DMBS FILE

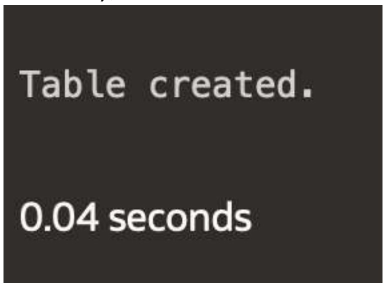
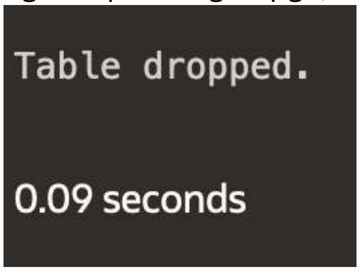
By – Tushya Gupta
Roll number – UE233106

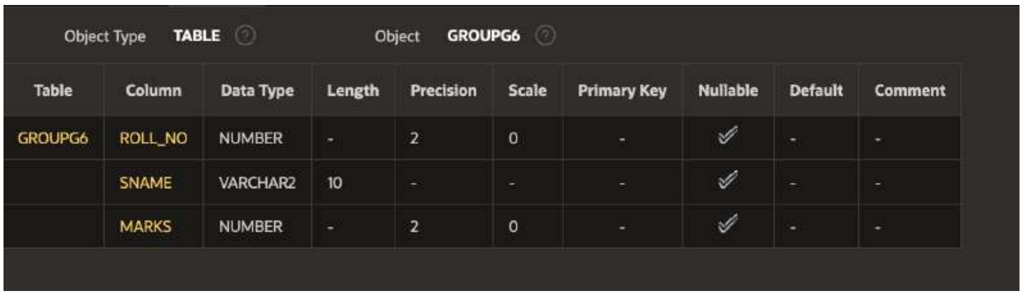
SQL

Structured Query Language is a domain-specific language used to manage data, especially in a relational database management system. It is particularly useful in handling structured data, i.e., data incorporating relations among entities and variables.

Different SQL Commands

SQL commands are instructions or queries sent to the database to interact with it in a certain way.
They are majorly of 3 types –

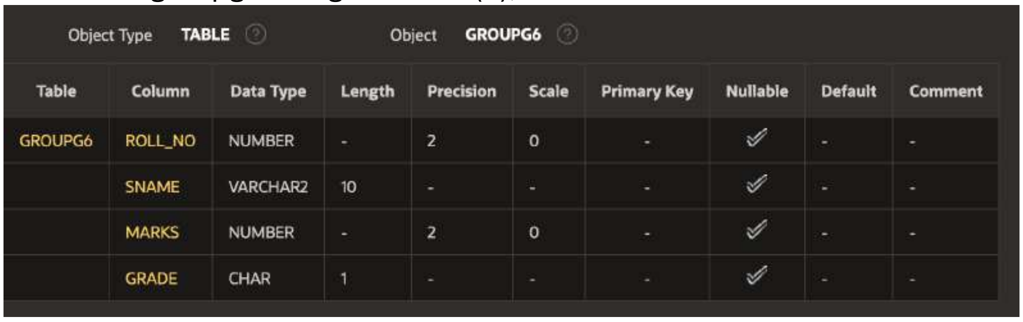
1. Data Definition Language – The following commands fall under DDL-
   a. CREATE – used to create a table for data
   Eg - create table groupg6(
   ```
           roll_no number(2),
           sname varchar(10),
           marks number(2)
       );
   ```
   
   ```
   Table created.

   0.04 seconds
   ```
   b. DROP – Used to delete an existing table from the database
   Eg - drop table groupg6;
   
   ```
   Table dropped.

   0.09 seconds
   ```
   c. ALTER – Used to modify/alter the structure of a table i.e. change the columns of a table
   Eg -
   Table before changes



Object Type TABLE Object GROUPG6

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG6	ROLL_NO	NUMBER	-	2	0	-	✓	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-

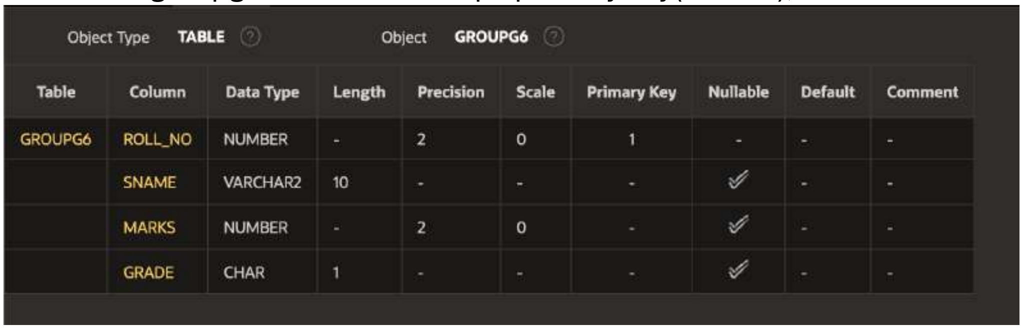
alter table groupg6 add grade char(1);



Object Type TABLE Object GROUPG6

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG6	ROLL_NO	NUMBER	-	2	0	-	✓	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-
	GRADE	CHAR	1	-	-	-	✓	-	-

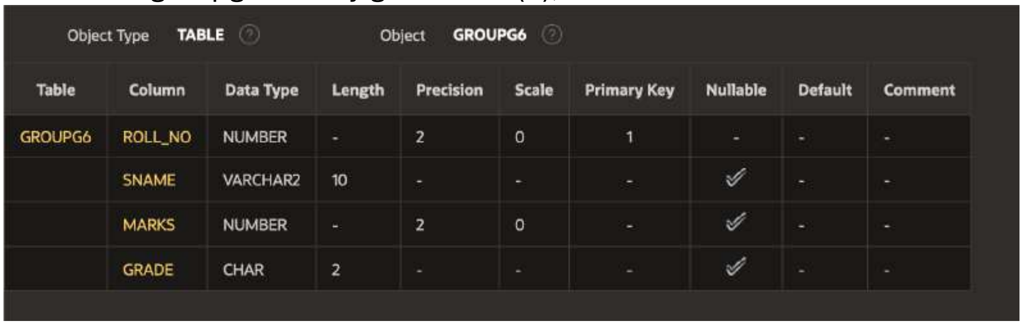
alter table groupg6 add constraint pk primary key(roll_no);



Object Type TABLE Object GROUPG6

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG6	ROLL_NO	NUMBER	-	2	0	1	-	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-
	GRADE	CHAR	1	-	-	-	✓	-	-

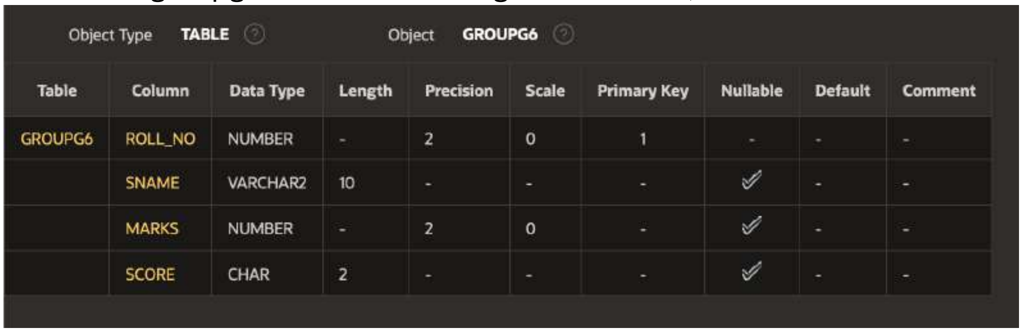
alter table groupg6 modify grade char(2);



Object Type TABLE Object GROUPG6

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG6	ROLL_NO	NUMBER	-	2	0	1	-	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-
	GRADE	CHAR	2	-	-	-	✓	-	-

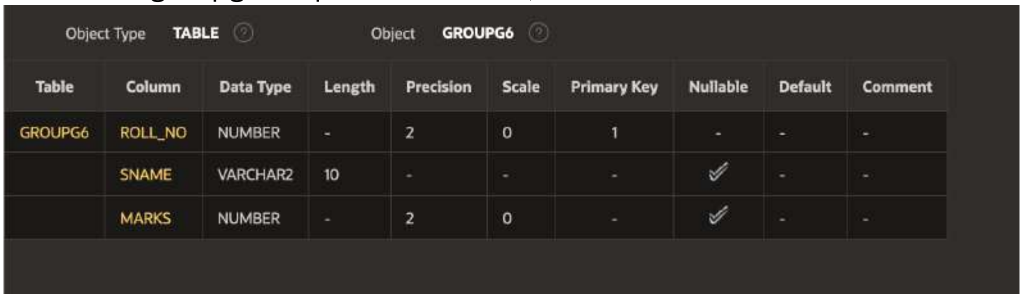
alter table groupg6 rename column grade to score;



Object Type TABLE Object GROUPG6

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG6	ROLL_NO	NUMBER	-	2	0	1	-	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-
	SCORE	CHAR	2	-	-	-	✓	-	-

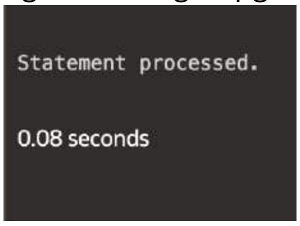
alter table groupg6 drop column score;



Object Type **TABLE** Object **GROUPG6**

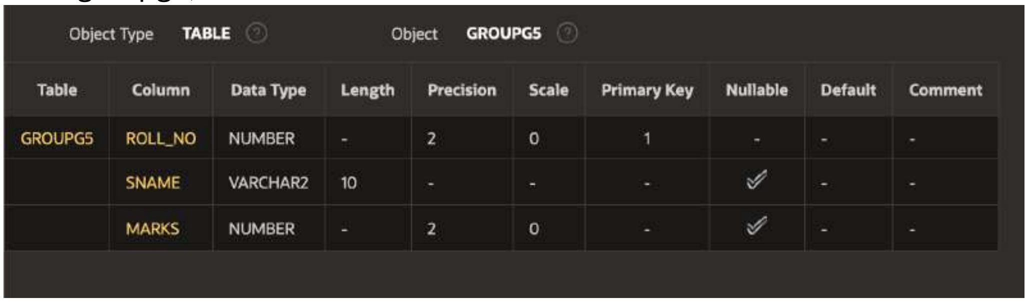
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG6	ROLL_NO	NUMBER	-	2	0	1	-	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-

d. RENAME – Change the name of an existing table
Eg - rename groupg6 to groupg5;



```
Statement processed.

0.08 seconds
```

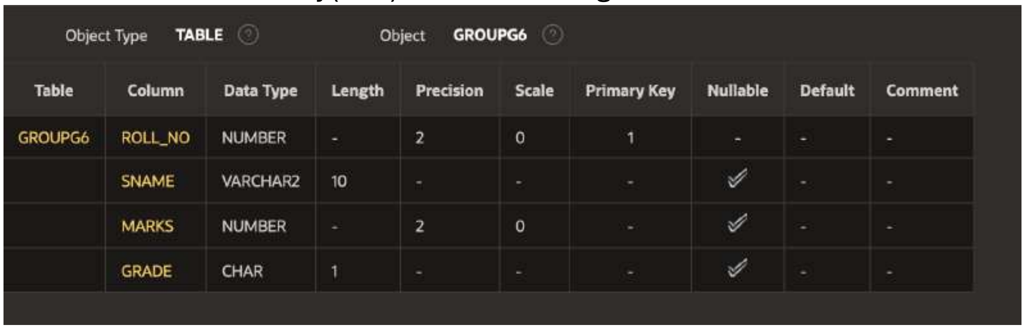
desc groupg5;



