DATABASE MANAGEMENT SYSTEMS LAB
FACULTY MANUAL
II Year II Semester

Prepared by

Dr. S. Narayana
Professor

Mrs. G. Keerthi
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GUDLAVALLERU ENGINEERING COLLEGE
(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)
Seshadrirao Knowledge Village, Gudlavalleru – 521356

GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institution with Permanent Affiliation to JNTUK, Kakinada)
Seshadri Rao Knowledge Village, Gudlavalleru – 521356

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE VISION & MISSION

Institute Vision

To be a leading institution of engineering education and research, preparing students for
leadership in their fields in a caring and challenging learning environment.

Institute Mission

- To produce quality engineers by providing state-of-the-art engineering education.
- To attract and retain knowledgeable, creative, motivated and highly skilled individuals whose leadership and contributions uphold the college tenets of education, creativity, research and responsible public service.
- To develop faculty and resources to impart and disseminate knowledge and information to students and also to society that will enhance educational level, which in turn, will contribute to social and economic betterment of society.
- To provide an environment that values and encourages knowledge acquisition and academic freedom, making this a preferred institution for knowledge seekers.
- To provide quality assurance.
- To partner and collaborate with industry, government, and R&D institutes to develop new knowledge and sustainable technologies and serve as an engine for facilitating the nation’s economic development.
- To impart personality development skills to students that will help them to succeed and lead.
- To instil in students the attitude, values and vision that will prepare them to lead lives of personal integrity and civic responsibility.
- To promote a campus environment that welcomes and makes students of all races, cultures and civilizations feel at home.
- Putting students face to face with industrial, governmental and societal challenges.

DEPARTMENT VISION & MISSION

VISION

To be a Centre of Excellence in Computer Science and Engineering education and training to meet the challenging needs of the industry and society.

MISSION

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.

PEO2: Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.

PEO3: Demonstrate commitment and progress in lifelong learning, professional development, leader ship and communicate effectively with professional clients and the public.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

Students will be able to

PSO1: Design, develop, test and maintain reliable software systems and intelligent systems. **PSO2**: Design and develop web sites, web apps and mobile apps.

Course Objectives

- To familiarize with creation of database and formulate SQL solutions to manipulate the database.
- To disseminate knowledge on integrity constraints, triggers and PL/SQL programs in a database environment.

Course Outcomes

Students will be able to

- create relational database with constraints.
- formulate simple and complex queries using features of Structured Query Language (SQL) for storage, retrieval and manipulation of data in a relational database.
- create views on relational database based on the requirements of users.
- implement PL/SQL programs for processing multiple SQL statements.
- implement triggers on a relational database.

Mapping Of Course Outcomes With Program Outcomes

ADVANCED DATA STRUCTURES AND ALGORITHMS LAB	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**PSO 1**	**PSO 2**
CO1:Create relational database with constraints	**3**	**3**	**3**					**2**	**2**	**2**	**2**	**2**	**2**	**3**
CO2:Formulate simple and complex queries using features of Structured Query Language (SQL) for storage, retrieval and manipulation of data in a relational database	**3**	**3**	**3**					**2**	**2**	**2**	**2**	**2**	**2**	**3**
CO3:Create views on relational database based on the requirements of users	**3**	**2**	**2**					**2**	**2**	**2**	**1**	**2**	**2**	**2**
CO4:Implement PL/SQL programs for processing multiple SQL statements														
CO5: Implement triggers on a relational database	**3**	**3**	**3**					**2**	**2**	**2**	**1**		**2**	**2**

EXERCISE: 1

AIM: Execute DDL, DML, DCL and TCL Commands.

Description

Structured query language (SQL) is a programming language used for storing and managing data in RDBMS.

Different data languages are:

1. DDL
2. DML
3. TCL
4. DCL

1. Data Definition Language(DDL)

DDL statements or commands are used to define database structure or schema.

1. Create
2. Alter
3. Drop
4. Truncate
5. Rename

CREATE

- Create command is used to create a table i.e create table command defines each column of the table uniquely.
- Each column has minimum of three attributes.
- Those are name, datatype, size.

