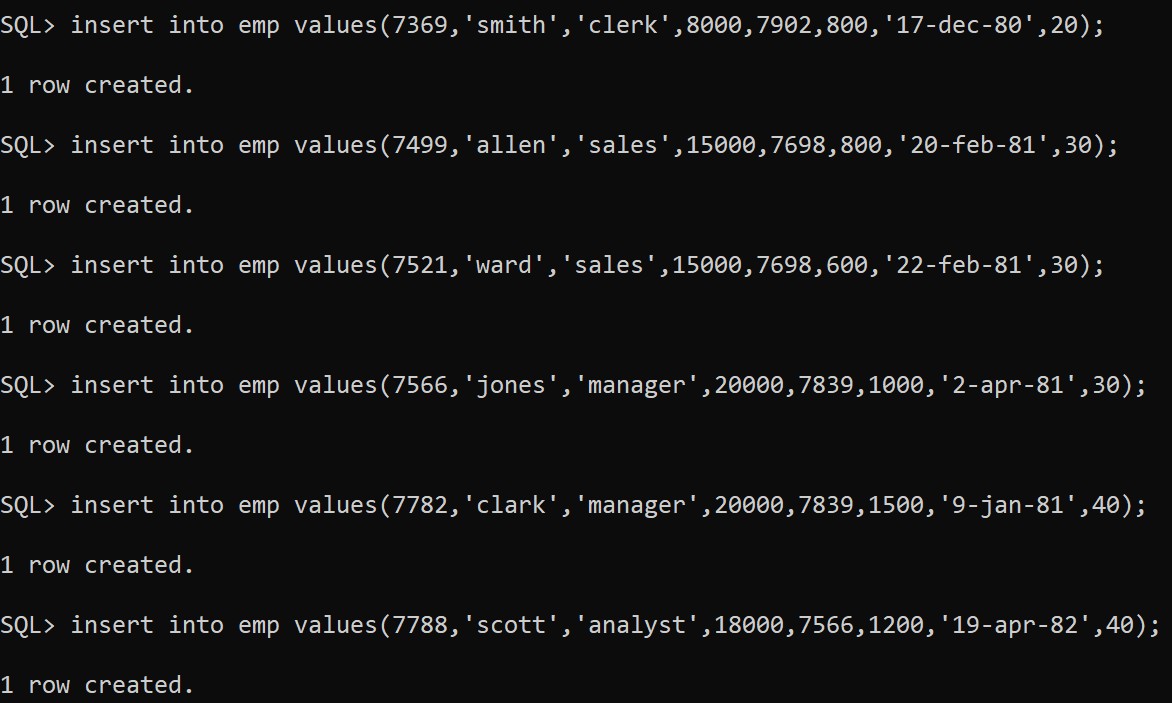


1. **INSERT :**It is an SQL command used to insert new rows in a table.

**Syntax :**

INSERT INTO table\_name values(value1 , value 2 , …); **Example :**

insert into emp values(7369,'smith','clerk',8000,7902,800,'17-dec-80',20); insert into emp values(7499,'allen','sales',15000,7698,800,'20-feb-81',30); insert into emp values(7521,'ward','sales',15000,7698,600,'22-feb-81',30) insert into emp values(7566,'jones','manager',20000,7839,1000,'2-apr-81',30); insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40); insert into emp values(7788,'scott','analyst',18000,7566,1200,'19-apr-82',40);



1. **UPDATE :** It is an SQL command used to update existing rows in a table.

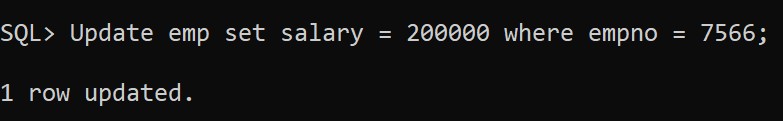
**Syntax**

UPDATE table\_name

SET attribute = value

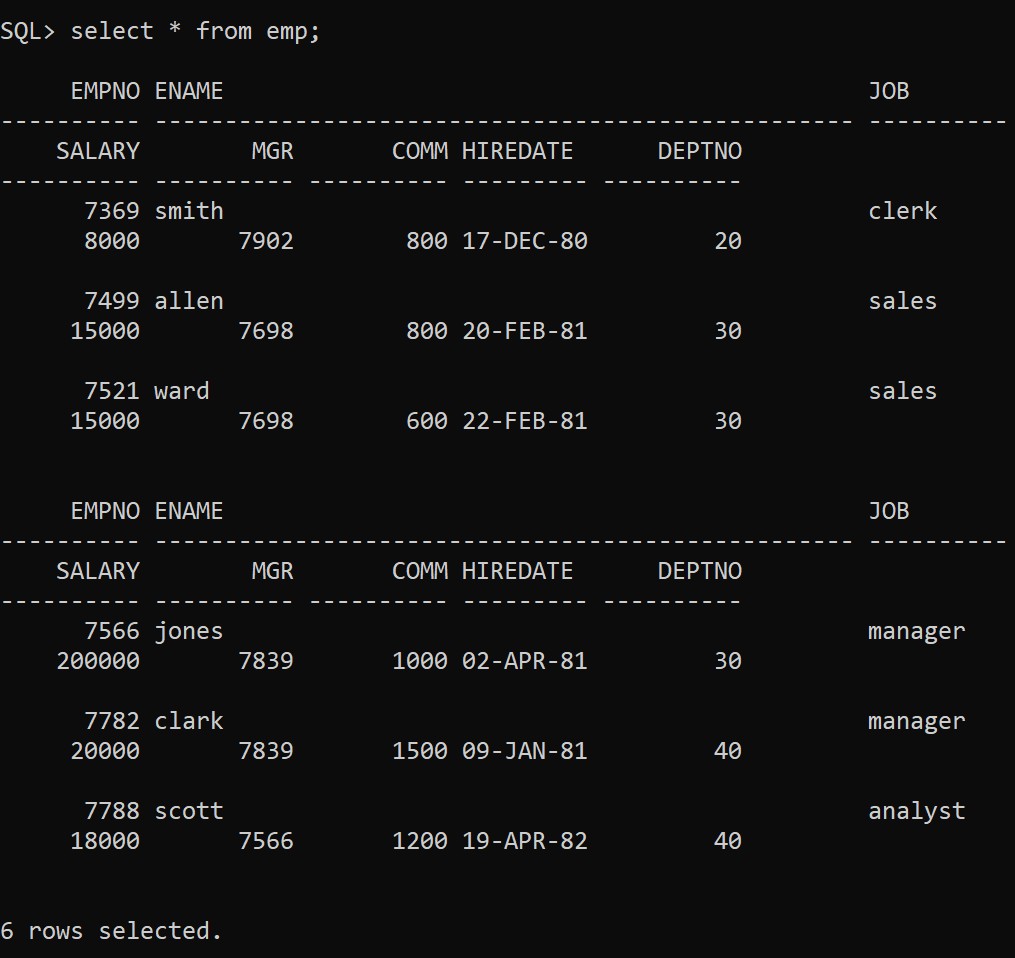
WHERE condition; **Example :**

Update emp set salary = 200000 where empno = 7566;



...

select \* from emp;



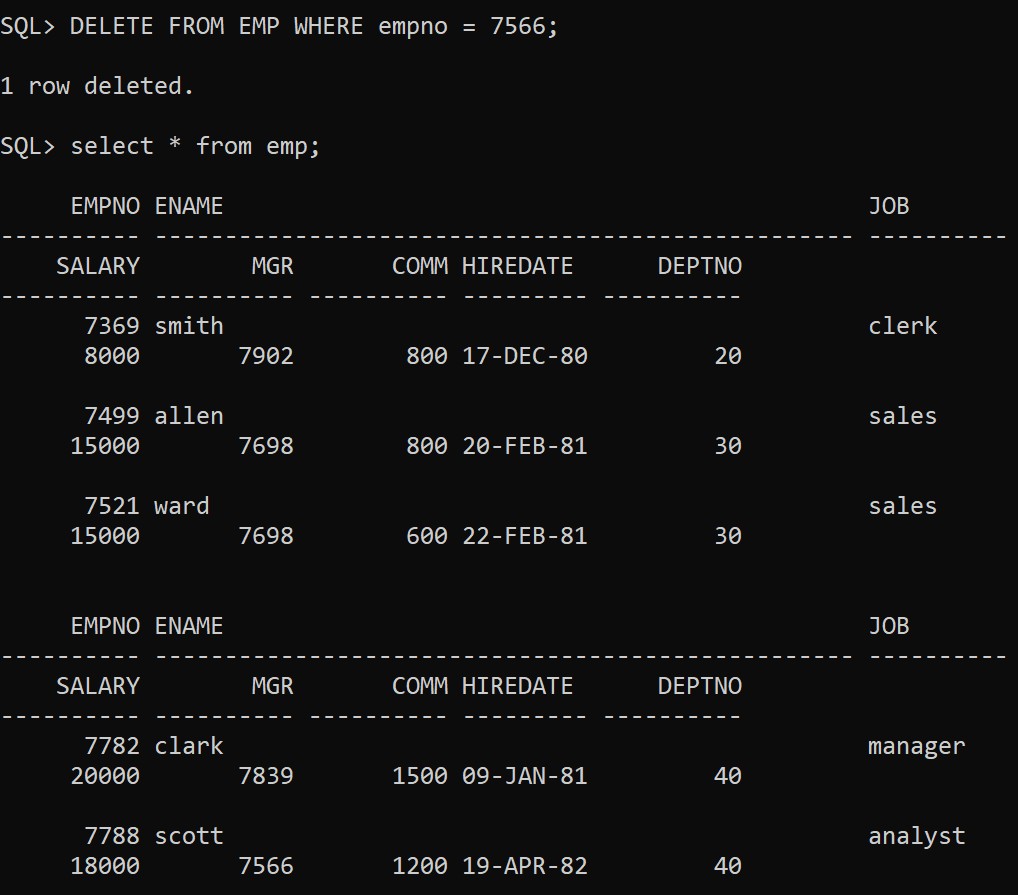
1. **DELETE :**The delete command is an SQL command used to delete existing records in a table.

**Syntax :**

DELETE FROM table\_name WHERE condition ; **Example :**

DELETE FROM EMP WHERE empno = 7566;

Select \* from emp;



**Data Control Language (DCL) :**

DCL commands mainly deals with the rights , permissions , and other controls of the database system.

List of DCL commands :

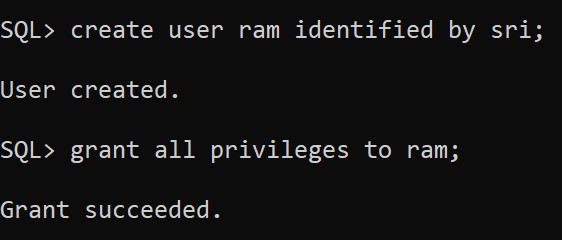
1. GRANT – this command gives users access privileges to the database.
2. REVOKE – this command withdraws the users access privileges given by the GRANT command.
3. **GRANT :** SQL grant command is specifically used to provide privileges to database objects for a user. This command also allows users to grant permissions to other users too.

**Syntax**

Grant privilege\_name on object\_name to {usesr\_name}; **Example :**

Create user ram identified by sri; User created.

Grant all privileges to ram; Grant succeded.

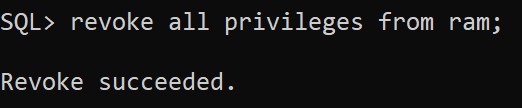


1. **REVOKE :**Revoke command withdraw user privileges on database objects if any granted.