Object Type **TABLE** Object **GROUPG5**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG5	ROLL_NO	NUMBER	-	2	0	1	-	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-

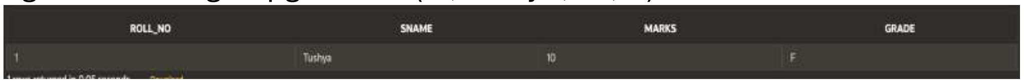
2. Data Manipulation Language –
   a. INSERT – Insert an entry(row) into an existing table



Object Type **TABLE** Object **GROUPG6**

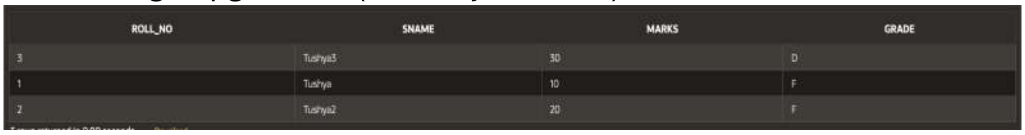
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG6	ROLL_NO	NUMBER	-	2	0	1	-	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-
	GRADE	CHAR	1	-	-	-	✓	-	-

Eg - insert into groupg6 values('1','Tushya','10','F')



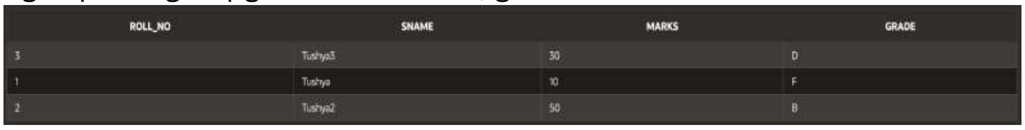
ROLL_NO	SNAME	MARKS	GRADE
1	Tushya	10	F

insert into groupg6 values('2','Tushya2','20','F')
insert into groupg6 values('3','Tushya3','30','D')



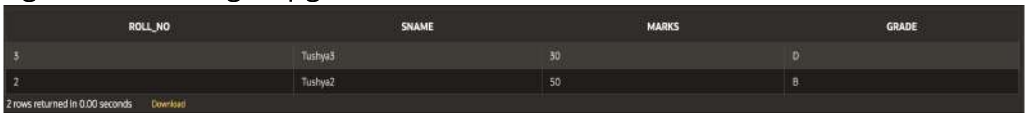
ROLL_NO	SNAME	MARKS	GRADE
3	Tushya3	30	D
1	Tushya	10	F
2	Tushya2	20	F

b. UPDATE – Change a value in specified entries(rows) from an existing table
Eg - update groupg6 set marks='50', grade='B' where roll_no='2'



ROLL_NO	SNAME	MARKS	GRADE
3	Tushya3	30	D
1	Tushya	10	F
2	Tushya2	50	B

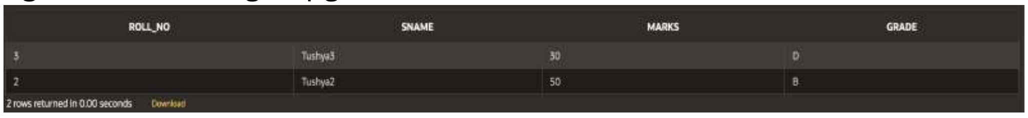
c. DELETE – Delete a specified entry from an existing table
Eg - delete from groupg6 where roll_no='1'



ROLL_NO	SNAME	MARKS	GRADE
3	Tushya3	30	D
2	Tushya2	50	B

2 rows returned in 0.00 seconds Download

3. Data Query Language –
   a. SELECT – Print/Display a specified range of entries from an existing table
   Eg - select * from groupg6;



ROLL_NO	SNAME	MARKS	GRADE
3	Tushya3	30	D
2	Tushya2	50	B

2 rows returned in 0.00 seconds Download

Constraints

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

Types of constraints –

1. UNIQUE – Ensures that each entry in this column is unique

   Eg - create table second1(
   ```
           rollcall numeric(3) unique,
           sname varchar(30) not null,
           sports varchar(20),
           grade varchar(2)
       );
   ```

2. NOT NULL – Makes it so that there must be an entry into this column of the table and cannot be left blank.

   Eg - create table second1(
   ```
           rollcall numeric(3) unique,
           sname varchar(30) not null,
           sports varchar(20),
           grade varchar(2)
       );
   ```

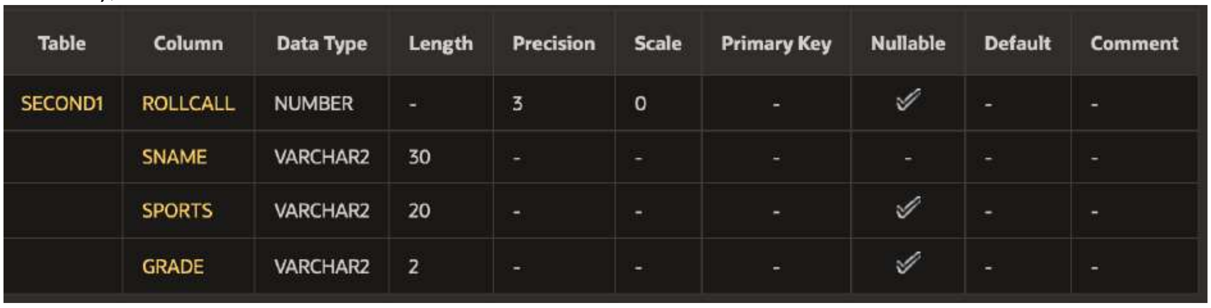


Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
SECOND1	ROLLCALL	NUMBER	-	3	0	-	✓	-	-
	SNAME	VARCHAR2	30	-	-	-	-	-	-
	SPORTS	VARCHAR2	20	-	-	-	✓	-	-
	GRADE	VARCHAR2	2	-	-	-	✓	-	-

3. PRIMARY KEY – An identifying key to distinguish each entry into a table. It is NOT NULL and UNIQUE by default

   Eg - alter table second1 add aid numeric(3);

   alter table second1 add constraint pkey primary key(aid);

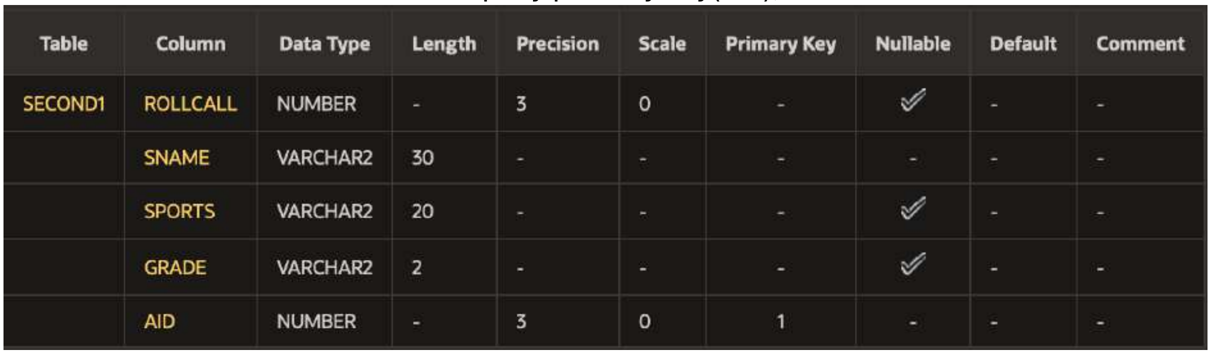


Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
SECOND1	ROLLCALL	NUMBER	-	3	0	-	✓	-	-
	SNAME	VARCHAR2	30	-	-	-	-	-	-
	SPORTS	VARCHAR2	20	-	-	-	✓	-	-
	GRADE	VARCHAR2	2	-	-	-	✓	-	-
	AID	NUMBER	-	3	0	1	-	-	-

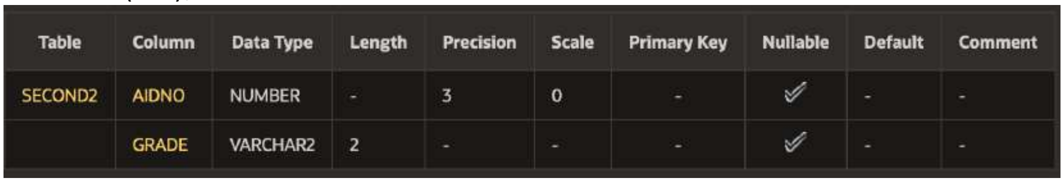
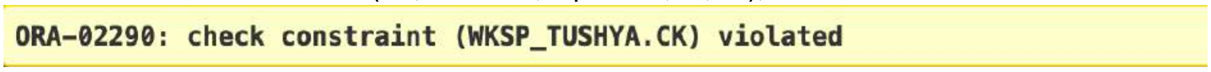
4. FOREIGN KEY – It points to a column in another table and that column must be UNIQUE
   Eg - create table second2(
   ```
           aidno numeric(3) unique,
           grade varchar(2)
       );
       alter table second2 add constraint fkey foreign key (aidno) references
       second1(aid);
   ```
   

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
SECOND2	AIDNO	NUMBER	-	3	0	-	✓	-	-
	GRADE	VARCHAR2	2	-	-	-	✓	-	-

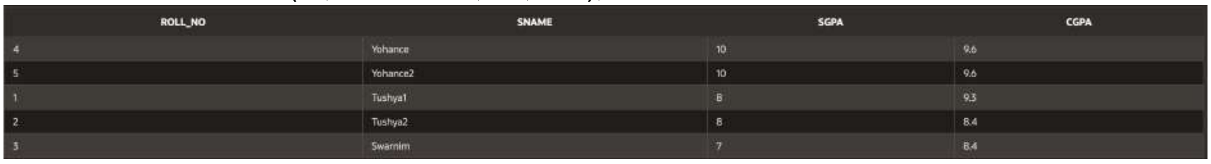
5. CHECK – Applies a logical condition to a column
   Eg – alter table second1 add constraint ck check(rollcall > 99);
   insert into second1 values (99, 'name1', 'sports1', 'A',20);
   

   ```
   ORA-02290: check constraint (WKSP_TUSHYA.CK) violated
   ```

To implement the use of clauses, aggregate functions, set operations and pattern matching in SQL