Syntax

Create table <tablename>(<attribute><datatype(size)>,…..);

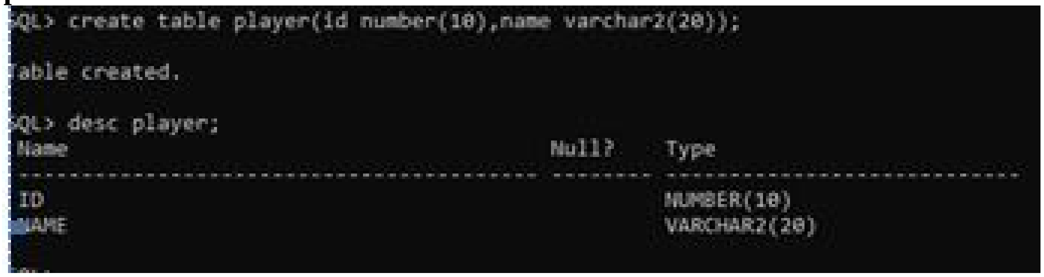
Example

Create table player(id number(10),name varchar2(20));

Table created.

Desc player;

Output



```
SQL> create table player(id number(10),name varchar2(20));

Table created.

SQL> desc player;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(10)
 NAME                                               VARCHAR2(20)
```

2. ALTER

Alter command is used to alter the structure of a database.

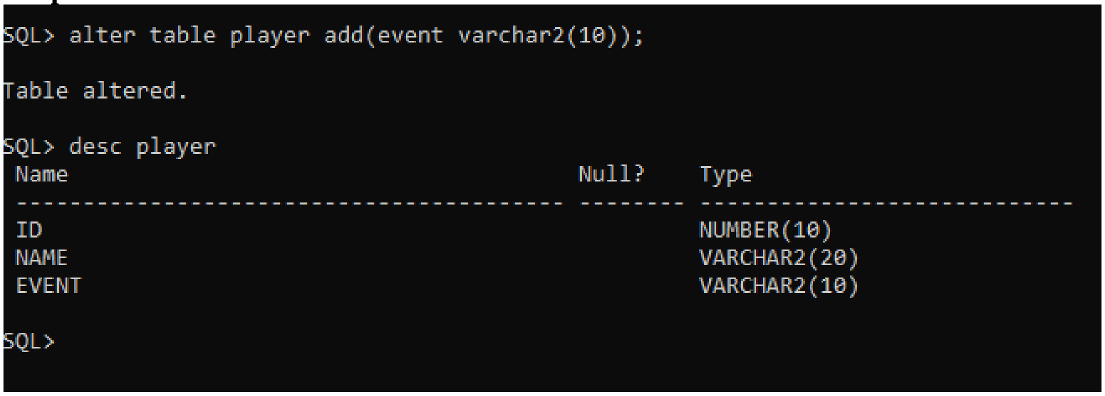
Syntax

alter table<tablename>add(<newattribute><datatype(size)>);

Example

alter table player add(event varchar2(10));
Table Altered.
desc player

Output



```
SQL> alter table player add(event varchar2(10));

Table altered.

SQL> desc player
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(10)
 NAME                                               VARCHAR2(20)
 EVENT                                              VARCHAR2(10)

SQL>
```

3. RENAME

Rename will be in two situations.