**Syntax :**

Revoke privilege\_name on onject\_name from {user\_name}; **Example:**

Revoke all privileges from ram;



**Transaction Control Language(TCL) :**

List of TCL commands :

* 1. COMMIT – commits a transaction.
  2. ROLLBACK – rollbacks a transaction in case of any error occurs.
  3. SAVEPOINT – sets a savepoint within a transaction.

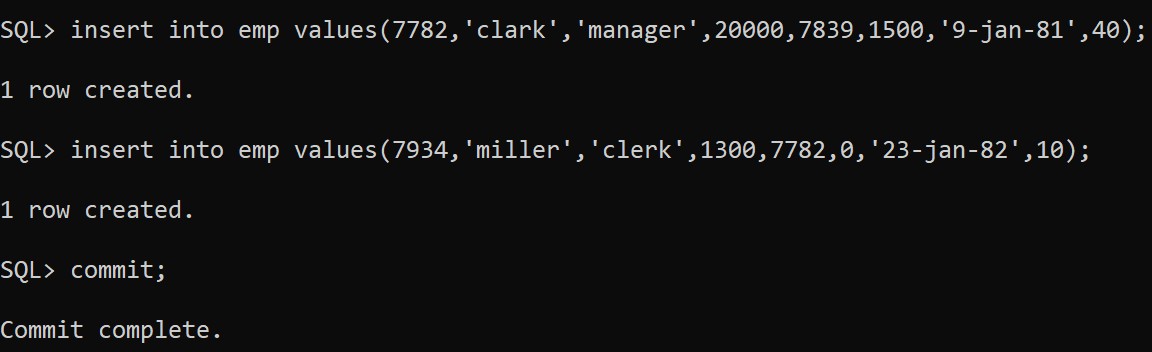
1. **COMMIT :** Commit command is used to save all transactions to the database.

**Syntax :**

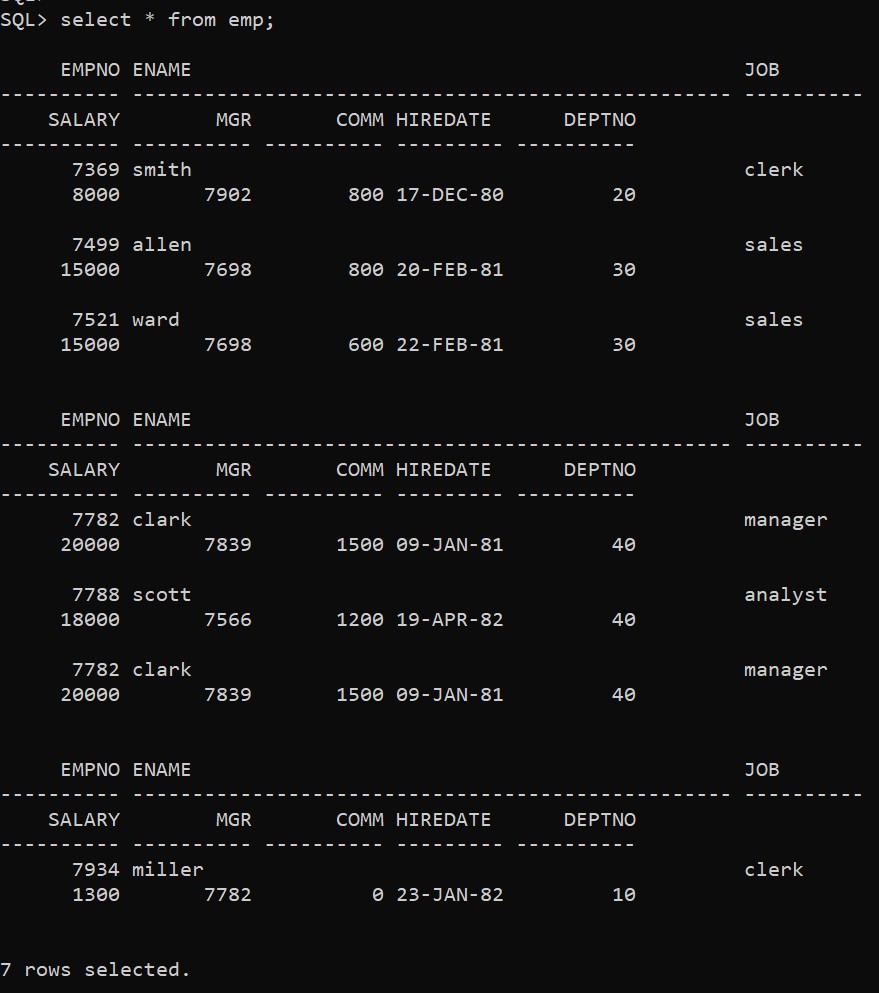
COMMIT;

**Example :**

insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40); insert into emp values(7934,'miller','clerk',1300,7782,0,'23-jan-82',10); Commit;



Select \* from emp;



1. **ROLLBACK :**

It is used to undo transactions that have not already been saved to the database.

**Syntax :**

ROLLBACK;

**Example :**

insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10); 1 row created.

insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created. savepoint A; Savepoint created.

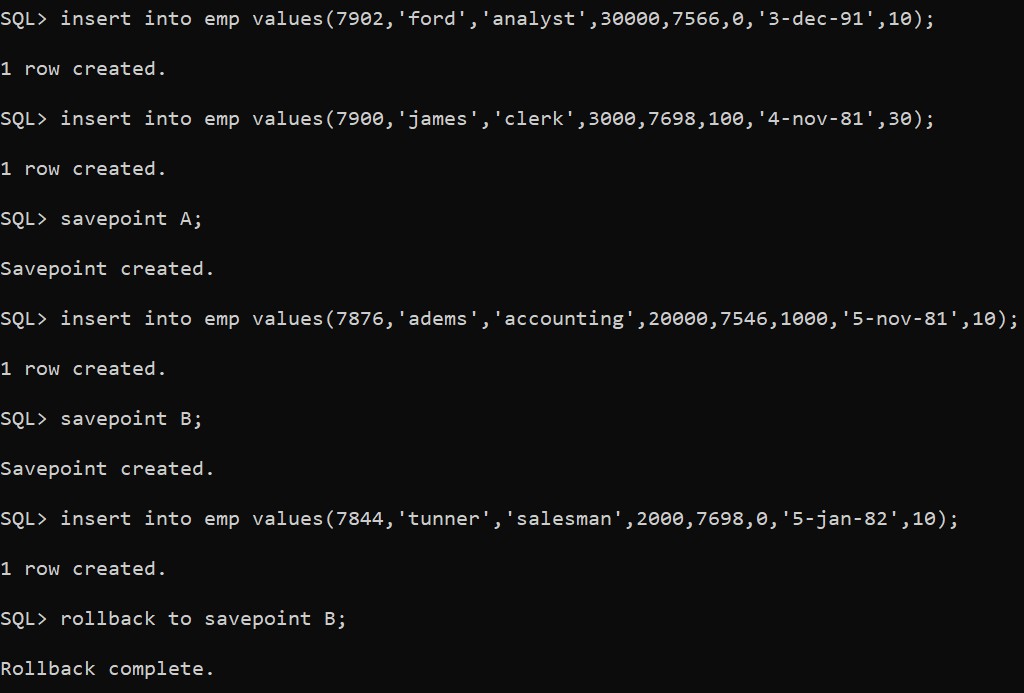
insert into emp values(7876,'adems','accounting',20000,7546,1000,'5-nov-81',10);

1 row created. savepoint B; Savepoint created.

insert into emp values(7844,'tunner','salesman',2000,7698,0,'5-jan-82',10);

1 row created.

rollback to savepoint B; Rollback complete.

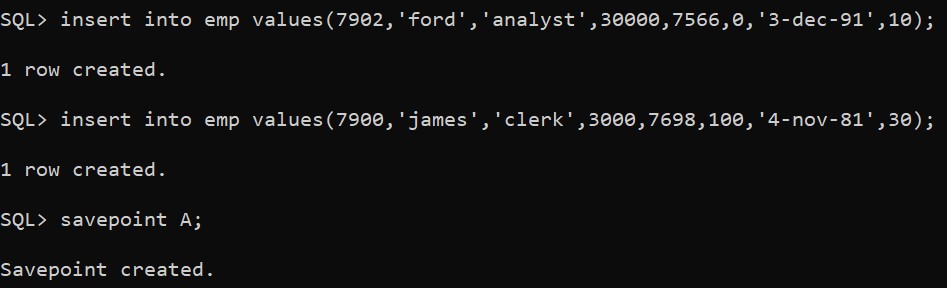


**3. SAVEPOINT :** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax :**

SAVEPPOINT savepoint\_name; **Example :** insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10); 1 row created.

insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30); 1 row created. savepoint A;



**VIVA-VOCE QUESTIONS**

1. List out DDL, DML, TCL and DCL commands.
2. Difference between Truncate and Drop.
3. Difference between Commit and Savepoint.
4. Creation of a table.

**EXERCISE : 2**

**AIM** : Implement the following integrity constraints on the following database EMP (Empno,

Ename, Job, Salary, Mgr, Comm, Hiredate, Deptno) DEPT(Deptno, Dname,

Location) a. Primary Key b.

Foreign Key c. Unique d. Not NULL e. Check

**Description :**

**Constraints**

* **KEY CONSTRAINTS** 
  + **Super key :** set of one or more attributes that uniquely identifies a tuple in a relation is called as a super key.
  + **Candidate key :** minimal set of attributes that uniquely identifies a tuple in a relation is called as a candidate key.
  + **Primary key :** it is a key which uniquely identifies a tuple in a relation . the two properties of primary key are unique and not null.
  + **Alternate key:** an alternate key is a key that can be work as a primary key .basically it is a candidate key that is not a primary key.
  + **Foreign key:** ensure that referential integrity of the data in one table to match values in another table . ensure that the foregin key in the child table match with the primary key in the parent table.

* **INTEGRITY CONSTRAINTS** 
  + **Unique key** : unique key is a set of one or more fields/columns of a table that uniquely identify a record in database table .it is like primary key but it can accept only one null value and it cannot have duplicate values.
  + **Check :** ensures that the value in a field meets a specified condition.
  + **Not NULL :** indicates that a field cannot store a NULL value.

* **Constraints according to the aim :**

**a) Primary Key constraint :**

The primary key constraint uniquely identifies each record in atable. They must contain UNIQUE values and cannot contain NULL values. A table can have only ONE primary key and in the table, this primary key can consist of single or multiple columns/fields .

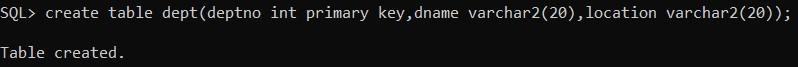
**Syntax :**

Create table table\_name(attribute name domaintype primary key , .. );

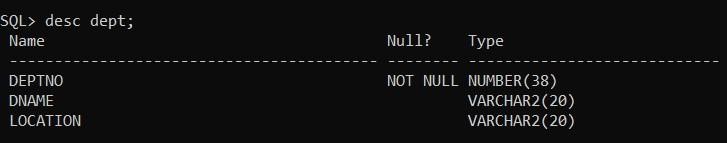
Or

**By using alter :**

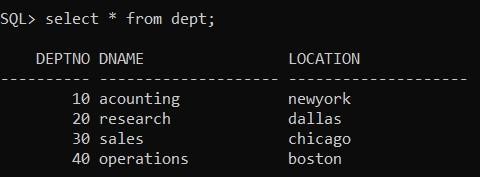
Alter table table\_name add constraint constraint\_name primary key(attribute); **Example :** create table dept(deptno int primary key,dname varchar2(20),location varchar2(20));



Desc dept;



Select \* from dept;



**b) Foreign Key Constraint:**

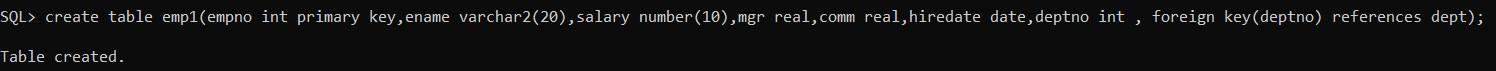
The foreign key constraint is used to prevent actions that would destroy links between tables. A foreign key is a field or a collection of fields in one table , that refers to the primary key in another table. The table with the foreign key is called the child table , and the table with the primary key is called the referenced or parent table.