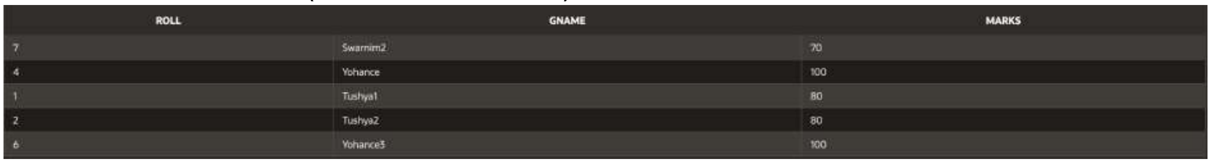
Tables used –

```
create table stu(
   roll_no int primary key,
   sname varchar(10) not null,
   sgpa float,
   cgpa float
);

insert into stu values('1','Tushya1','8','9.3');
insert into stu values('2','Tushya2','8','8.4');
insert into stu values('3','Swarnim','7','8.4');
insert into stu values('4','Yohance','10','9.6');
insert into stu values('5','Yohance2','10','9.6');
```



ROLL_NO	SNAME	SGPA	CGPA
4	Yohance	10	9.6
5	Yohance2	10	9.6
1	Tushya1	8	9.3
2	Tushya2	8	8.4
3	Swarnim	7	8.4

```
create table stu2 (
   roll int primary key,
   gname varchar(10) not null,
   marks int
);
insert into stu2 values('1','Tushya1','80');
insert into stu2 values('2','Tushya2','80');
insert into stu2 values('7','Swarnim2','70');
insert into stu2 values('4','Yohance','100');
insert into stu2 values('6','Yohance3','100');
```

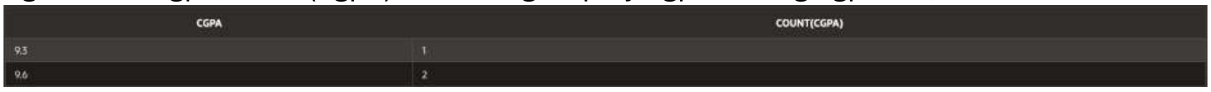


ROLL	GNAME	MARKS
7	Swarnim2	70
4	Yohance	100
1	Tushya1	80
2	Tushya2	80
6	Yohance3	100

Clauses –

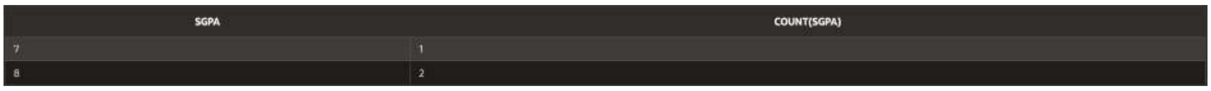
1. Group by –
2. Having –

   Eg - select cgpa,count(cgpa) from stu group by cgpa having cgpa > 9;



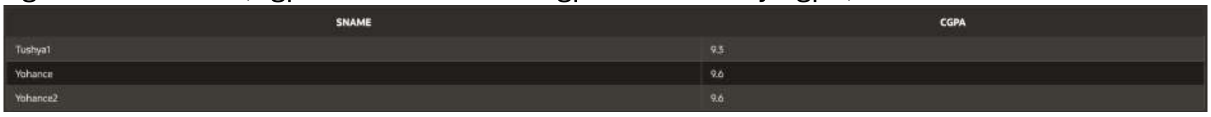
CGPA	COUNT(CGPA)
9.3	1
9.6	2

   select sgpa,count(sgpa) from stu group by sgpa having sgpa < 9;



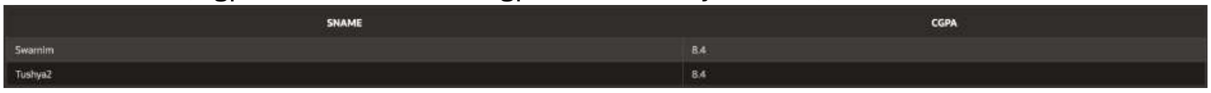
SGPA	COUNT(SGPA)
7	1
8	2

3. Order By –
4. Where –
   Eg - select sname,cgpa from stu where cgpa > 9 order by cgpa;



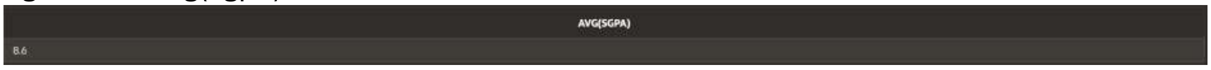
SNAME	CGPA
Tushya1	9.3
Yohance	9.6
Yohance2	9.6

   select sname,cgpa from stu where cgpa < 9 order by sname;



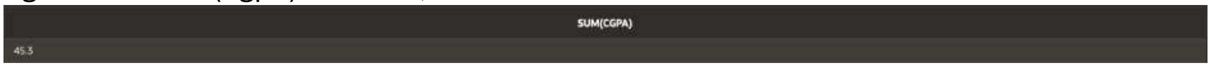
SNAME	CGPA
Swarnim	8.4
Tushya2	8.4

Aggregate functions –
1. Avg –
   Eg - select avg(sgpa) from stu;



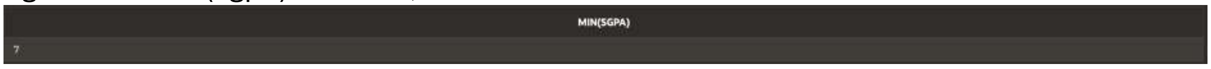
AVG(SGPA)
8.6

2. Sum –
   Eg - select sum(cgpa) from stu;



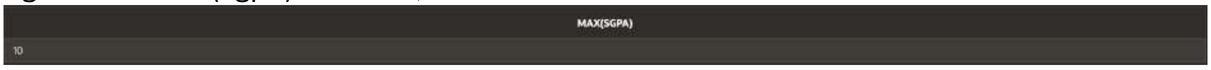
SUM(CGPA)
45.3

3. Min –
   Eg - select min(sgpa) from stu;



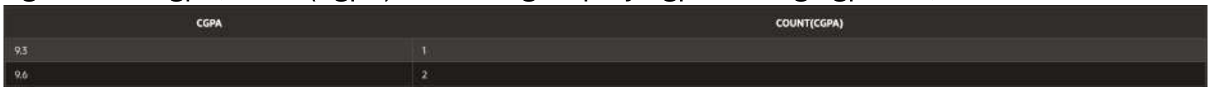
MIN(SGPA)
7

4. Max –
   Eg - select max(sgpa) from stu;



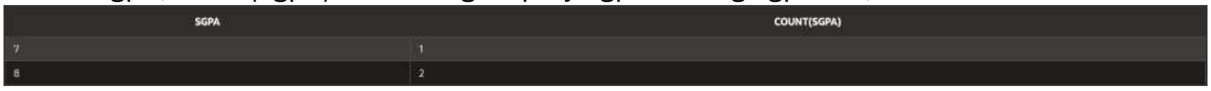
MAX(SGPA)
10

5. Count –
   Eg - select cgpa,count(cgpa) from stu group by cgpa having cgpa > 9;



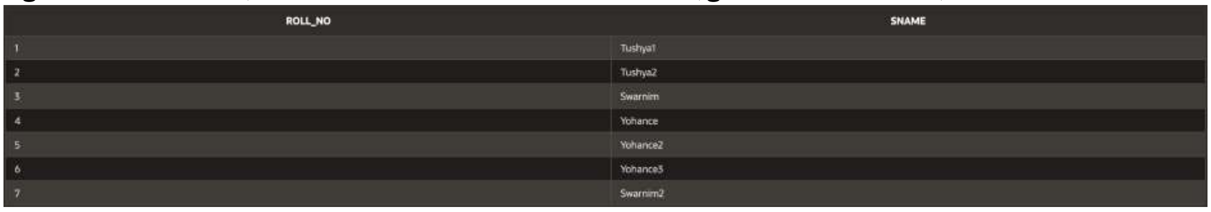
CGPA	COUNT(CGPA)
9.3	1
9.6	2

   select sgpa,count(sgpa) from stu group by sgpa having sgpa < 9;



SGPA	COUNT(SGPA)
7	1
8	2

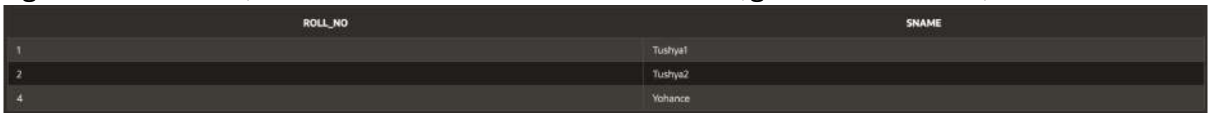
Set Operations –
1. Union –
   Eg - select roll_no,sname from stu union select roll,gname from stu2;



ROLL_NO	SNAME
1	Tushya1
2	Tushya2
3	Swarnim
4	Yohance
5	Yohance2
6	Yohance3
7	Swarnim2

2. Intersect –

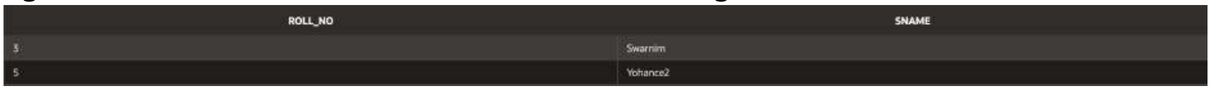
Eg - select roll_no,sname from stu intersect select roll,gname from stu2;



ROLL_NO	SNAME
1	Tushya1
2	Tushya2
4	Yohance

3. Minus –

   Eg - select roll_no,sname from stu minus select roll,gname from stu2;

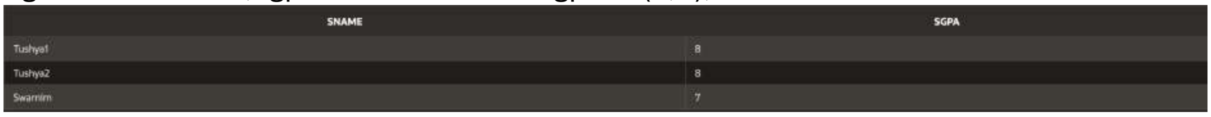


ROLL_NO	SNAME
3	Swarnim
5	Yohance2

Pattern matching –

1. In –

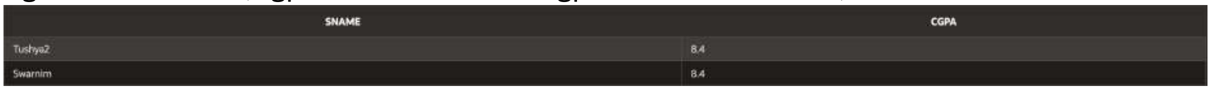
   Eg - select sname,sgpa from stu where sgpa in (7,8);



SNAME	SGPA
Tushya1	8
Tushya2	8
Swarnim	7

2. Between –

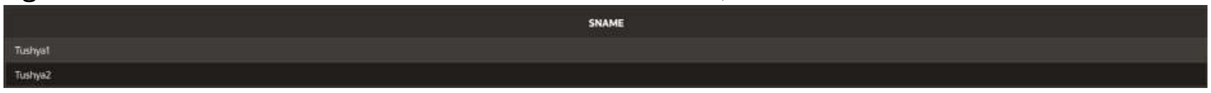
   Eg - select sname,cgpa from stu where cgpa between 8 and 9;