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

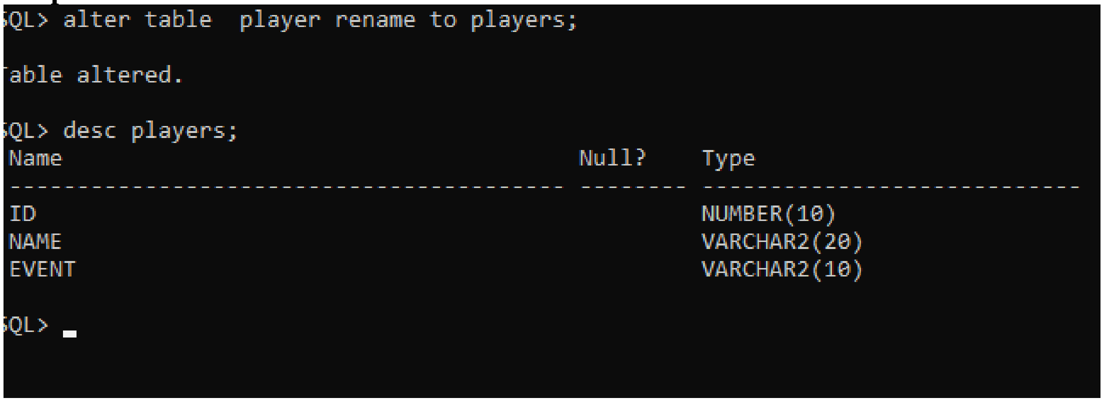
Syntax

i) alter table tablename rename to players.

Example

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

Output



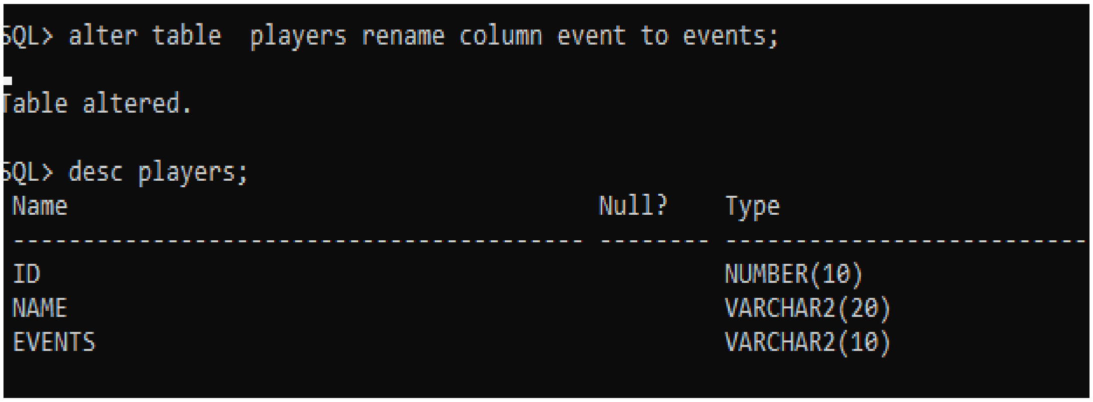
```
QL> alter table  player rename to players;

able altered.

QL> desc players;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(10)
 NAME                                               VARCHAR2(20)
 EVENT                                              VARCHAR2(10)

QL> _
```

ii) alter table tablename column<old-column> to <new-coloumn>

Example

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

Output



```
SQL> alter table  players rename column event to events;

Table altered.

SQL> desc players;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(10)
 NAME                                               VARCHAR2(20)
 EVENTS                                             VARCHAR2(10)
```

4. DROP

- Drop command is used to delete objects from the database i.e it will destroy table and all data which will be recorded in it.

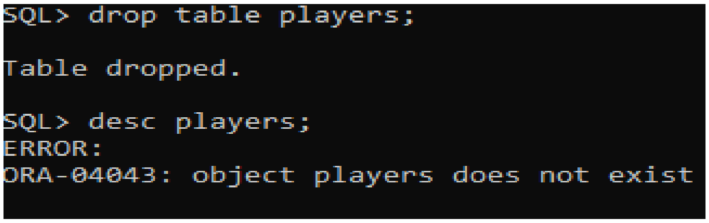
Syntax

Drop table<table name>;

Example

Drop table players;
table dropped.
desc players;

Output



```
SQL> drop table players;

Table dropped.

SQL> desc players;
ERROR:
ORA-04043: object players does not exist
```

5. TRUNCATE

- In Truncate command, table rows,indexes,privilege will also be removed.

Syntax

Truncate table<tablename>

Example

truncate table players;
select * from players;

Output

Object to be truncated.
No data found.