**Syntax :**

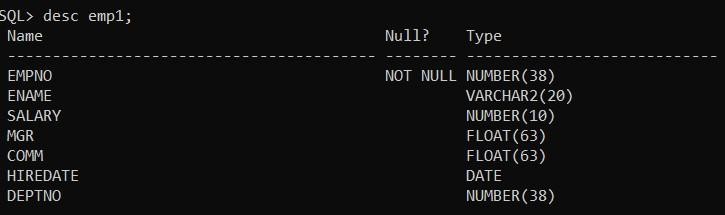
Create table table\_name(column domain type ,… , column n domain type n , foregin key(column) , references column in parent table);

**Example :**

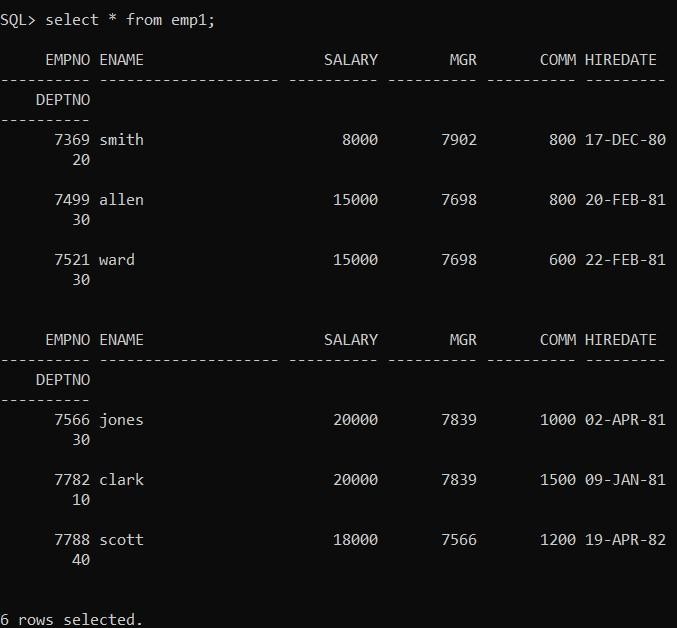
create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept);



Desc emp;



Select \* from emp;



**It has two attributes :**

1. **ON DELETE CASCADE :**

when a primary key is deleted in the parent table then corresponding data in the child table also gets deleted.

**Syntax :**

Create table table\_name(attribute domain type , foregin key(attribute) references parent table ON

DELETE CASCADE):

**Example :**

create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE CASCADE);



1. **ON DELETE SET NULL :**

When a primary key and its corresponding tuples gets deleted in parent table and then corresponding records in the child table will have the foreign key field set to null but not get deleted.

**Syntax :**

Create table table\_name(attribute domain type , foregin key(attribute) references parent table ON

DELETE SET NULL):

**Example :**

create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE SET NULL);



1. **Unique Key Constraint :**

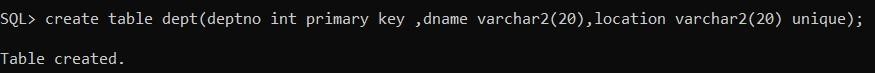
The unique constraint imposes that every value in a column or set of columns by unique. It means that no two rows of a table can have duplicate values in a specified column or set of columns.

**Syntax:**

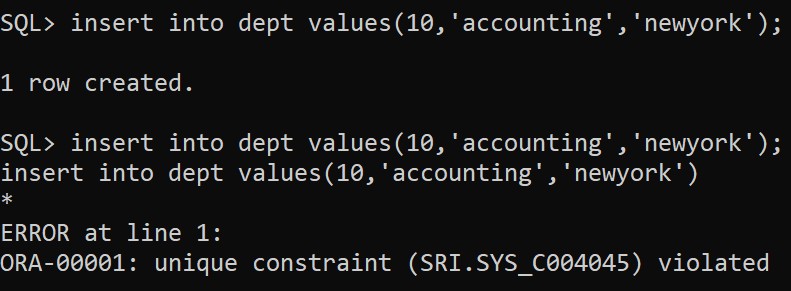
**While creating table :**

**Syntax :**

Create table table\_name(coloumn domain type unique); **Example :** create table dept(deptno int primary key ,dname varchar2(20),location varchar2(20) unique);



Unique constraint violation :

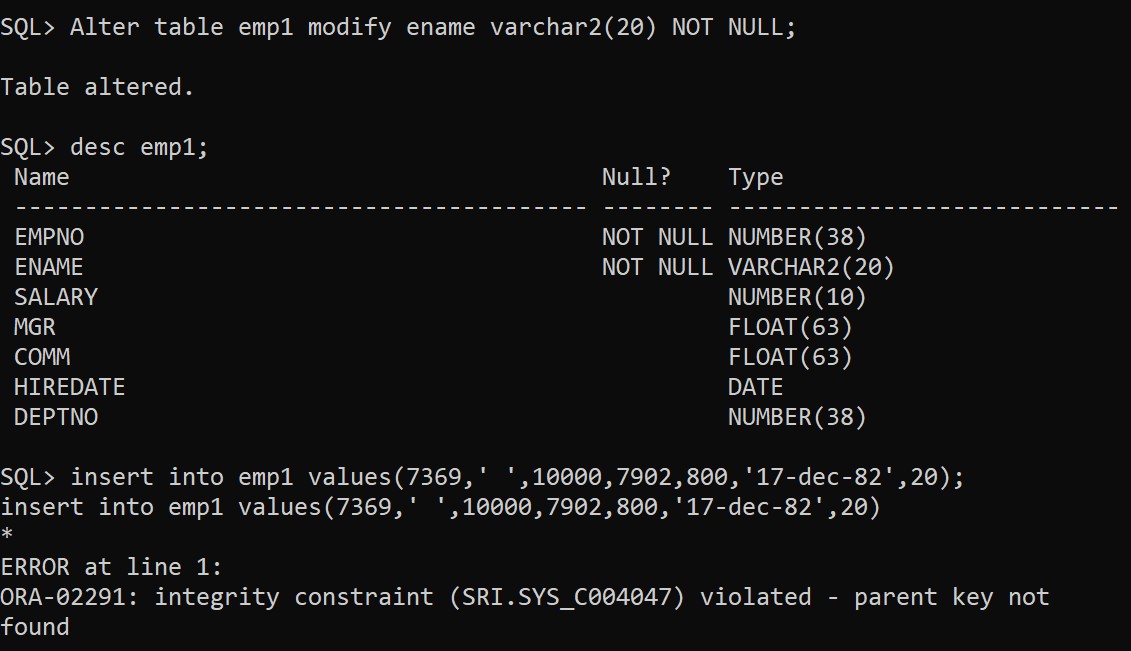


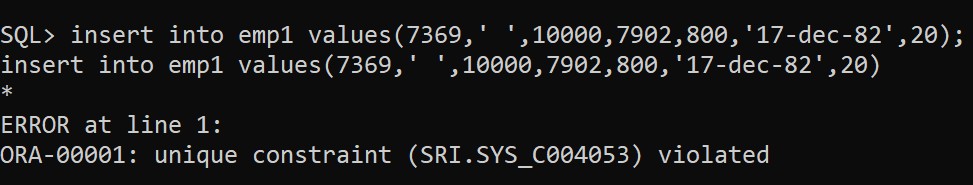
1. **NOT NULL:** Indicates that a column cannot store NULL value.

**Syntax :**

Alter table table\_name modify attribute domain type NOT NULL; **Example :**

Alter table emp modify attribute varchar2(20) NOT NULL;





1. **Check :** Ensures that the value in a column meets a specific condition. **Syntax :**

Alter table table\_name add constraint constraint\_name check(condition); **Example :**

Alter table emp add constraint s3 check(salary > 500 and salary < = 20000);



Select \* from emp;



**bEXERCISE :2a**

**AIM : Execute Set opersations on various Relations**

* **UNION • UNION ALL • INTERSECT**
* **MINUS.**

**Description :**

**Set operations in sql:**

**UNION :**

**Let R and S are two union compatible relations then, union operation returns the tuples that are present in R or s or both.**

* **Two relational instances are said to be union compatible if the following conditions are hold.** 
  1. **They have the same number of columns.**
  2. **Corresponding columns taken in order from left to right have same data type.**
  3. **Find the names of sailors who have reserved red or green boat.99 Query**

**select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'**

**UNION**

**select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'green';**

**Output**

* 1. **Find all sid’s of sailors who have rating of 10 or reserved boat no.104.**

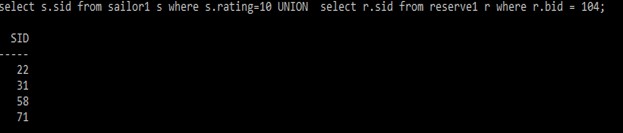
**Query :**

**select s.sid from sailor1 s where s.rating=10**

**UNION**

**select r.sid from reserve1 r where r.bid = 104;**

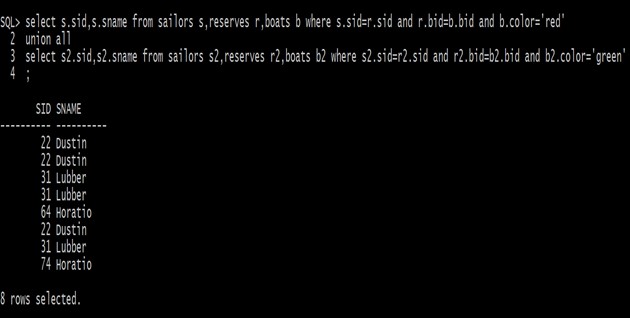
**Output**

**Us 2**

**UNION ALL :**

**The UNION ALL command combines the result set of two or more SELECT statements (allows duplicate values). Syntax :**

**Example :**

**22**

**INTERSECT :**

**Let R and S are two union compatible relations then, intersect operation returns the tuples that are common in both the relations.**

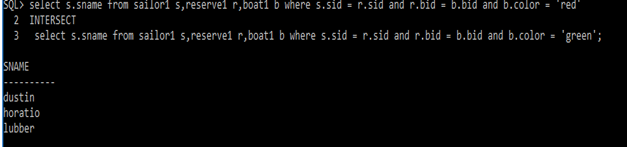
**1. Find the names of sailors who have reserved red and green boat. Query**

**select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'**

**INTERSECT**

**select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'green';**

**Output**



**MINUS :**

**Let R and S are two union compatible relations then, intersect operation returns the tuples that are present in R but not in S.**

**1. Find the sid’s of sailors who have reserved red but not green boat. Query select r.sid from boats b,reserves r where r.bid=b.bid and b.color='red' minus select r.sid from boats b,reserves r where r.bid=b.bid and b.color='green';**