SNAME	CGPA
Tushya2	8.4
Swarnim	8.4

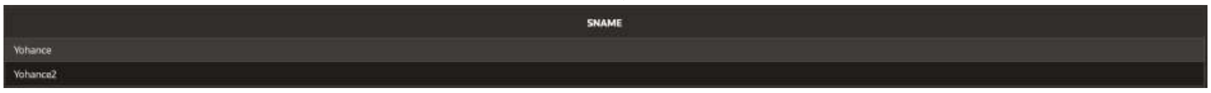
3. Like –

   Eg - select sname from stu where sname like '%sh%';



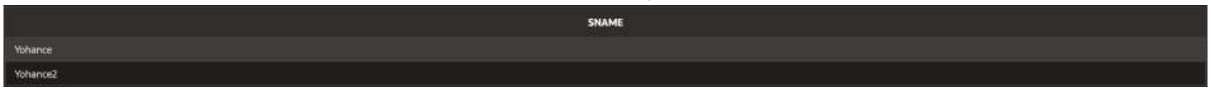
SNAME
Tushya1
Tushya2

select sname from stu where sname like 'Y%';



SNAME
Yohance
Yohance2

select sname from stu where sname like '%e%';

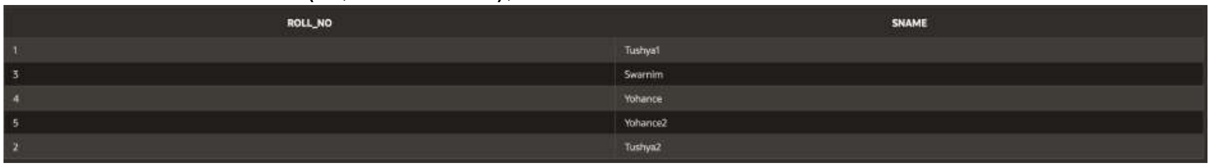


SNAME
Yohance
Yohance2

To implement the use of joins, nested queries and sequence.

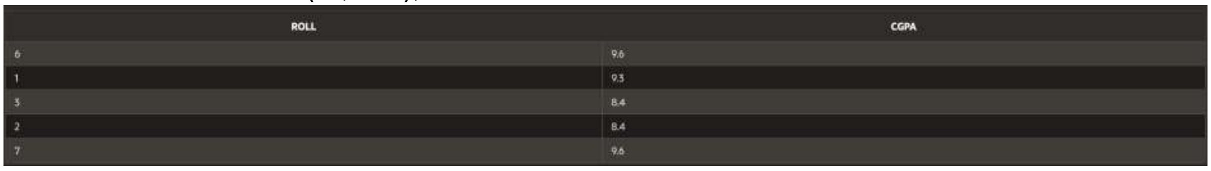
Tables Used –

```
create table stu1(
   roll_no int primary key,
   sname varchar(10) not null
);

insert into stu1 values('1','Tushya1');
insert into stu1 values('3','Swarnim');
insert into stu1 values('4','Yohance');
insert into stu1 values('2','Tushya2');
insert into stu1 values('5','Yohance2');
```



ROLL_NO	SNAME
1	Tushya1
3	Swarnim
4	Yohance
5	Yohance2
2	Tushya2

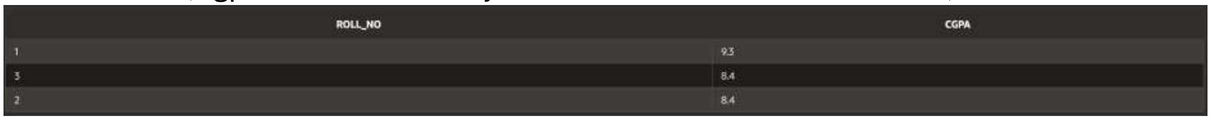
```
create table stu2(
   roll int primary key,
   cgpa float
);

insert into stu2 values('1','9.3');
insert into stu2 values('2','8.4');
insert into stu2 values('3','8.4');
insert into stu2 values('6','9.6');
insert into stu2 values('7','9.6');
```



ROLL	CGPA
6	9.6
1	9.3
3	8.4
2	8.4
7	9.6

Inner Join – Only common rows taken

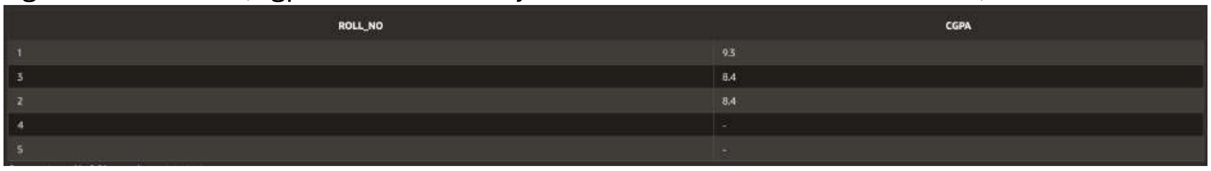
```
select roll_no,cgpa from stu1 inner join stu2 on stu1.roll_no=stu2.roll;
```



ROLL_NO	CGPA
1	9.3
3	8.4
2	8.4

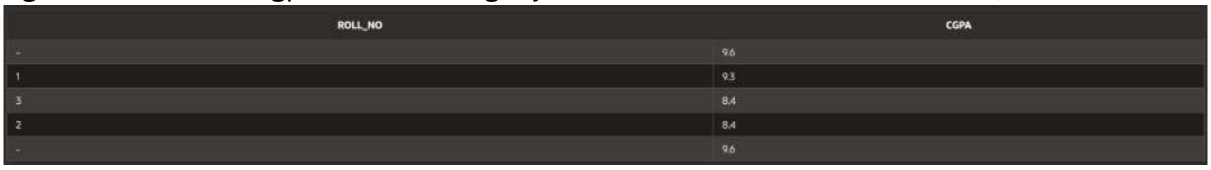
Outer Join –

1. Left join – All rows of left table taken
   Eg - select roll_no,cgpa from stu1 left join stu2 on stu1.roll_no=stu2.roll;



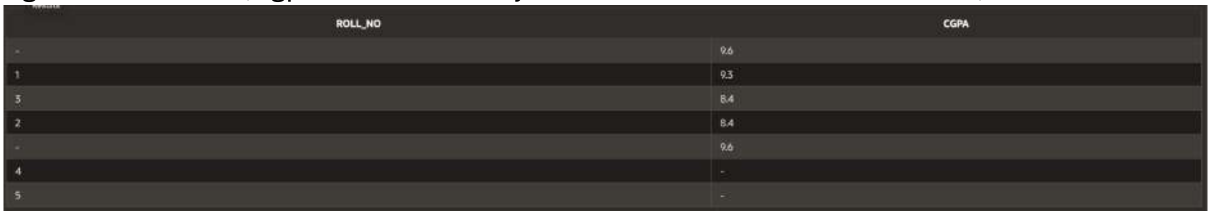
ROLL_NO	CGPA
1	9.3
3	8.4
2	8.4
4	-
5	-

2. Right Join – All rows of right table taken
   Eg - select roll_no,cgpa from stu1 right join stu2 on stu1.roll_no=stu2.roll;



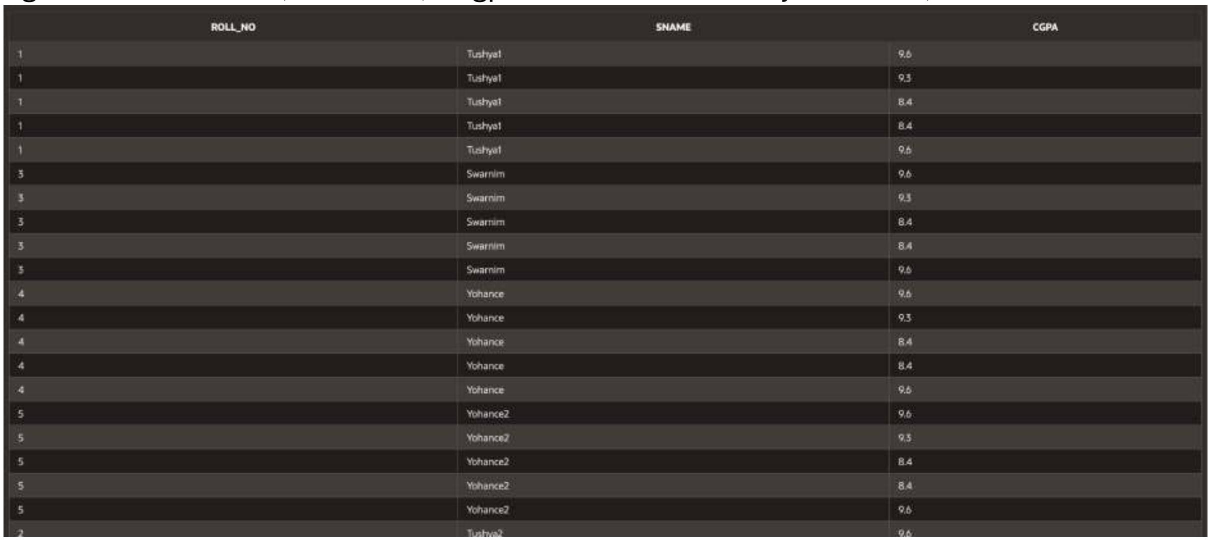
ROLL_NO	CGPA
-	9.6
1	9.3
3	8.4
2	8.4
-	9.6

3. Full Join – All rows from both tables
   Eg - select roll_no,cgpa from stu1 full join stu2 on stu1.roll_no=stu2.roll;



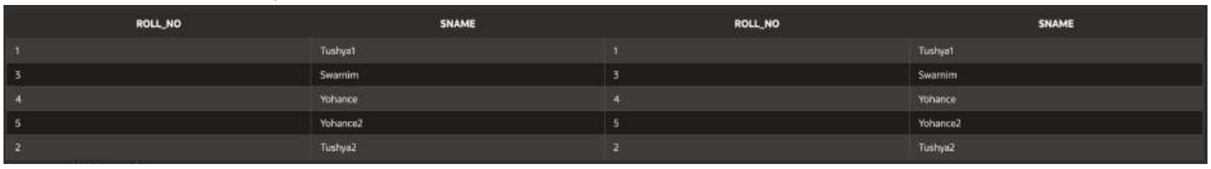
ROLL_NO	CGPA
-	9.6
1	9.3
3	8.4
2	8.4
-	9.6
4	-
5	-

Cross Join –
Eg - select m.roll_no,m.sname,n.cgpa from stu1 m cross join stu2 n;



ROLL_NO	SNAME	CGPA
1	Tushya1	9.6
1	Tushya1	9.3
1	Tushya1	8.4
1	Tushya1	8.4
1	Tushya1	9.6
3	Swarnim	9.6
3	Swarnim	9.3
3	Swarnim	8.4
3	Swarnim	8.4
3	Swarnim	9.6
4	Yohance	9.6
4	Yohance	9.3
4	Yohance	8.4
4	Yohance	8.4
4	Yohance	9.6
5	Yohance2	9.6
5	Yohance2	9.3
5	Yohance2	8.4
5	Yohance2	8.4
5	Yohance2	9.6
2	Tushya2	9.6