2. DATA MANIPULATION LANGUAGES(DML)

- DML is used for managing data within schema objects.
- Some commands are:

1. insert

2. update
3. delete
4. select

INSERT

- Insert command is used to insert data into a table.

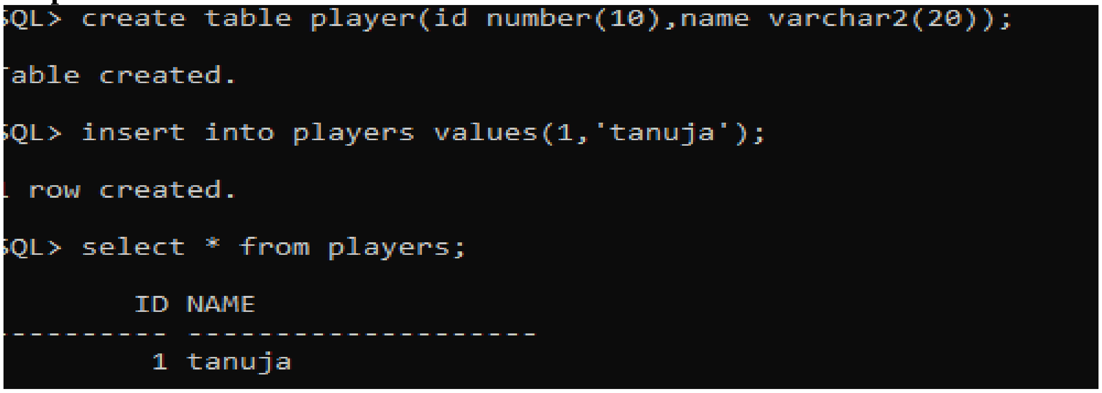
Syntax

Insert into <table name> values(data1,data2,….);

Example

Insert into players values(1,’tanuja’);
1 row created.
Select * from players;

Output



```
SQL> create table player(id number(10),name varchar2(20));

Table created.

SQL> insert into players values(1,'tanuja');

1 row created.

SQL> select * from players;

        ID NAME
---------- --------------------
         1 tanuja
```

2. UPDATE

- Update command is used to update existing data with the table.

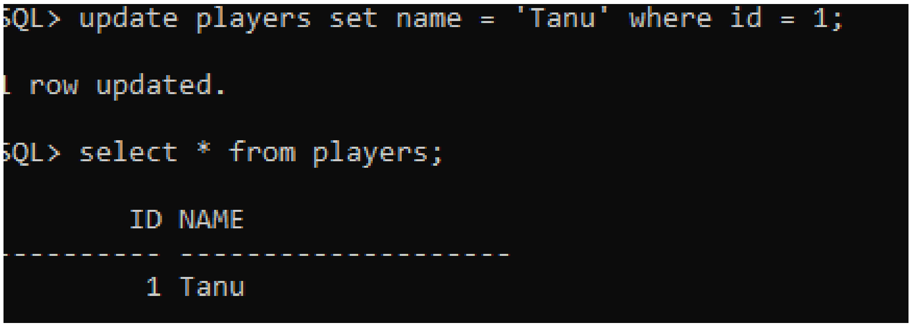
Syntax

update<table tablename>set column name = value where condition;

Example

Update players set name = ‘tanu’ where id = 1;
1 row is updated.
Select * from players;

Output



```
SQL> update players set name = 'Tanu' where id = 1;

1 row updated.

SQL> select * from players;

        ID NAME
---------- --------------------
         1 Tanu
```

3. DELETE

- Delete command is used to delete all records from a table, the spaces for the record remain.

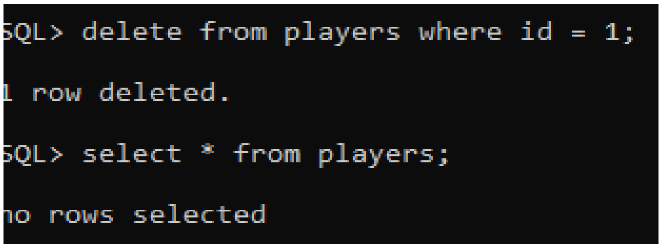
Syntax

- To delete a particular row.
  - ➢ Delete from <table name>where condition;

Example

Delete from players where id = 1;
Select * from players;

Output



```
SQL> delete from players where id = 1;

1 row deleted.

SQL> select * from players;

no rows selected
```

4. SELECT

- Select command is used to retrieve data from a database.