**Output:**

**VIVA QUESTION:**

1. **List various set Operations**
2. **Differentiate UNION and UNION ALL**

**EXERCISE : 2b**

**AIM : Execute Sub Queries and Co-Related Nested Queries on Relations.**

* **Implement**

**o Single-row subquery o Multiple-row subquery**

* **Using Group Functions in a Subquery**
* **Using HAVING Clause with Subqueries**
* **Using Null Values in a Subquery**
* **Data retrieval using Correlated Subqueries o EXISTS Operator o NOT EXISTS Operator .**

**Description :**

**NESTED QUERIES**

**A query embedded inside another query is called a sub query. Inner query executes initially only once and that result will be used by all the tuples of outer query.**

**Co-Related nested queries: Correlated subquery is a query in which the inner query is executed for each row of the outer query.**

**Implement :**

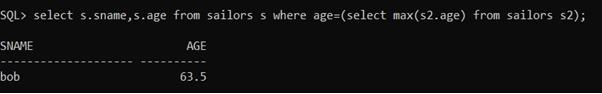
o **Single row subquery:**

**A single row subquery returns zero or one row to the outer SQL statement. You can place a subquery in a WHERE clause, a HAVING clause, or a FROM clause of a SELECT statement.**

**EXAMPLE :**

**1. Find the name and age of the oldest sailor.**

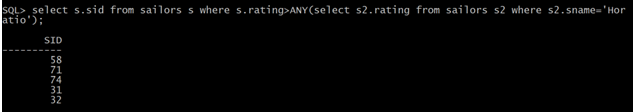
**select s.sname,s.age from sailors s where age=(select max(s2.age) from sailors s2);**



o **Multiple row subquery :**

* **Multiple-row subqueries are nested queries that can return more than one row of results to the parent query. Multiple-row subqueries are used most commonly in WHERE and HAVING clauses.**
* **Since it returns multiple rows, it must be handled by set comparison operators (IN, ALL, ANY).**
* **While IN operator holds the same meaning as discussed in the earlier chapter, ANY operator compares a specified value to each value returned by the subquery while ALL compares a value to every value returned by a subquery.**
* **The below query will show the error because single-row subquery returns multiple rows EXAMPLE :**

**1. Find sailors whose rating is better than some sailor called Horatio select s.sid from sailors s where s.rating select s.sid from sailors s where s.rating s2.sname='Ho**

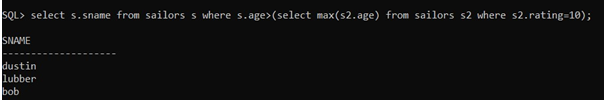


**Using Group functions in a subqueries :**

**EXAMPLE :**

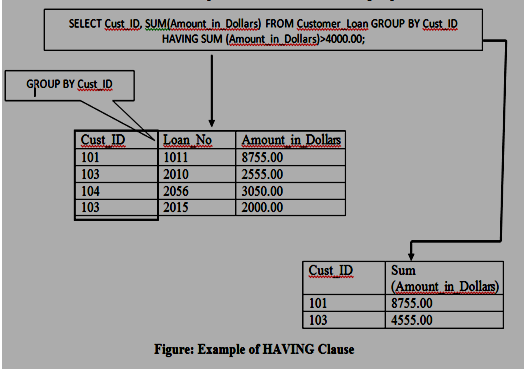
**1. Find the names of sailors who are older than oldest sailor with a rating of 10.**

**select s.sname from sailors s where s.age>(select max(s2.age) from sailors s2 where s2.rating=10);**



**Using HAVING clauses with subqueries :**

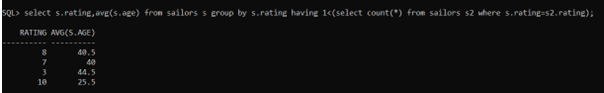
* **The HAVING clause is used along with the GROUP BY clause. The HAVING clause can be used to select and reject row groups.**
* **The format of the HAVING clause is similar to the WHERE clause, consisting of the keyword HAVING followed by a search condition.**
* **The HAVING clause thus specifies a search condition for groups.**



**EXAMPLE :**

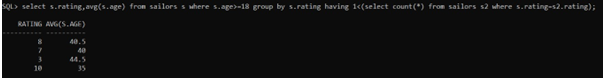
1. **Find the average age of sailors for each rating level that has at least two sailors.**

**select s.rating,avg(s.age) from sailors s group by s.rating having 1 <(select count(\*) from sailors s1 where s.rating=s1.rating);**



1. **Find the average age of sailors who are of voting age (i.e., at least 18 years old) for each rating level that has at least two sailors.**

**select s.rating,avg(s.age) from sailors s where s.age>=18 group by s.rating having 1<(select count(\*) from sailors s2 where s.rating=s2.rating);**



**Using NULL values in a subquery :**

**A field with a NULL value is a field with no value.**

**Syntax :**

**a. IS NULL :**

**Syntax**

**SELECT column\_names**

**FROM table\_name**

**WHERE column\_name IS NULL; IS NOT NULL :**

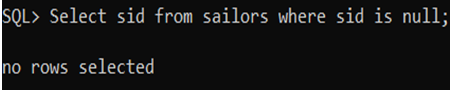
**Syntax**

**SELECT column\_names**

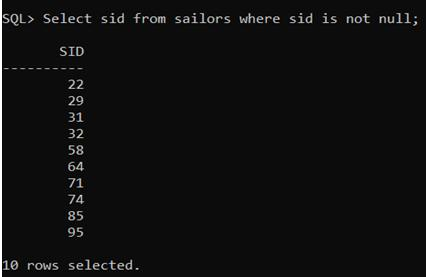
**FROM table\_name**

**WHERE column\_name IS NOT NULL; Example :**

**Select sid from sailors where sid is null;**



**Select sid from sailors where sid is not null;**



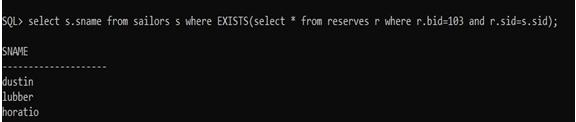
**Data retrieval using correlated sub queries:** o **EXISTS operator:**

**The EXISTS operator is used to test for the existence of any record in a subquery.**

**The EXISTS operator returns TRUE if the subquery returns one or more records.**

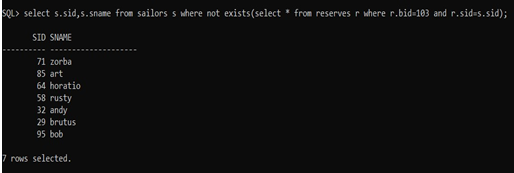
**Example:**

**1.Find the names of sailors who have reserved boat number 103. select s.sname from sailors s where EXISTS(select \* from reserves r where r.bid=103 and r.sid=s.sid);**

o **NOT EXISTS operator : Negated version of EXISTS Example:**

**1. Find the sid’s and names of sailors who have not reserved boat number 103.**

**select s.sid,s.sname from sailors s where not exists(select \* from reserves r where r.bid=103 and r.sid=s.sid);**



**EXERCISE : 3a**

**QUERIES USING AGGREGATE FUNCTIO**

**(COUNT, SUM, AVG, MAX AND MIN) GROUP BY and HAVING.**

**AIM : Execute the following Multiple row functions (Aggregate Functions) on**

**Relation**

* **Group functions(AVG, COUNT, MAX, MIN, SUM)**
* **DISTINCT Keyword in Count Function**
* **Null Values in Group Functions**
* **NVL Function with Group Functions. Description :**

**AGGREGATE FUNCTIONS**

**In data base management system ,an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.**

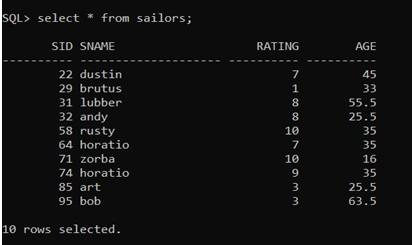
**The aggregate functions are:**

1. **MAX(): It returns the max value in the given column.**
2. **MIN(): It returns the max value in the given column.**
3. **SUM(): It returns the sum of all numeric values in the given column.**
4. **AVG(): It returns the average of all values in the given column.**
5. **COUNT():It returns the total number of all values in the given column(excluding null values).**
6. **COUNT(\*):It returns the number of all rows in the given table(including null values).**

**Group Functions(AVG , COUNT , MAX , MIN , SUM) :**

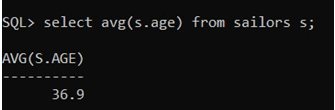
**Example :**

**SELECT \* FROM sailors;**



**AVERAGE (AVG):**

**Example : select avg(s.age) from sailors s;**



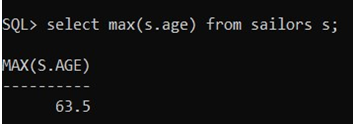
**select avg(s.age) from sailors s where s.rating=10;**



**MAXIMUN (MAX):**

**Example:**

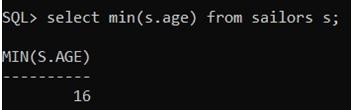
**select max(s.age) from sailors s;**



**MINIMUM (MIN):**

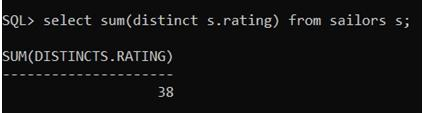
**Example:**

**select min(s.age) from sailors s;**



**SUM:**

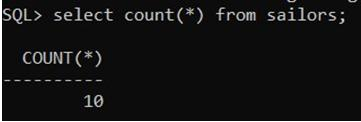
**Example: select sum(distinct s.rating) from sailors s;**



**COUNT:**

**Example:**

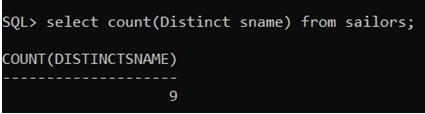
**select count(\*) from sailors;**



**DISTINCT keyword in count function :**

**EXAMPLE :**

**select count(Distinct sname) from sailors;**



**Null values in group functions :**

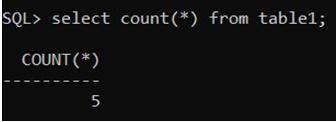
**EXAMPLE :**

**Create table table1 (id int, col int);**

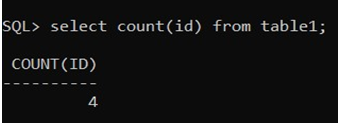
**Select \* from table1;**



**select count(\*) from table1;**



**select count(id) from table1;**

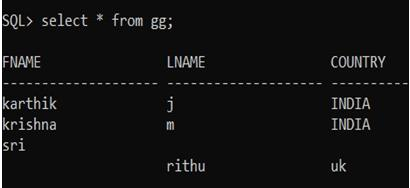


**NVL function with group functions :**

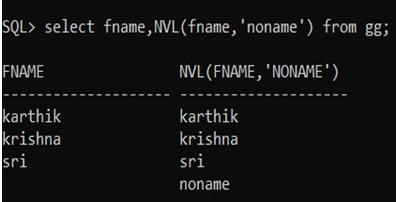
**EXAMPLE :**

**create table gg(fname varchar2(20),lname varchar2(20),country varchar2(10));**

**SELECT \* FROM GG;**



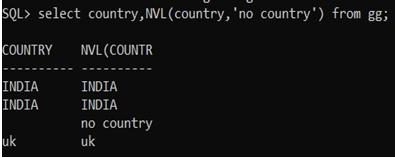
**select fname,NVL(fname,'noname') from gg;**