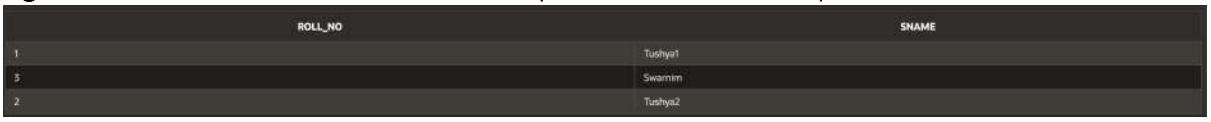
Self Join –
Eg - select m.roll_no,m.sname,n.roll_no,n.sname from stu1 m inner join stu1 n on m.roll_no=n.roll_no;



ROLL_NO	SNAME	ROLL_NO	SNAME
1	Tushya1	1	Tushya1
3	Swarnim	3	Swarnim
4	Yohance	4	Yohance
5	Yohance2	5	Yohance2
2	Tushya2	2	Tushya2

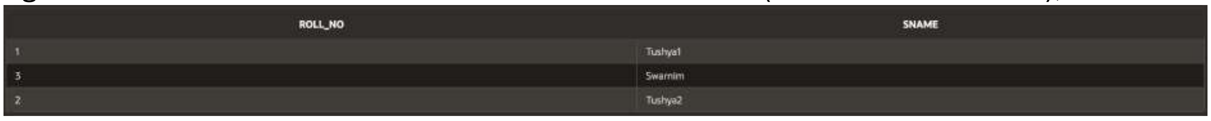
Nested Queries –

- Select –
  Eg - select * from stu1 where roll_no in (select roll from stu2);



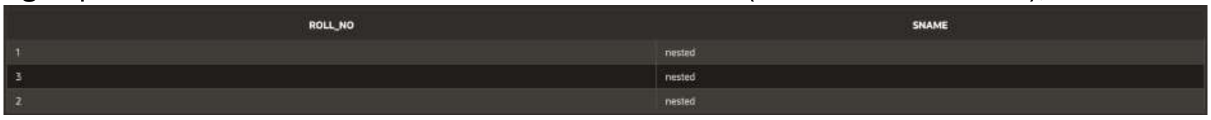
ROLL_NO	SNAME
1	Tushya1
3	Swamim
2	Tushya2

- Insert –
  Eg - insert into stu1_ select * from stu1 where roll_no in (select roll from stu2);



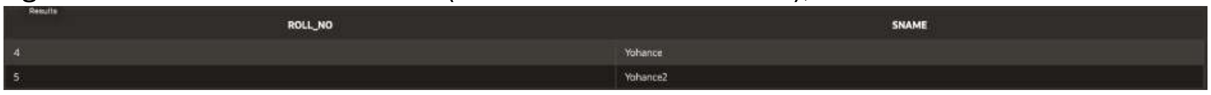
ROLL_NO	SNAME
1	Tushya1
3	Swarnim
2	Tushya2

- Update –
  Eg - update stu1_ set sname='nested' where roll_no in (select roll from stu2);



ROLL_NO	SNAME
1	nested
3	nested
2	nested

- Delete –
  Eg - delete stu1 where roll_no in (select roll_no from stu1_);

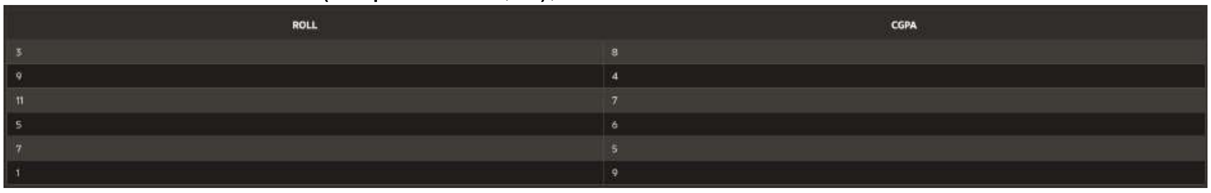


Results

ROLL_NO	SNAME
4	Yohance
5	Yohance2

Sequence –
create sequence seq1 start with 1 increment by 2 maxvalue 15;

insert into stu2_ values(seq1.nextval,'9');
insert into stu2_ values(seq1.nextval,'8');
insert into stu2_ values(seq1.nextval,'6');
insert into stu2_ values(seq1.nextval,'5');
insert into stu2_ values(seq1.nextval,'4');
insert into stu2_ values(seq1.nextval,'7');



ROLL	CGPA
3	8
9	4
11	7
5	6
7	5
1	9

alter sequence seq1 maxvalue 21;

To implement views of indexs and views

Table used –

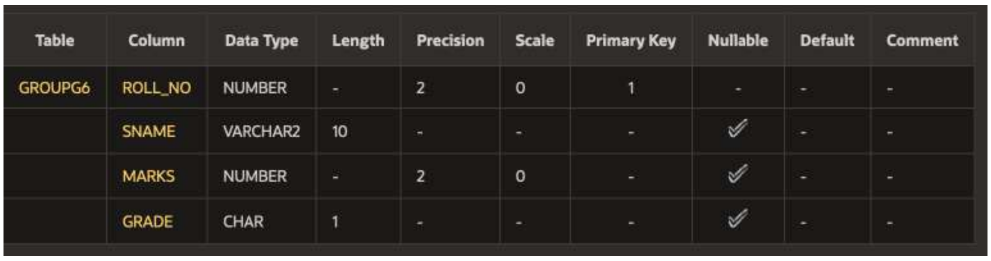
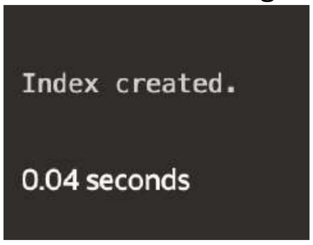


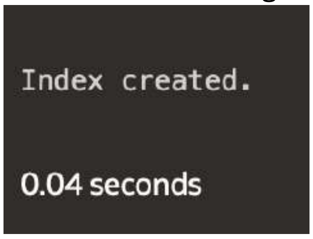
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
GROUPG6	ROLL_NO	NUMBER	-	2	0	1	-	-	-
	SNAME	VARCHAR2	10	-	-	-	✓	-	-
	MARKS	NUMBER	-	2	0	-	✓	-	-
	GRADE	CHAR	1	-	-	-	✓	-	-

Create index-
create index sn on groupg6(sname);



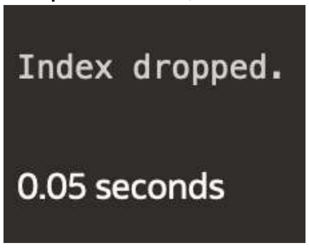
```
Index created.

0.04 seconds
```

Create composite index-
create index de on groupg6(roll_no,sname);



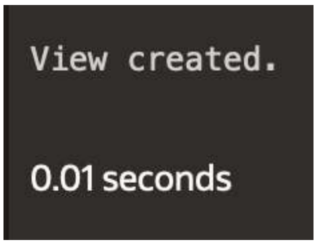
```
Index created.

0.04 seconds
```

Drop index –
drop index sn;



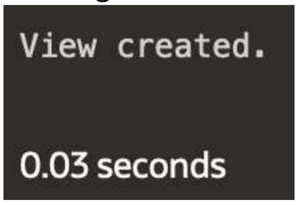
```
Index dropped.

0.05 seconds
```

Create view –
create view stu_view as select * from groupg6;



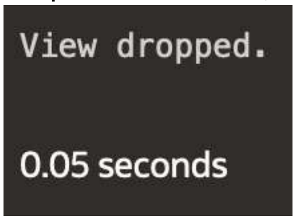
```
View created.

0.01 seconds
```

create view studel as select g.roll_no, s.sname, s.marks from groupg6 g, student s where g.roll_no = s.roll_no;



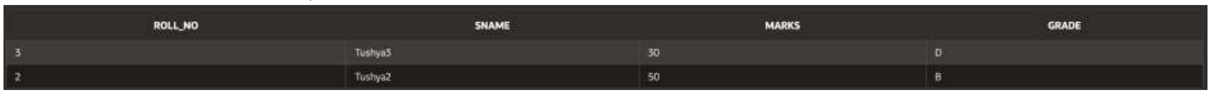
```
View created.

0.03 seconds
```

Drop view –
drop view stu_view;



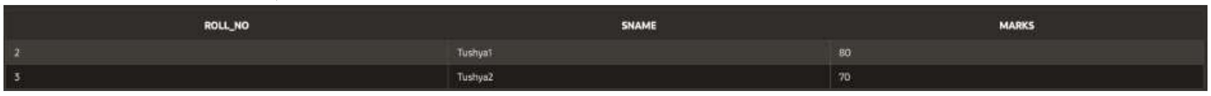
```
View dropped.

0.05 seconds
```

Query on view –
select * from stu_view;



ROLL_NO	SNAME	MARKS	GRADE
3	Tushya3	30	D
2	Tushya2	50	B

select * from studel;



ROLL_NO	SNAME	MARKS
2	Tushya1	80
3	Tushya2	70

To implement the usage of PL/SQL

PL/SQL is a block structure language. The PL/SQL programs are divided and written in logical blocks of code. Each block consists of 3 sub parts –

- Declarations – This section starts with a keyword DECLARE. It is an optional section and defines all variables to be used in the program.
- Executable commands – This section is enclosed between the keywords BEGIN and END. It is a mandatory section that consists of the executable pl/sql statements of the program. It must have at least one executable line of code.
- Exception handling – This section starts with the keyword EXCEPTION. It is an optional section and contains exceptions that handle errors in the program.

Every statement In PL/SQL ends with a semicolon(;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END.

Displaying messages on the screen

- **dbms_output** – This is a package that includes a number of procedures and functions that accumulate information in a buffer so that I can be retrieved later. These functions can be used to display message to the user.
- **put_line** – This command puts a piece of information in the buffer followed by an end of line marker and is used to display a message to the user.