Syntax

select * from tablename;

Example

select * from players;

TCL commands: TCL statements allow you to control and manage transactions to maintain the integrity of data with SQL statements.

1. **COMMIT**

- Commit command is used to permanently save any transaction into database.

Syntax:

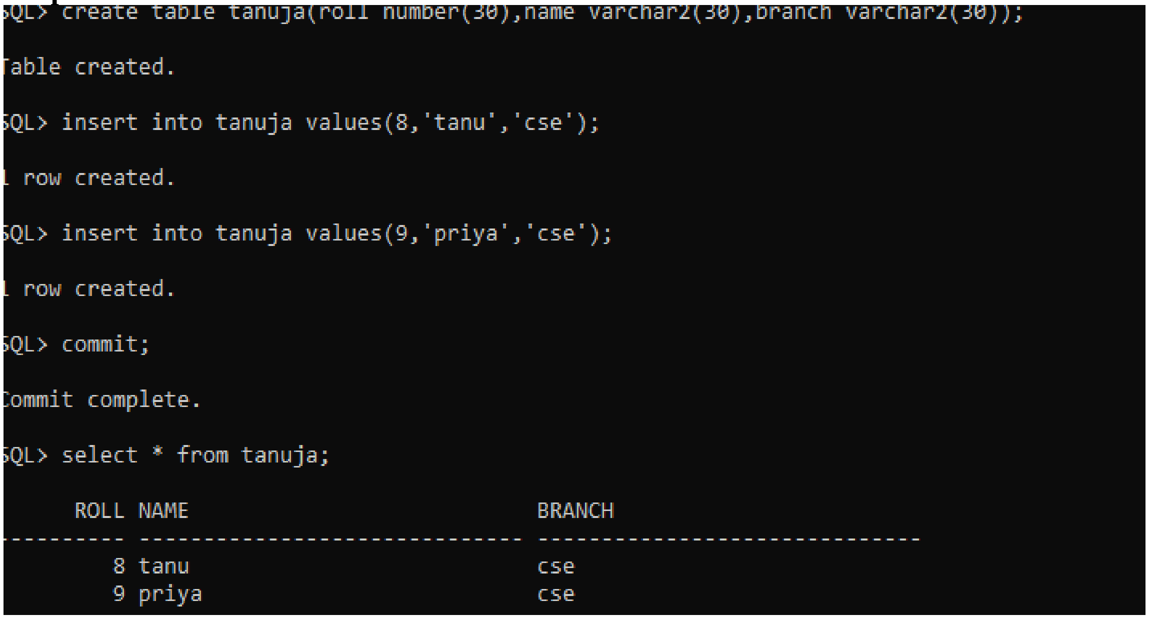
commit;

Example

create table tanuja(roll number(30),name varchar2(30),branch varchar2(30));

table created.
Insert into tanuja values(8,’tanu’,’cse’);
1 row created.
Insert into tanuja values(9,’priya’,’cse’);
1 row created.

Output



```
SQL> create table tanuja(roll number(30),name varchar2(30),branch varchar2(30));

Table created.

SQL> insert into tanuja values(8,'tanu','cse');

1 row created.

SQL> insert into tanuja values(9,'priya','cse');

1 row created.

SQL> commit;

Commit complete.

SQL> select * from tanuja;

      ROLL NAME                           BRANCH
---------- ------------------------------ ------------------------------
         8 tanu                           cse
         9 priya                          cse
```

2. ROLLBACK:

- This command restores the database to last commited state.
- It is also use with savepoint command to jump to a savepoint in a transaction.