**select lname,NVL(lname,'empty') from gg;**



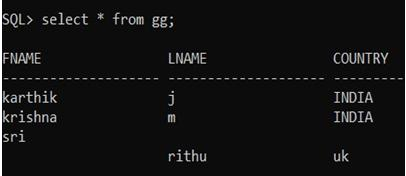
**select country,NVL(country,'no country') from gg;**



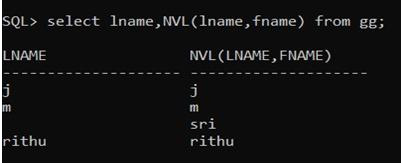
**select country,NVL(country,'0') from gg;**



**select \* from gg;**



**select lname,NVL(lname,fname) from gg;**



**EXERCISE : 3b**

**AIM : Create Groups of Data using Group By clause**

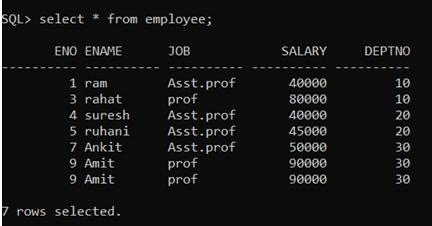
* **Grouping by One Column**
* **Grouping by More Than One Column**
* **Illegal Queries Using Group Functions • Restricting groups using HAVING Clause**
* **Nesting Group Functions. Description :**

**GROUP by clause :**

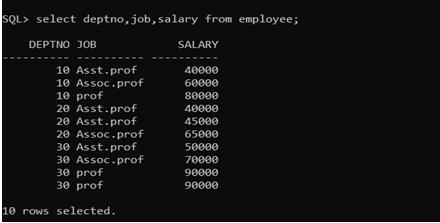
* **The GROUP BY clause is used in a SELECT statement to collect data across multiple records and group the results by one or more columns.**
* **Sometimes it is required to get information not about each row, but about each group.**
* **Related rows can be grouped together by the GROUP BY clause by specifying a column as a grouping column.**
* **In the output table all the rows with an identical value in the grouping column will be grouped together. Hence, the number of rows in the output is equal to the number of distinct values of the grouping column.**

**Grouping by More Than One Column EXAMPLE :**

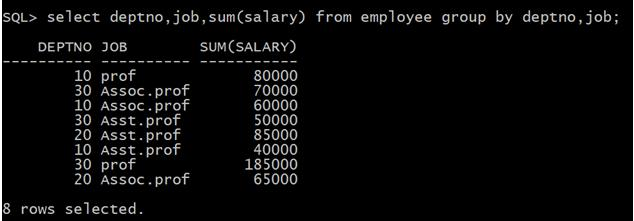
**create table employee(eno int,ename varchar2(10),job varchar2(10),salary int,deptno int); select \* from employee;**



**Query:Total salary paid to each job in each department select Deptno,job,salary from employee;**



**select deptno,job,sum(salary) Total\_Salary from employee group by deptno,job;**

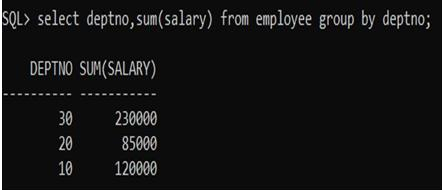


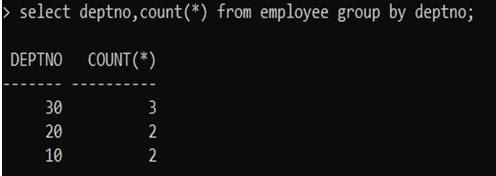
**Query: Total Salary paid to each job in each department excluding Assoc Prof**



**Grouping by one column :**

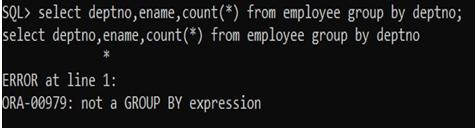
**EXAMPLE : select deptno,sum(salary) from employee group by deptno;**

 **select deptno,count(\*) from employee group by deptno;**

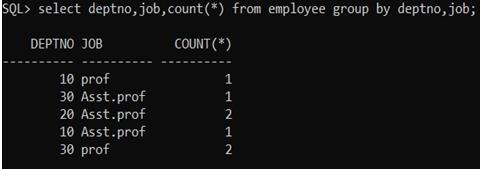


**Illegal Queries using Group functions :**

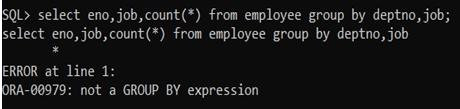
**EXAMPLE : select deptno,ename,count(\*) from employee group by deptno;**



**select deptno,job,count(\*) from employee group by deptno,job;**



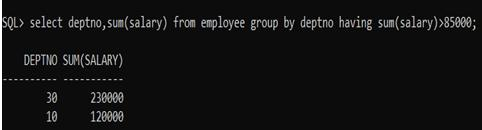
**select eno,job,count(\*) from employee group by deptno,job;**



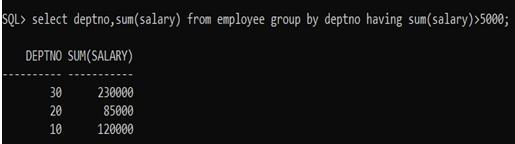
**Restricting groups using HAVING clause :**

**EXAMPLE :**

**select deptno,sum(salary) from employee group by deptno having sum(salary)>85000;**



**select deptno,sum(salary) from employee group by deptno having sum(salary)>5000;**



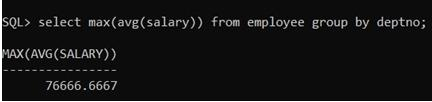
**Nesting group functions :**

**EXAMPLE :**

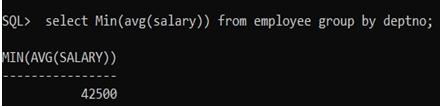
**select deptno,avg(slary) from employee group by deptno;**



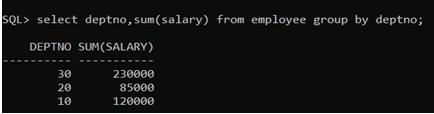
**select max(avg(salary)) from employee group by deptno;**



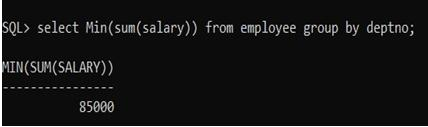
**select Min(avg(salary)) from employee group by deptno;**



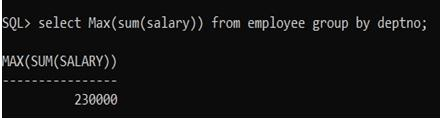
**select deptno,sum(salary) from employee group by deptno;**



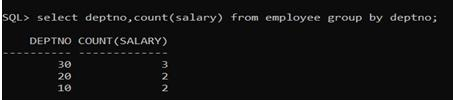
**select Min(sum(salary)) from employee group by deptno;**



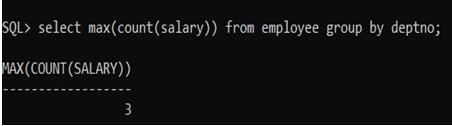
**select Max(sum(salary)) from employee group by deptno;**



**select deptno,count(salary) from employee group by deptno;**



**select max(count(salary)) from employee group by deptno;**



**EXERCISE : 4**

**QUERIES USING CONVERSION FUNCTIONS (TO\_CHAR, TO\_NUMBER AND TO\_DATE),**

**STRING FUNCTIONS (CONCATENATION, LPAD, RPAD, LTRIM, RTRIM, LOWER, UPPER,**

**INITCAP, LENGTH, SUBSTR AND INSTR), DATE FUNCTIONS (SYSDATE, NEXT\_DAY,**

**ADD\_MONTHS, LAST\_DAY, MONTHS\_BETWEEN, LEAST,**

**GREATEST, TRUNC, ROUND, TO\_CHAR)**

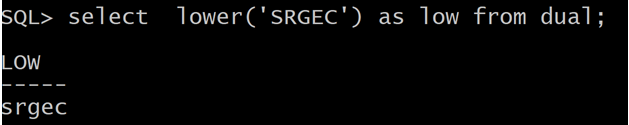
**Description :**

**a. Character functions :**

**Case – manipulation functions ( LOWER , UPPER , INITCAP ) :**

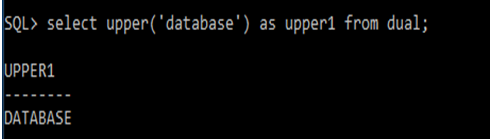
**1. lower (): this function converts the uppercase letters to lower case letters what you are passed to the function.**

**Syntax: lower(message) Example select lower('SRGEC') as low from dual;**



**2.upper(): this function is used to convert the lower case letters into uppercase letters.**

**Syntax: upper(message) Example select upper('database') as upper1 from dual;**

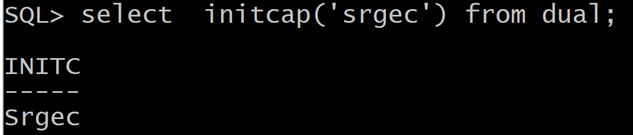


**3. initcap():**

**It make initial letter to capital letter what you have passed to the function.**

**Syntax:**

**initcap(message) Example : select initcap('srgec') from dual;**

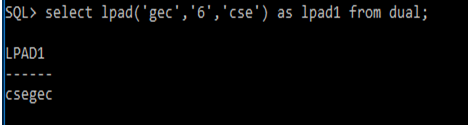


**Character manipulation functions (CONCAT , SUBSTR , LENGTH , INSTR , LPAD | RPAD , TRIM , REPLACE) :**

1. **lpad(): This function is used for attaching a new word to the original one at left side.**

**Syntax:**

**lpad(word1,length,word2) Example: select lpad('gec',’6','cse') as lpad1 from dual;**

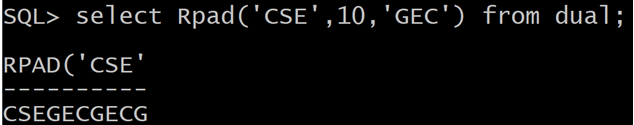


1. **rpad(): This function is used for attaching a new word to the original one at right side.**

**Syntax:**

**rpad(word1,length,word2)**

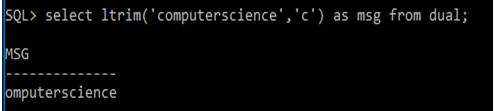
**Example: select Rpad('CSE',10,'GEC') from dual;**



1. **ltrm():This function is used for left trimming i.e, it delete(cut) the left most letter.**

**Syntax:**

**ltrim('message','character') Example: select ltrim('computerscience','c') as msg from dual;**

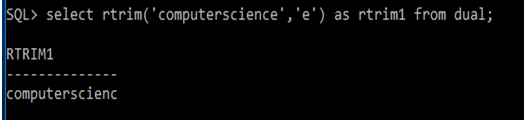


1. **rtrim()**

**This function is used for right trimming.**

**Syntax:**

**rtrim('message','character') Example: select rtrim('computerscience','e') as rtrim1 from dual;**



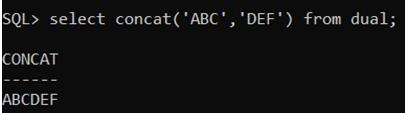
1. **concat():This function is used to add two strings.**