Setting the server output ON

```
SET SERVER OUTPUT ON END;
```

Conditional control in PL/SQL

if-else

```
IF <condition> THEN <action>
ELSIF <condition>
        <action>
ELSE
        <action>
ENDIF;
```

While loop

```
WHILE <condition>
LOOP
        <action>
END LOOP;
```

Loop while

```
LOOP
        <action>
END WHEN <condition>
END LOOP;
```

For loop

```
FOR VARIABLE IN [REVERSE] START...END
LOOP
        <action>
` END LOOP;
```

Goto

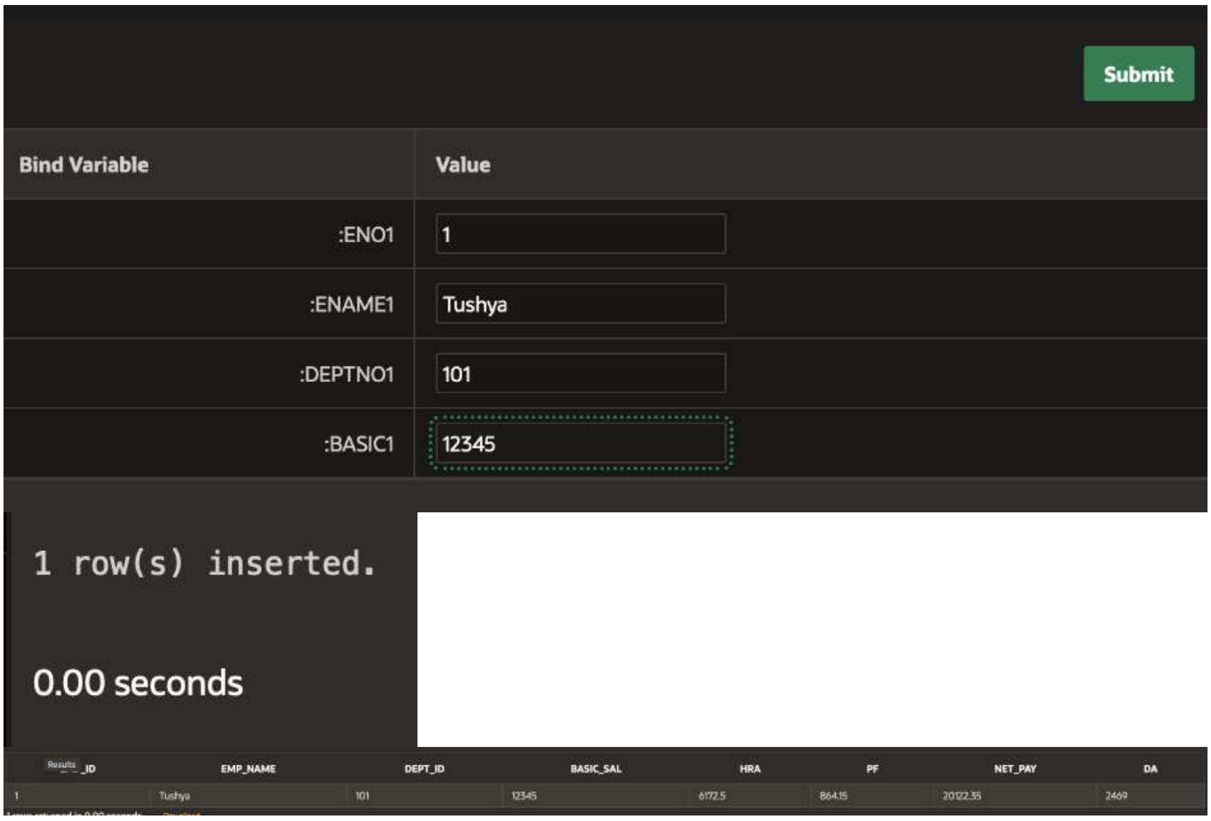
```
GOTO <statement label>
```

Example

```
DECLARE
<declaration of memory variables used later>
BEGIN
<SQL executable statements for manipulating table data>
EXCEPTION
<SQL and PL/SQL code to handle errors>
END;
```

Q1. Create a PL/SQL block for inserting values in the employe table. Only emp_id, emp_name, dept, basic_sal should be received as input while executing the block and for the rest of the fields the values need to be calculated.
HRA = 0.5 * basic_sal
DA = 0.2 * basic_sal
PF = 0.07 * basic_sal
Net_pay = basic_sal + HRA + DA – PF

```
DECLARE
  Eno1 employee.emp_id % type;
  Ename1 employee.emp_name % type;
  Deptno1 employee.dept_id % type;
  basic1 employee.basic_sal % type;
  HRA1 employee.hra % type;
  DA1 employee.da % type;
  PF1 employee.pf % type;
  NETPAY1 employee.net_pay % type;
BEGIN
  Eno1 := :Eno1;
  Ename1 := :Ename1;
  Deptno1 := :Deptno1;
  basic1 := :basic1;
  HRA1 := (basic1 * 50 ) / 100 ;
  DA1 := (basic1 * 20) / 100;
  PF1 := (basic1 * 7) / 100;
  NETPAY1 := basic1 + HRA1 + DA1 - PF1;
  INSERT into employee(emp_id, emp_name, dept_id, basic_sal, hra, da, pf, net_pay)
VALUES(Eno1,Ename1,Deptno1,basic1,HRA1,DA1,PF1,NETPAY1);
END;
```



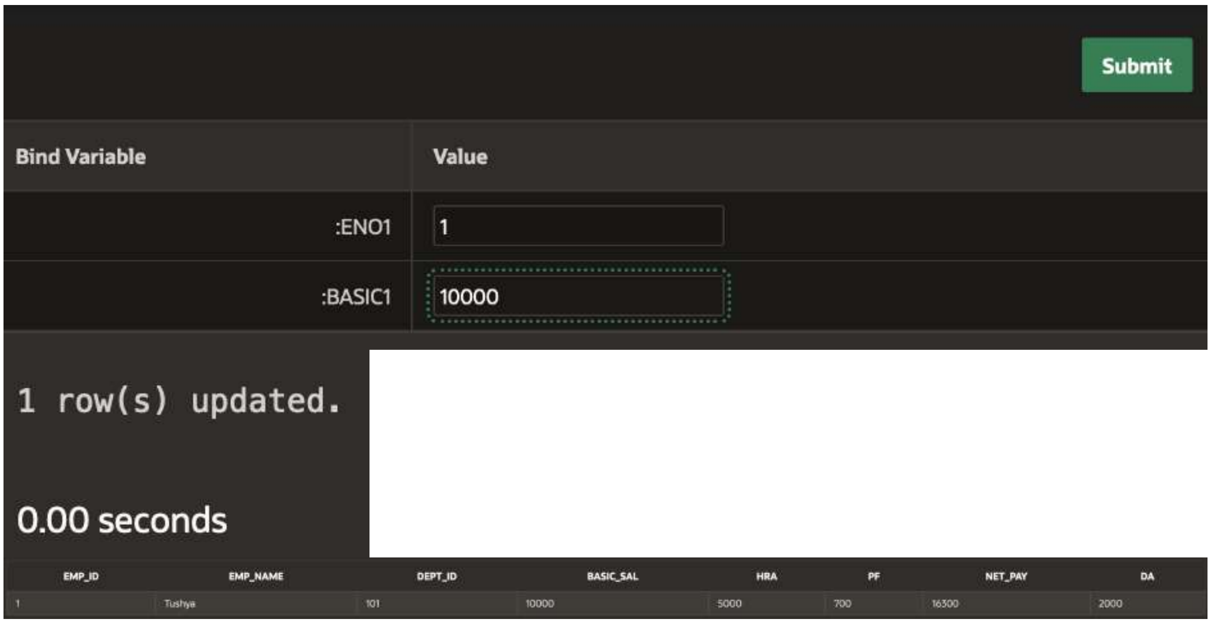
Bind Variable	Value
:ENO1	1
:ENAME1	Tushya
:DEPTNO1	101
:BASIC1	12345

```
1 row(s) inserted.

0.00 seconds
```

_ID	EMP_NAME	DEPT_ID	BASIC_SAL	HRA	PF	NET_PAY	DA
1	Tushya	101	12345	6172.5	864.15	20122.35	2469

Q2. Create a PL/SQL block for updating records in employe table where the user should provide emp_id and new basic_sal and thus the HRA, DA, PF and net_pay should get calculated and updated accordingly.

```
DECLARE
  Eno1 employee.emp_id % type;
  basic1 employee.basic_sal % type;
  HRA1 employee.hra % type;
  DA1 employee.da % type;
  PF1 employee.pf % type;
  NETPAY1 employee.net_pay % type;
BEGIN
  Eno1 := :Eno1;
  basic1 := :basic1;
  HRA1 := (basic1 * 50 ) / 100 ;
  DA1 := (basic1 * 20) / 100;
  PF1 := (basic1 * 7) / 100;
  NETPAY1 := basic1 + HRA1 + DA1 - PF1;
  UPDATE employee SET
basic_sal=basic1,hra=HRA1,da=DA1,pf=PF1,net_pay=NETPAY1 WHERE emp_id=Eno1;
END;
```



Bind Variable	Value
:ENO1	1
:BASIC1	10000

```
1 row(s) updated.
```

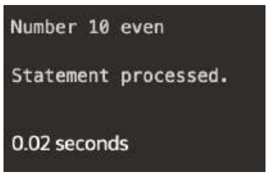
0.00 seconds

EMP_ID	EMP_NAME	DEPT_ID	BASIC_SAL	HRA	PF	NET_PAY	DA
1	Tushya	101	10000	5000	700	16300	2000

PL/SQL

Q1. Write a PL/SQL block for checking is a number is odd/even

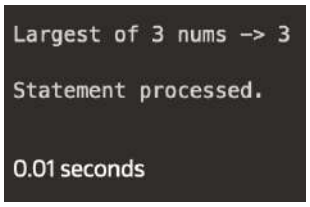
```
DECLARE
  num number(3);
  rem number;
BEGIN
  num := :num;
  rem := mod(num,2);
  IF rem=0 THEN
    dbms_output.put_line('Number ' || num || ' even');
  ELSE
    dbms_output.put_line('Number ' || num || ' odd');
  END IF;
END
```



```
Number 10 even

Statement processed.

0.02 seconds
```

Q2. Write a PL/SQL block to find the largest of given three numbers.

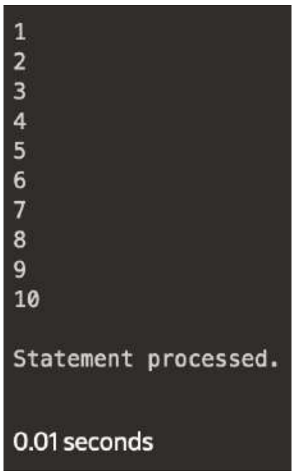
```
DECLARE
  a1 number;
  b1 number;
  c1 number;
  l number;
BEGIN
  a1 := :a1;
  b1 := :b1;
  c1 := :c1;
  l := a1;
  IF b1>l THEN
    l := b1;
  END IF;
  IF c1>l THEN
    l := c1;
  END IF;
  dbms_output.put_line('Largest of 3 nums -> ' || l);
END
```



```
Largest of 3 nums -> 3

Statement processed.

0.01 seconds
```

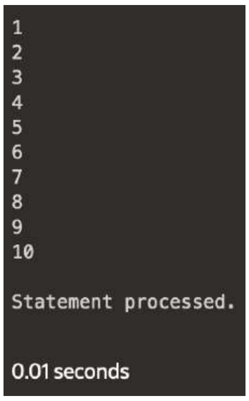
Q3. Write a PL/SQL block to generate first 10 natural numbers using loop, while and for loop.

```
DECLARE
  num number;
BEGIN
  num := 1;
  WHILE num<11 LOOP
    dbms_output.put_line(num);
    num := num + 1;
  END LOOP;
END
```



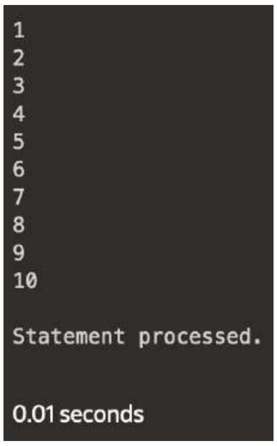
```
1
2
3
4
5
6
7
8
9
10

Statement processed.


0.01 seconds
```

```
DECLARE
  num number;
BEGIN
  num := 1;
  LOOP
    dbms_output.put_line(num);
    num := num + 1;
    EXIT WHEN num>10;
  END LOOP;
END
```



```
1
2
3
4
5
6
7
8
9
10

Statement processed.


0.01 seconds
```

```
DECLARE
  num number;
BEGIN
  num := 1;
  FOR num in 1 .. 10 LOOP
    dbms_output.put_line(num);
  END LOOP;
END
```



```
1
2
3
4
5
6
7
8
9
10

Statement processed.

0.01 seconds
```

To Implement the use of stored procedures in PL/SQL

An oracle stored procedure is a program stored in an oracle database. It is a precompiled set of SQL statements that can be shared by a number of programs.

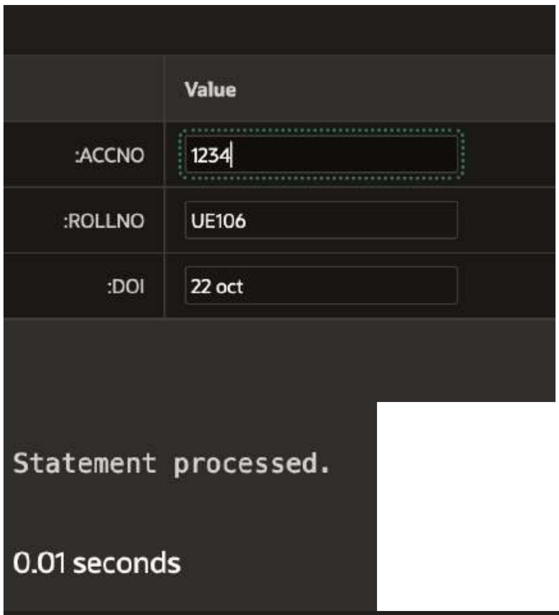
Syntax –

```
CREATE[or REPLACE] PROCEDURE procedure_name(parameter_list)
IS
        <declaration_section>
BEGIN
        <executable_section>
EXCEPTION
        <exception_section>
END;
```

Q1. Write a procedure to insert a record in borrow relation(ie borrow table). Before inserting check whether the book is available or not.

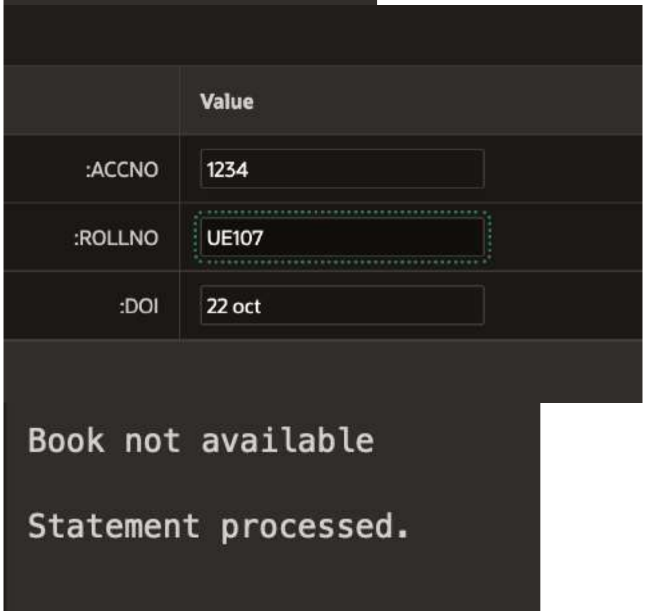
```
CREATE OR REPLACE PROCEDURE
  proc_borrow(
    acc_no borrow.acc_no%type,
    roll_no borrow.roll_no%type,
    DOI borrow.DOI%type
  )
IS
  CNT NUMBER(5);
  ACCNO NUMBER(10);
BEGIN
  ACCNO := acc_no;
  SELECT COUNT(*) INTO CNT
  FROM borrow
  WHERE acc_no=ACCNO;
  IF(CNT=0) THEN
    INSERT INTO borrow values(acc_no,roll_no,DOI);
  ELSE
    dbms_output.put_line('Book not available');
  END IF;
END;

DECLARE
  ACCNO borrow.acc_no%type;
  ROLLNO borrow.roll_no%type;
  DOI borrow.DOI%type;
BEGIN
```

```
   ACCNO := :ACCNO;
   ROLLNO := :ROLLNO;
   DOI := :DOI;
   proc_borrow(ACCNO,ROLLNO,DOI);
END;
```



	Value
:ACCNO	1234
:ROLLNO	UE106
:DOI	22 oct

```
Statement processed.

0.01 seconds
```



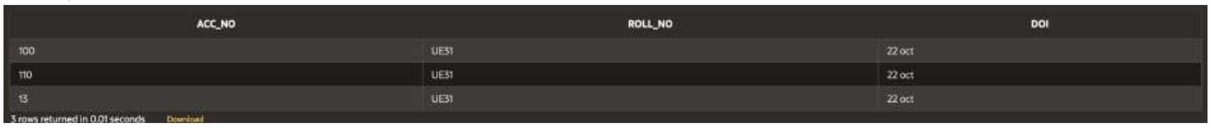
	Value
:ACCNO	1234
:ROLLNO	UE107
:DOI	22 oct

```
Book not available

Statement processed.
```

Q2. Write a procedure to insert a record in borrow relation with the same constraints as in the previous procedure and also ensure that a member has not borrowed more than 3 books.

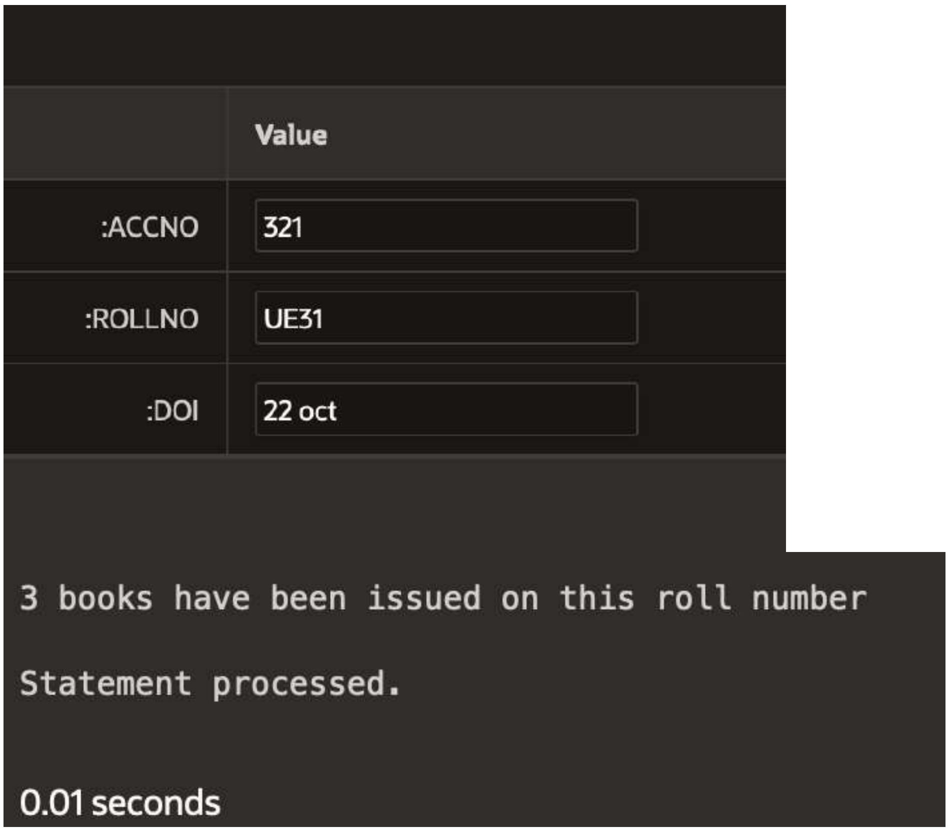
```
CREATE OR REPLACE PROCEDURE
   proc_borrow2(
      acc_no borrow.acc_no%type,
      roll_no borrow.roll_no%type,
      DOI borrow.DOI%type
```

```
  )
IS
  CNT NUMBER(5);
  CNTR NUMBER(5);
  ACCNO NUMBER(10);
  ROLLNO VARCHAR(8);
BEGIN
  ACCNO := acc_no;
  ROLLNO := roll_no;

  SELECT COUNT(*) INTO CNT
  FROM borrow
  WHERE acc_no=ACCNO;

  IF(CNT=0) THEN
    SELECT COUNT(*) INTO CNTR
    FROM borrow
    WHERE roll_no=ROLLNO;
    IF (CNTR<3) THEN
      INSERT INTO borrow values(acc_no,roll_no,DOI);
    ELSE
      dbms_output.put_line('3 books have been issued on this roll number');
    END IF;
  ELSE
    dbms_output.put_line('Book not available');
  END IF;
END;

DECLARE
  ACCNO borrow.acc_no%type;
  ROLLNO borrow.roll_no%type;
  DOI borrow.DOI%type;
BEGIN
  ACCNO := :ACCNO;
  ROLLNO := :ROLLNO;
  DOI := :DOI;
  proc_borrow2(ACCNO,ROLLNO,DOI);
END;
```



ACC_NO	ROLL_NO	DOI
100	UE31	22 oct
110	UE31	22 oct
13	UE31	22 oct

3 rows returned in 0.01 seconds Download



	Value
:ACCNO	321
:ROLLNO	UE31
:DOI	22 oct

```
3 books have been issued on this roll number

Statement processed.
```

0.01 seconds

To Implement the use of PL/SQL functions

A PL/SQL function is the same as a procedure except that it returns a value

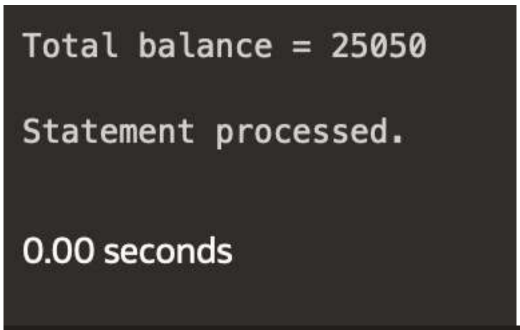
Syntax –

```
CREATE[or REPLACE] FUNCTION function_name(parameter_list [IN|OUT|INOUT]
type[,...])
RETURN return_datatype
[IS|AS]
BEGIN
        <function_body>
END;
```

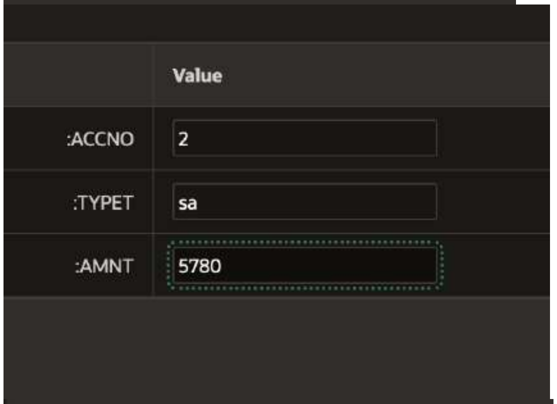
Q1. Create a function to insert the recording into the transaction table after preforming each transaction in transaction table. Show the net balance of the account.

```
CREATE OR REPLACE FUNCTION
  func_trans(
    acc_no trans.acc_no%type,
    amnt trans.amnt%type,
    typet trans.typet%type
  )
RETURN NUMBER
IS
  BAL NUMBER;
  ACCNO NUMBER;
BEGIN
  ACCNO := acc_no;
  INSERT INTO trans VALUES(acc_no, amnt,typet);
  SELECT SUM(amnt) INTO BAL
  FROM trans
  WHERE acc_no=ACCNO;
  RETURN BAL;
END;

DECLARE
  bal_amnt NUMBER(10);
  ACCNO NUMBER(10);
  amnt NUMBER(10);
  typet VARCHAR(2);
BEGIN
  ACCNO := :ACCNO;
  typet := :typet;
  amnt := :amnt;
```