**Syntax:**

**Concat(‘string1’ , ‘string 2’)**

**Example :**

**select concat('ABC','DEF') from dual;**

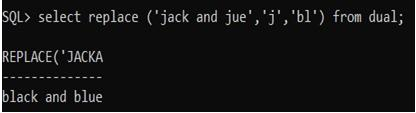


1. **Replace : This function is used to replace a particular character from a string.**

**Syntax :**

**Replace(‘string’, ‘replaceable char’,char); Example :**

**select replace ('jack and jue','j','bl') from dual;**

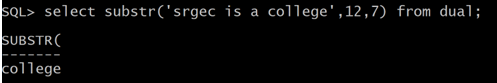


1. **substring :This function is used to extract the substring from a main string .**

**syntax :**

**substr(string , indexing , size); Example :**

**SQL> select substr('srgec is a college',12,7) from dual;**



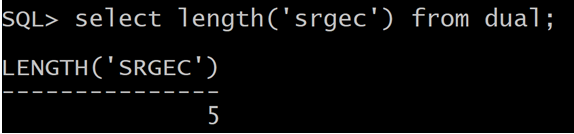
1. **length :This function is used to find the length of a given string.**

**Syntax :**

**Length(string);**

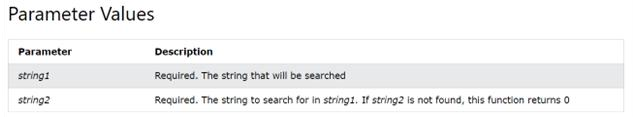
**Example:**

**SQL> select length('srgec') from dual;**

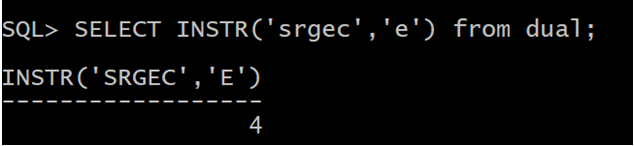


1. **INSTR():The INSTR() function returns the position of the first occurrence of a string in another string. This function performs a case-insensitive search.**

**Syntax: INSTR(string1, string2)**



**Example: SQL> SELECT INSTR('srgec','e') from dual; Output:**

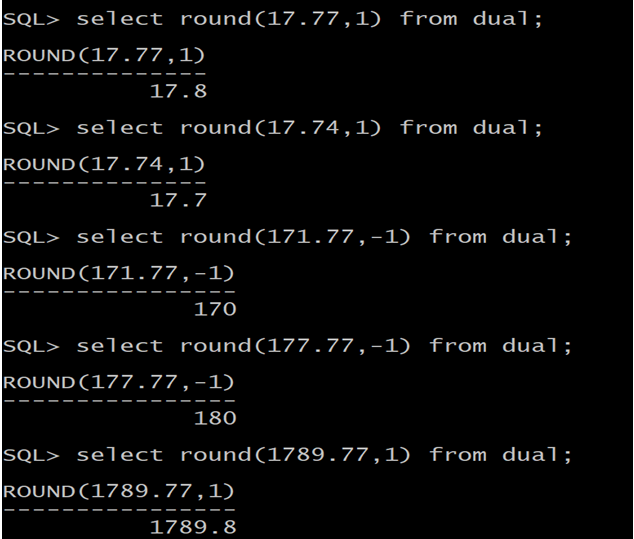


1. **Number functions (ROUND , TRUNCATE , MOD ) :**
2. **ROUND :**

**The ROUND() function rounds a number to a specified number of decimal places.**

**SYNTAX :**

**ROUND(*number*, *decimals*, *operation*) Example :**



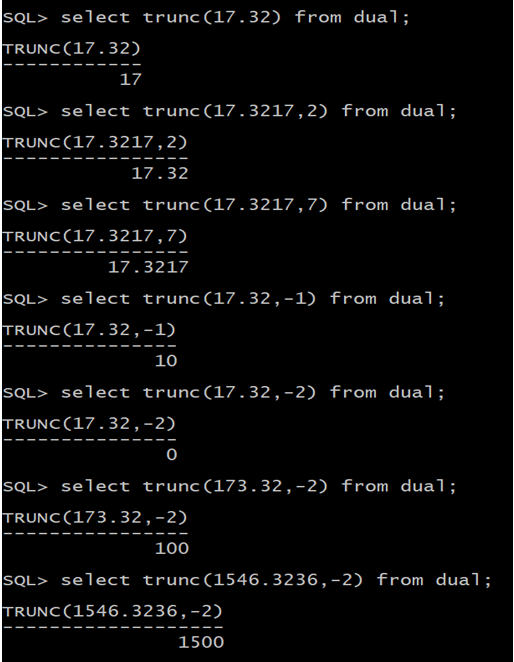
1. **TRUNCATE :**

**The TRUNCATE() function truncates a number to the specified number of decimal places.**

**Syntax :**

**TRUNCATE(*number*, *decimals*) Example :**

**Output :**

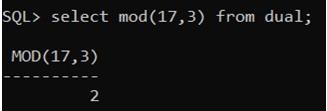


1. **MOD :**

**The MOD() function returns the remainder of a number divided by another number.**

**Syntax : MOD(x , y) Example :**

**select mod(17,3) from dual;**



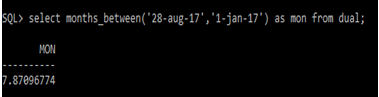
1. **Date functions :**

**Months \_ Between :It gives the number of months between specified two dates.**

|  |  |
| --- | --- |
| **Result value** | **Months\_between(date-exp1,date-exp2)** |
| **Negative result** | **If date-exp1 is earlier than date-exp2** |
| **Integer result** | **If date-exp1 and date-exp2 have the same day,or both specify th the month.** |
| **Decimal result** | **If days are different and they are not both specify the last day o** |
| **Fractional part** | **Always calcilated as the difference between days divided by 31 d number of days in the month.** |

**Syntax:**

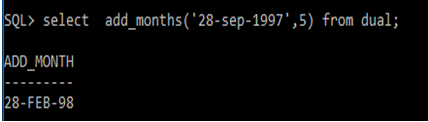
**months\_between(date1,date2) Example: select months\_between('28-aug-17','1-jan-17') as mon from dual;**



**Add \_Months : This function is used to add the 'n' number of months to a given date.**

**Example:**

**select add\_months('28-sep-1997',5) from dual;**

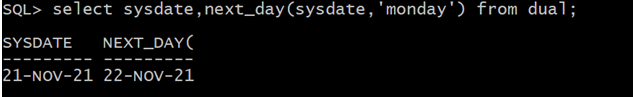


**Next\_Day :**

**Syntax : next\_day(date,dayname)**

**EXAMPLE :**

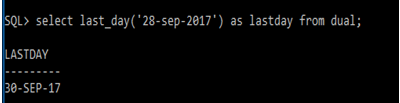
**SQL> select sysdate,next\_day(sysdate,'monday') from**

**dual;** 

**Last \_ Day : It gives the last day of the specified month in a date.**

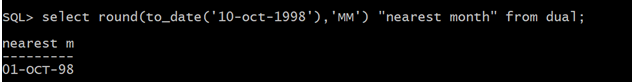
**Syntax:**

**last\_date(date) Example: select last\_day('28-sep-2017') as lastday from dual;**



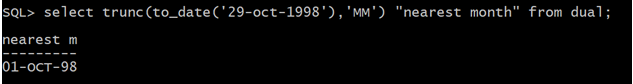
**Round :**

**The Round() Returns the date rounded by the specified format unit Example : select round(to\_date('10-oct-1998'),'MM') "nearest month" from dual;**



**Trunc : Truncates the specified date of its time portion according to the format unit provided.**

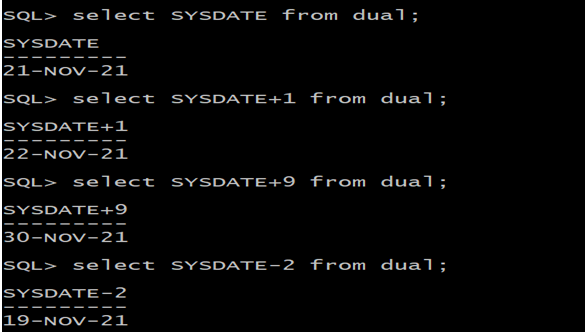
**Example :**



**Arithmetic with Dates:**

* **Add or subtract a number to or from a date for a resultant date value**
* **Subtract two dates to find the number of days between those dates.**
* **Add hours to a date by dividing the number of hours by 24**
* **Since the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates. You can perform the following operations:**





**Experiment 5**

1. **Create a simple PL/SQL program which includes declaration section, executable section and exception –Handling section (Ex. Student marks can be selected from the table and printed for those**

**who secured first class and an exception can be raised if no records were found).**

1. **Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block..**

**i). We have to create the student table and insert the records in to the table as follows: SQL> create table student(sid number(10),sname varchar2(20),rank varchar(10)); Table created.**

**SQL> insert into student values(501,'Ravi','second'); 1 row created.**

**SQL> insert into student values(502,'Raju','third'); 1 row created.**

**SQL> insert into student values(503,'Ramu',''); 1 row created.**

**SQL> select \*from student;**

**SID SNAME RANK**

1. **Ravi second**
2. **Raju third**
3. **Ramu**

**PL/SQL CODE:**

**SQL>ed 5a**

**Enter the following code into the text editor and save the file with .sql format set serveroutput on; declare temp1 number(10); temp2 varchar2(10);**

**begin select sid,sname into temp1,temp2 from student where rank='first'; dbms\_output.put\_line('Student No:'|| temp1 ||' Name:'||temp2||' got first rank'); exception when no\_data\_found then dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'); dbms\_output.put\_line('# Error: there is no student got first rank'); end;**

**/**

**SQL> @5a;**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* # Error: there is no student got first rank PL/SQL procedure successfully completed.**

**SQL> update student set rank='first' where sid=503; 1 row updated.**

**SQL> select \*from student;**

**SID SNAME RANK**



1. **Ravi second**
2. **Raju third**
3. **Ramu first**

**SQL> @5a**

**Student No:503 Name:Ramu got first rank PL/SQL procedure successfully completed. ii)**

**SQL> select \*from student;**

**SID SNAME RANK**

1. **Ravi second**
2. **Raju third**
3. **Ramu first**

**PL/SQL CODE:**

**SQL>ed 5b**

**Enter the following code into the text editor and save the file with .sql format set serverout**

**DECLARE**

**sno student.sid%type; name student.sname%type; srank student.rank%type;**

**BEGIN**

**sno := &sno; name := '&name'; srank := '&srank';**

**INSERT into student values(sno,name,srank); dbms\_output.put\_line('One record inserted**

**COMMIT;**

**-- adding savepoint SAVEPOINT s1;**

**-- second time asking user for input sno := &sno; name := '&name'; srank := '&srank';**

**INSERT into student values(sno,name,srank); dbms\_output.put\_line('One record inserted**

**ROLLBACK TO SAVEPOINT s1;**

**END;**

**/**

**SQL> @5b;**

***EXPERIMENT: 6***

Develop a program that includes the features NESTED IF, CASE and CASE expression.

The programcan be extended using the NULLIF and COALESCE functions. A. NESTED IF:

A nested if-then is an if statement that is the target of another if statement. Nested ifthen statements mean an if statement inside another if statement **Syntax:-** if (condition1) then -- Executes when condition1 is true if

(condition2) then

-- Executes when condition2 is true end if;

end if;

**PL/SQL CODE:** PL/SQL Program to find biggest of three number using nested if.

SQL>ed 6a

Enter the following code into the text editor and save the file with .sql format

|  |
| --- |
| declare a number:=10; b number:=12; c number:=5; begin  dbms\_output.put\_line('a='||a||' b='||b||' c='||c);  if a>b AND a>c then dbms\_output.put\_line('a is greatest');  else if b>a AND b>c then dbms\_output.put\_line('b is greatest');  else dbms\_output.put\_line('c is greatest');  end if;  end if;  end;  / |

SQL> @ "D:\dbms\lab experiments\6a.sql";

**a=10 b=12 c=5 b is greatest**

PL/SQL procedure success

**B. CASE and CASE Expression : CASE statement** selects one sequence of statements to execute. However, to select the sequence, the **CASE** statement uses a selector rather than multiple Boolean expressions. A selector is an expression, the value of which is used to select one of several alternatives. **Syntax**

CASE selector

WHEN 'value1'

THEN S1; WHEN

'value2' THEN S2; WHEN 'value3'

THEN S3;

...

ELSE Sn; -- default

case END CASE;

**SQL> create table emp(eno number(5), ename varchar2(10), loc varchar(10), salary number(10,2));**

Table created.

SQL > insert into emp values(101,'ali','vja',15000);

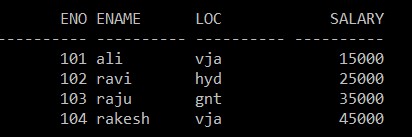
1 row created.

SQL> insert into emp values(102,'ravi','hyd',25000); 1 row created.

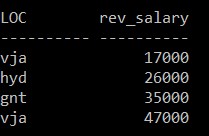
SQL> insert into emp values(103,'raju','gnt',35000); 1 row created.

SQL> insert into emp values(104,'rakesh','vja',45000); 1 row created.

**SQL> select \*from emp;**



**SQL> select loc, case(loc) when 'vja' then salary+2000 when 'hyd' then salary+1000 else salary end "rev\_salary" from emp;**



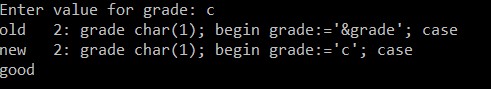
PL/SQL CODE: PL/SQL CODE to demonstrate CASE

SQL> ed 6b

declare grade char(1); begin grade:='&grade'; case when grade='a' then dbms\_output.put\_line('Excellent'); when grade='b' then dbms\_output.put\_line('very good'); when grade='c' then dbms\_output.put\_line('good'); when grade='d' then dbms\_output.put\_line('fair'); when grade='f' then dbms\_output.put\_line('poor'); else dbms\_output.put\_line('No such grade'); end case; end;

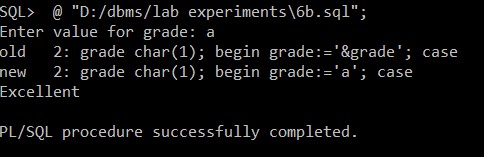
/

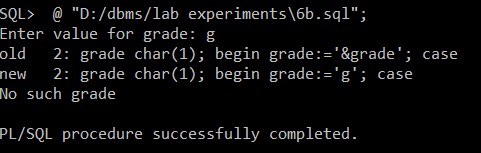
SQL> @ "D:/dbms/lab experiments\6b.sql";



PL/SQL procedure successfully completed.

SQL> @ "D:/dbms/lab experiments\6b.sql";





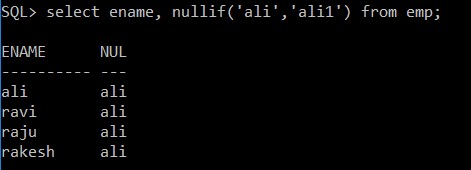
**C. NULLIF:** Takes two arguments. If the two arguments are equal, then NULL is returned.

otherwise the first argument is returned.

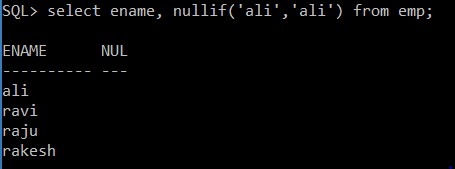
# Syntax: select column\_name, NULLIF(argument1,arguement2) from table\_name;

**Example:**

**SQL> select ename, nullif('ali','ali1') from emp;**



**SQL>select ename, nullif('ali','ali') from emp;**

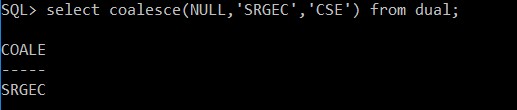


**D. COALESCE:** COALESCE () function accepts a list of arguments and returns the first one that evaluates to a non-null value.

**Syntax: coalesce("expression1","expression2",...);**

**Example:**

**SQL> select coalesce(NULL,'SRGEC','CSE') from dual;**



**EXPERIMENT:7**

**Program development using WHILE LOOPS, numeric FOR LOOPS, nested loops using ERROR**

**Handling, Built–In Exceptions, User defined Exceptions, Raise Application Error**

**A. WHILE LOOP: AWHILE LOOP statementin PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.**

**Syntax:**

**WHILE condition LOOP sequence\_of\_statements**

**END LOOP;**

**PL/SQL Code:APL/SQL Program to find sum of ODD number upto given umber using While loop**

**Set server output on; declare inval number; endval number; s number default 0; begin**

**inval := 1;**

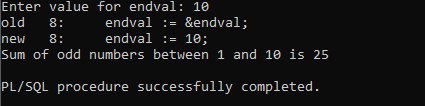
**-- Prompt the user for the value of endval**

**endval := &endval; while inval<endval loop**

**s := s + inval; inval := inval + 2; -- Move to the next odd number end loop;**

**-- Output the result dbms\_output.put\_line('Sum of odd numbers between 1 and ' || endval || ' is ' || s); end;**

**/**



**B. For Loop**

**In PL/SQL, a FOR loop allows you to execute a block of code a specific number of times. It works by iterating through a range of values, which you define**

**Syntax**

**FOR counter IN initial\_value..final\_value LOOP sequence\_of\_statements;**

**ENDLOOP;**

**PL/SQLCODE:APL/SQL code to print multiplication table using for loop**

**SET SERVEROUTPUT ON; DECLARE i NUMBER;**

**num NUMBER := 5; -- You can change the number to get the multiplication table for a different number**

**BEGIN**

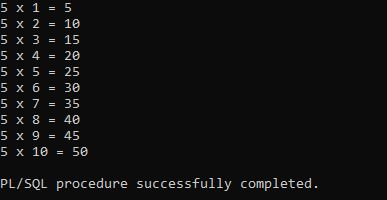
**FOR i IN 1..10 LOOP**

**DBMS\_OUTPUT.PUT\_LINE(num || ' x ' || i || ' = ' || (num \* i));**

**END LOOP;**

**END;**

**/**



**C.NESTED LOOP: PL/SQL allows using one loop inside another loop.It may be either basic,while or for loop.**

**Syntax:**

**WHILE condition1 LOOP sequence\_of\_statements1 WHILE condition2 LOOP sequence\_of\_statements2**

**ENDLOOP;**

**END LOOP;**

**PL/SQLCODE:A PL/SQL program to print n prime number using nested loop using Error Handling.**

**SET SERVEROUTPUT ON;**

**DECLARE**

**i NUMBER := 2; -- Starting from 2 j NUMBER; -- Divisor variable**

**BEGIN**

**-- Loop to find prime numbers LOOP j := 2; -- Start checking divisibility from 2**

**-- Nested loop to check if 'i' is prime**

**LOOP**

**BEGIN**

**-- Exit the loop if 'i' is divisible by 'j' or if 'j' reaches 'i' EXIT WHEN MOD(i, j) = 0 OR j = i; j := j + 1;**

**EXCEPTION**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error while checking number ' || i || ': ' || SQLERRM);**

**EXIT; -- Exit the inner loop in case of an error**

**END;**

**END LOOP;**

**-- If 'i' is prime (j == i), print it**

**IF j = i THEN**

**DBMS\_OUTPUT.PUT\_LINE(i || ' is prime');**

**END IF;**

**-- Increment 'i' to check the next number**

**i := i + 1;**

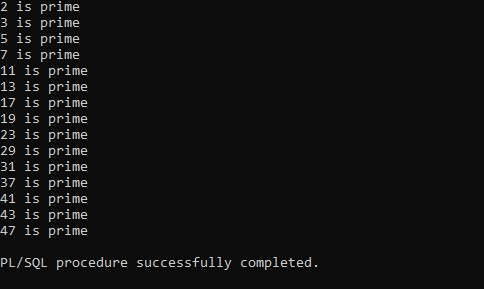
**-- Exit the outer loop when 'i' reaches 50**

**EXIT WHEN i = 50;**

**END LOOP;**

**END;**

**/**



**D.Built–In Exceptions::In PL/SQL, built-in exceptions are predefined exceptions that represent common error conditions.**

**DECLARE result NUMBER; BEGIN**

**result := 10 / 0; -- Division by zero**

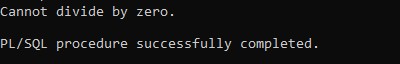
**EXCEPTION**

**WHEN ZERO\_DIVIDE THEN**

**DBMS\_OUTPUT.PUT\_LINE('Cannot divide by zero.');**

**END;**

**/**



**E:USER defined Exceptions:In PL/SQL, user-defined exceptions are exceptions that you can define explicitly in your code.**

**DECLARE**

**-- Declaring the user-defined exception insufficient\_balance EXCEPTION; balance NUMBER := 50; withdrawal\_amount NUMBER := 100;**

**BEGIN**

**-- Check if balance is sufficient for withdrawal**

**IF balance <withdrawal\_amount THEN**

**RAISE insufficient\_balance; -- Raise exception if balance is insufficient END IF;**

**-- If no exception, proceed with the withdrawal balance := balance - withdrawal\_amount;**

**DBMS\_OUTPUT.PUT\_LINE('Withdrawal successful! New balance: ' || balance);**

**EXCEPTION**

**WHEN insufficient\_balance THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance for withdrawal.');**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);**

**END;**

**/**



**F::Raise Application Error::The RAISE\_APPLICATION\_ERROR procedure in PL/SQL is used to raise user-defined exceptions with a custom error number and message.**

**DECLARE v\_balance NUMBER := 50; v\_purchase\_amount NUMBER := 100;**

**BEGIN**

**-- Check if the balance is sufficient for the purchase**

**IF v\_balance<v\_purchase\_amount THEN**

**-- Raise a custom error if the balance is insufficient**

**RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for purchase.'); END IF;**

**-- Continue with the purchase if balance is sufficient**

**v\_balance := v\_balance - v\_purchase\_amount;**

**DBMS\_OUTPUT.PUT\_LINE('Purchase successful! Remaining balance: ' || v\_balance);**

**EXCEPTION**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);**

**END;**

**/**



**8. Programs development using creation of procedures, passing parameters IN and OUT of Procedures.**

**A Stored procedure is a set of SQL statements stored in the database and executed as a single unit. It helps in improving performance, reducing redundancy, and ensuring data security.**