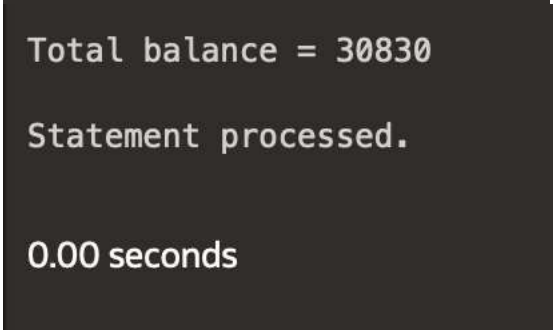
```
   bal_amnt := func_trans(ACCNO,amnt,typet);
   dbms_output.put_line('Total balance = ' || bal_amnt);
END;
```



```
Total balance = 25050

Statement processed.

0.00 seconds
```



	Value
:ACCNO	2
:TYPET	sa
:AMNT	5780



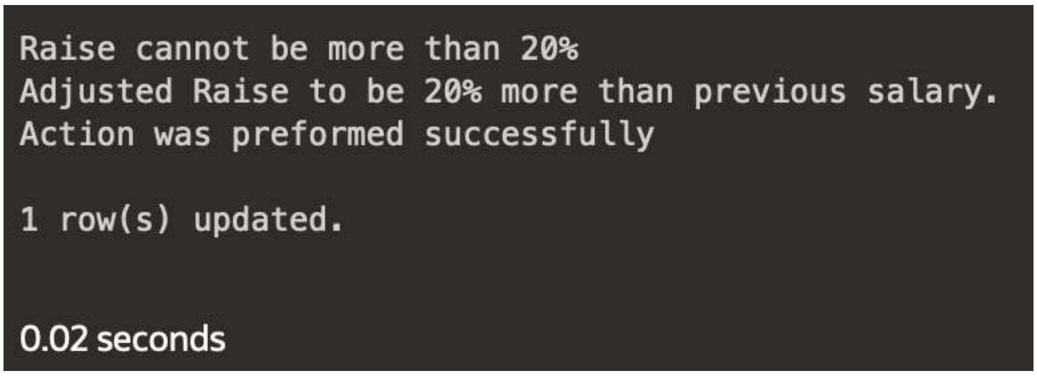
```
Total balance = 30830

Statement processed.

0.00 seconds
```

Triggers in PL/SQL

Row level triggers –

- BEFORE

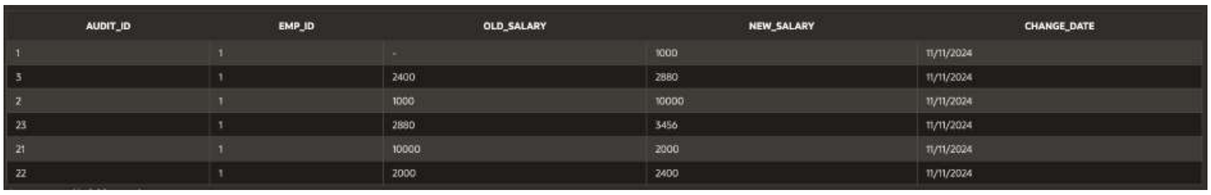
```
CREATE OR REPLACE TRIGGER check_sal
BEFORE UPDATE of salary on employees
FOR EACH ROW
DECLARE
  diff NUMBER(8,2);
  thresh NUMBER(8,2);
BEGIN
  diff := :NEW.salary - :OLD.salary;
  thresh := :OLD.salary * 0.2;
  IF diff > thresh THEN
    :NEW.salary := :OLD.salary + thresh;
    dbms_output.put_line('Raise cannot be more than 20%');
    dbms_output.put_line('Adjusted Raise to be 20% more than previous
salary.');
  END IF;
END
```



```
Raise cannot be more than 20%
Adjusted Raise to be 20% more than previous salary.
Action was preformed successfully

1 row(s) updated.


0.02 seconds
```

- AFTER

```
CREATE OR REPLACE TRIGGER enter_audit
AFTER INSERT or UPDATE of salary on employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_audit
VALUES(emp_audit.NEXTVAL,:NEW.emp_id,:OLD.salary,:NEW.salary,SYS
DATE);
END
```

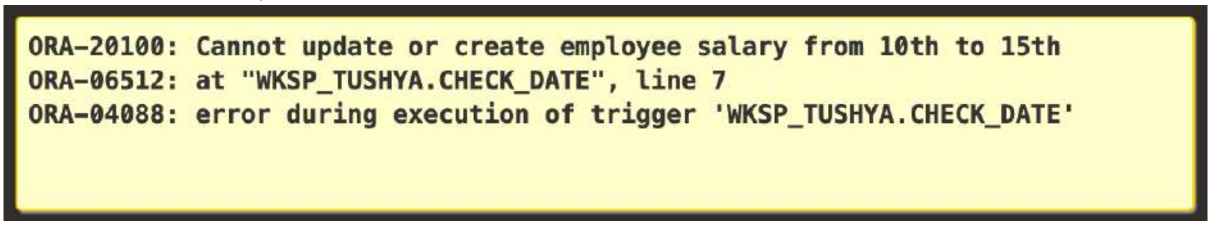


AUDIT_ID	EMP_ID	OLD_SALARY	NEW_SALARY	CHANGE_DATE
1	1	-	1000	11/11/2024
3	1	2400	2880	11/11/2024
2	1	1000	10000	11/11/2024
23	1	2880	3456	11/11/2024
21	1	10000	2000	11/11/2024
22	1	2000	2400	11/11/2024

Statement level –

- BEFORE

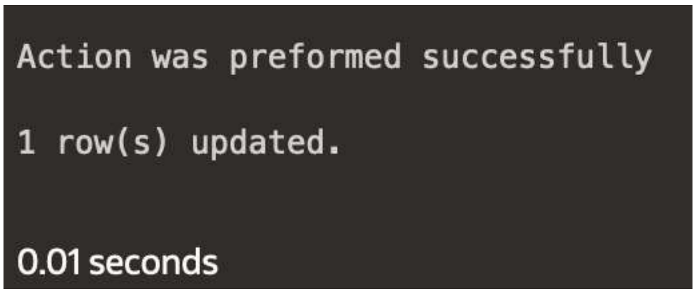
```
CREATE OR REPLACE TRIGGER check_date
BEFORE INSERT OR UPDATE on employees
DECLARE
  l_day_of_month NUMBER;
BEGIN
  l_day_of_month := EXTRACT(DAY FROM sysdate);

  IF l_day_of_month BETWEEN 10 AND 15 THEN
    raise_application_error(-20100,'Cannot update or create employee
salary from 10th to 15th');
  END IF;
END;
```



```
ORA-20100: Cannot update or create employee salary from 10th to 15th
ORA-06512: at "WKSP_TUSHYA.CHECK_DATE", line 7
ORA-04088: error during execution of trigger 'WKSP_TUSHYA.CHECK_DATE'
```

- AFTER

```
CREATE OR REPLACE TRIGGER succ_transaction
AFTER INSERT OR UPDATE on employees
BEGIN
  dbms_output.put_line('Action was preformed successfully');
END
```



```
Action was preformed successfully

1 row(s) updated.


0.01 seconds
```

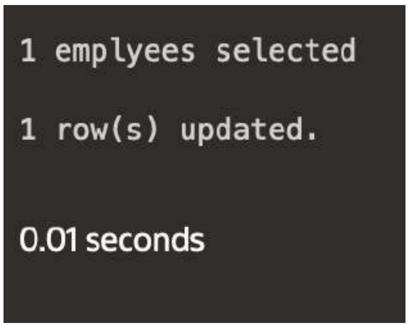
Cursors in PL/SQL

When a sql statement is processed a memory area or context area is created in memory which contains all information needed for processing the statements. For e.g. no of rows processed.

A cursor is a pointer to context area. It holds the rows(1 or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set. We can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement at a time.

Implicit cursor – These are created automatically whenever an SQL statement is executed. For insert the cursor holds the data to be inserted. For update or delete the cursor holds the data to be updated to deleted.

```
DECLARE
  total_rows number(3);
BEGIN
  UPDATE employee2 SET salary = salary + 500 WHERE emp_id=1;
  IF SQL%NOTFOUND THEN
    dbms_output.put_line('No employees of said id');
  ELSIF SQL%FOUND THEN
    total_rows := SQL%ROWCOUNT;
    dbms_output.put_line(total_rows||' emplyees selected');
  END IF;
END
```

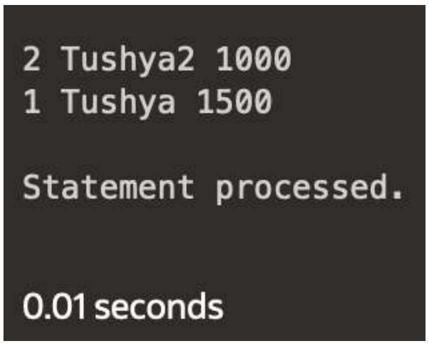


```
1 emplyees selected

1 row(s) updated.


0.01 seconds
```

Explicit cursor – These are programmer defined cursors. It is defined in the declaration section of the PLSQL block. It is created on a select statement which returns more than 1 row.

```
DECLARE
  c_eid employee2.emp_id%type;
  c_fn employee2.fname%type;
  c_sal employee2.salary%type;
  CURSOR c_emp IS select emp_id,fname,salary from employee2;
BEGIN
  OPEN c_emp;
  LOOP
```

```
    FETCH c_emp INTO c_eid,c_fn,c_sal;
    EXIT WHEN c_emp%NOTFOUND;
    dbms_output.put_line(c_eid || ' ' || c_fn || ' ' || c_sal);
  END LOOP;
  CLOSE c_emp;
END
```



```
2 Tushya2 1000
1 Tushya 1500

Statement processed.


0.01 seconds
```

Attributes –

- %FOUND – This returns true if an insert, update or delete statement effects 1 or more rows or a select into statement returns 1 or more rows otherwise it returns a false.
- %NOTFOUND – Opposite of %FOUND
- %ISOPEN – This always returns false for implicit cursors as SQL cursor is closed when SQL statement is executed.
- %ROWCOUNT – Returns the number of rows affected by an insert, update or delete statement or returned by a select into statement.

```
CURSOR cursor_name IS select_statement;
CURSOR c_customers IS select emp_id, fname from emplyee

OPEN cursor_name;
OPEN c_customers;

FETCH cursor_name INTO attributes;
FETCH c_customers INTO emp_id, fname;

CLOSE cursor_name;
CLOSE c_custormers;
```