**Step 1: Create the Table**

**CREATE TABLE employees ( id NUMBER PRIMARY KEY, name VARCHAR2(100), salary NUMBER**

**);**

**INSERT INTO employees VALUES (101, 'Alice', 50000);**

**INSERT INTO employees VALUES (102, 'Bob', 60000);**

**INSERT INTO employees VALUES (103, 'Charlie', 55000);**

**COMMIT;**

**Step 2: Create a Procedure with IN and OUT Parameters**

**This procedure takes an employee ID (IN parameter) and returns the employee's name (OUT parameter).**

**CREATE OR REPLACE PROCEDURE get\_employee\_name ( emp\_id IN NUMBER, emp\_name OUT VARCHAR2**

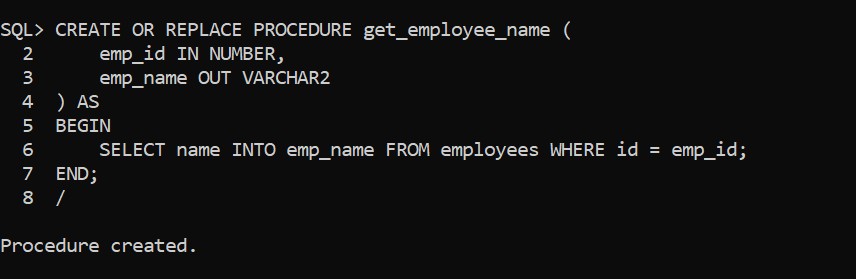
**) AS**

**BEGIN**

**SELECT name INTO emp\_name FROM employees WHERE id = emp\_id;**

**END;**

**/**



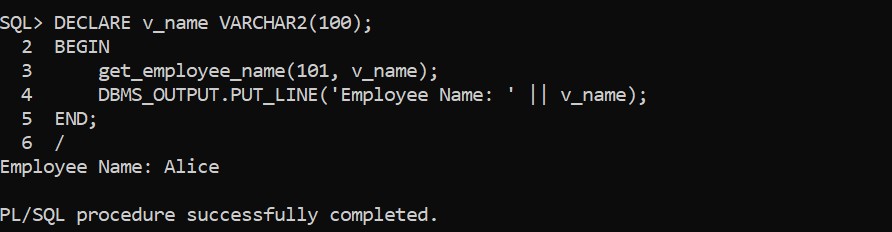
**Step 3: Execute the Procedure**

**DECLARE v\_name VARCHAR2(100); BEGIN get\_employee\_name(101, v\_name);**

**DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || v\_name);**

**END;**

**/**



**Step 4: Create a Procedure with an INOUT Parameter**

**This procedure:**

* **Takes an employee ID (IN parameter).**
* **Takes an existing salary (INOUT parameter).**
* **Increases the salary by 10% and updates the table.**

**CREATE OR REPLACE PROCEDURE update\_salary ( emp\_id IN NUMBER, emp\_salary IN OUT NUMBER**

**) AS**

**BEGIN**

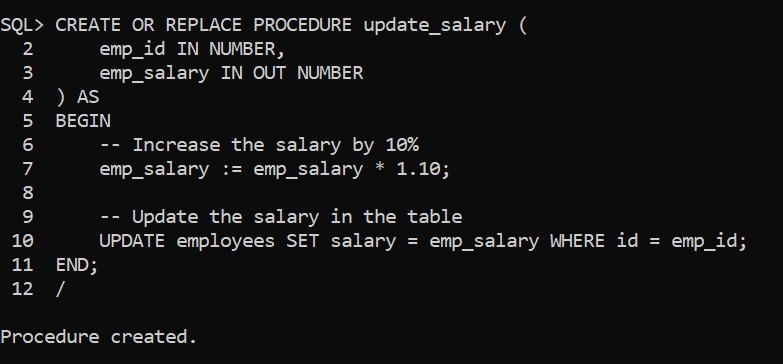
**-- Increase the salary by 10% emp\_salary := emp\_salary \* 1.10;**

**-- Update the salary in the table**

**UPDATE employees SET salary = emp\_salary WHERE id = emp\_id;**

**END;**

**/**



**Step 5: Execute the Procedure**

**DECLARE v\_salary NUMBER;**

**BEGIN**

**-- Get the initial salary of employee 101**

**SELECT salary INTO v\_salary FROM employees WHERE id = 101;**

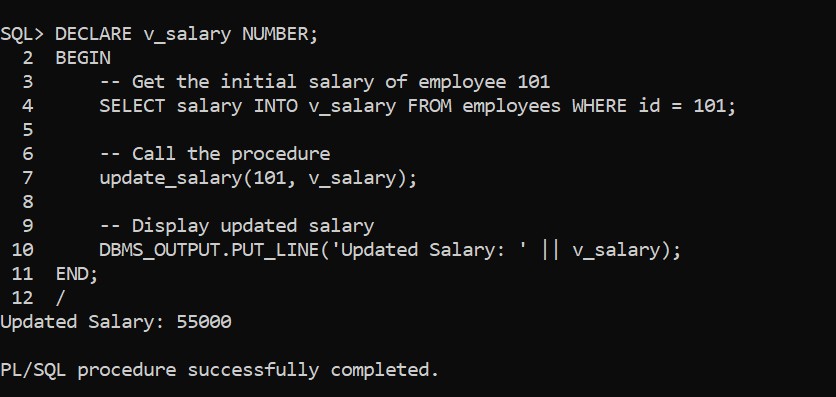
**-- Call the procedure update\_salary(101, v\_salary);**

**-- Display updated salary**

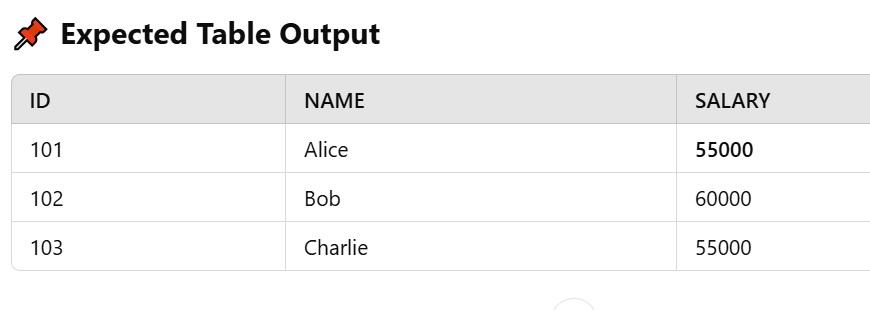
**DBMS\_OUTPUT.PUT\_LINE('Updated Salary: ' || v\_salary);**

**END;**

**/**



**Step 6: Verify Updated Table Data**



**Benefits of Stored Procedures**

**Improves Performance – Reduces SQL execution time.**

**Encapsulation – Business logic is stored in one place.**

**Security – Users can execute procedures without direct table access.**

**Reusability – The same procedure can be executed multiple times.**

**EXPERIMENT – 11**

**AIM : Develop programs using before and after triggers, row and statement triggers and instead of triggers.**

**Trigger : A trigger in SQL is a special type of stored procedure that is automatically executed or "triggered" when certain events occur on a specific table or view in a database. Triggers are typically used to enforce business rules, perform validation, or automate certain actions like logging or updating related data.**

**· BEFORE Trigger: Executes before the operation (e.g., before inserting or updating data).**

**· AFTER Trigger: Executes after the operation (e.g., after inserting, updating, or deleting data).**

**· ROW Trigger: Executes once for each row affected.**

**· STATEMENT Trigger: Executes once for the entire SQL statement.**

**· INSTEAD OF Trigger: Replaces the operation (e.g., in views to insert, update, or delete underlying data).**

**These triggers allow you to automate various actions such as validation, logging, and even preventing certain operations. create table customers and insert values into it.**

**CREATE table CUSTOMERS(**

**ID number(3),**

**Name varchar2(10),**

**Age number(3),**

**Address varchar2(10),**

**Salary number(10,2)**

**);**

**SQL> insert into customers values(1,'ramesh',32,'ahmedabad',2000);**

**1 row created.**

**SQL> insert into customers values(2,'khilan',25,'Delhi',1500);**

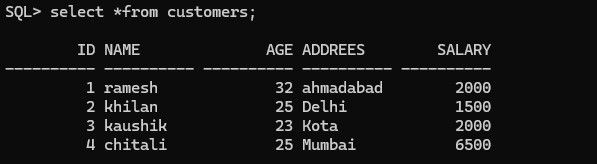
**1 row created.**

**SQL> insert into customers values(3,'kaushik',23,'Kota',2000); 1 row created.**

**SQL> insert into customers values(4,'chitali',25,'Mumbai',6500);**

**1 row created.**

**SQL> select \*from customers;**



**PL/SQL Code : for creation of trigger while insert / update records into a table.**

**SQL> ed 11a**

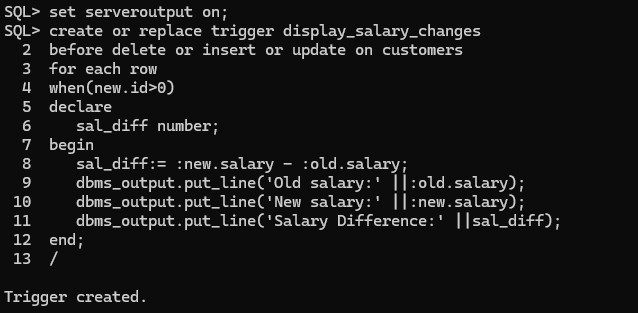
**CREATE OR REPLACE TRIGGER display\_salary\_changes BEFORE DELETE OR INSERT OR UPDATE**

**ON customers FOR EACH ROW WHEN (NEW.ID > 0) DECLARE sal\_diff number; BEGIN**

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

**END;**

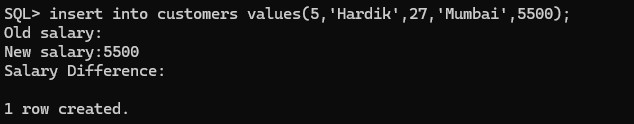
**/**



**Sql> @11a**

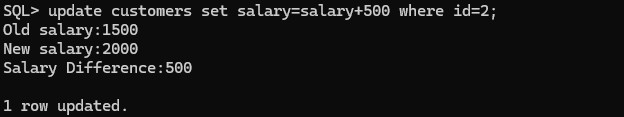
**Trigger created**

1. **Query to insert a new customer detail into the customer table. insert into customers values(5,'Hardik',27,'Mumbai',5500);**



1. **Query to update customers salary as salary+500 for customer id=2**

**SQL> update customers set salary=salary+500 where id=2;**

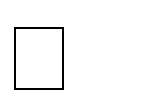


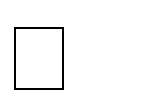
**EXPERIMENT-12**

**Create a table and perform the search operation on table using indexing and non-indexing techniques**

**An index is a database object that improves the speed of data retrieval operations on a table.**

**The DBMS can use indexes to find data more quickly.**

 **Without an index: The query might take more time, especially with a large dataset.**

 **With an index: The database can quickly locate the relevant rows based on the index, significantly reducing the search time.**

**CREATE TABLE teachers ( teacher\_id INT PRIMARY KEY, first\_name VARCHAR(50) NOT NULL, last\_name VARCHAR(50) NOT NULL, subject VARCHAR(100)**

**);**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (1, 'John', 'Doe', 'Mathematics');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (2, 'Jane', 'Smith', 'English');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (3, 'Michael', 'Johnson', 'Science');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (4, 'Emily', 'Davis', 'History');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (5, 'Daniel', 'Martinez', 'Physical Education');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (6, 'Sophia', 'Brown', 'Biology');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (7, 'David', 'Wilson', 'Chemistry');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (8, 'Olivia', 'Taylor', 'Art');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject)**

**VALUES (9, 'James', 'White', 'Geography');**

**INSERT INTO teachers (teacher\_id, first\_name, last\_name, subject) VALUES (10, 'Lily', 'Harris', 'Music');**

**Set timing on;**

**Select \* from teachers;**



**create index teacher\_subject\_ind on Teachers(subject);**