Syntax

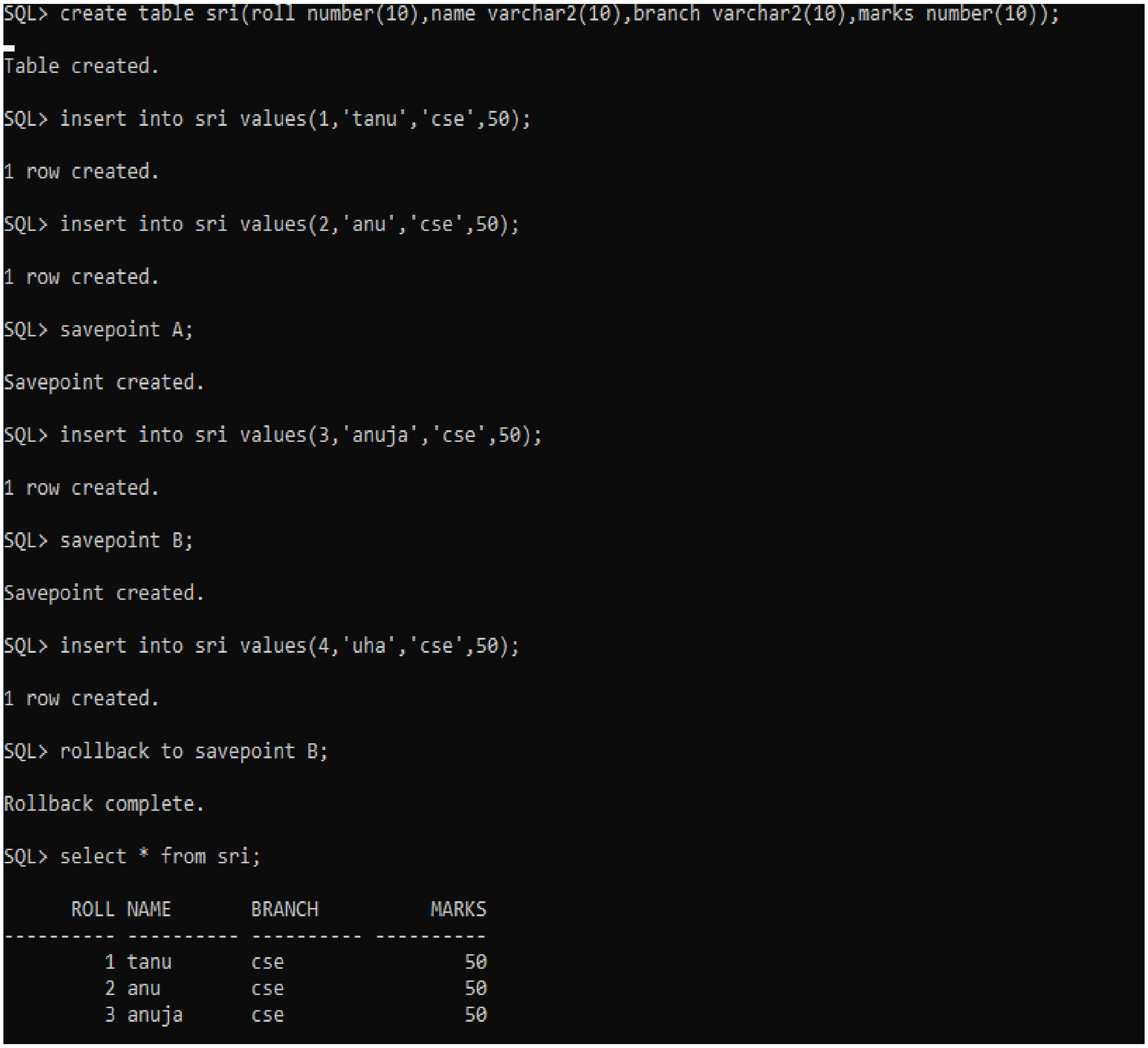
rollback to savepoint name;

Example

create table sri(roll number(10),name varchar2(10) ,branch varchar2(10),marks number(10));

Table created.
Insert into sri values(1,’tanu’,’cse’,50);
1 row created.
Insert into sri values(2,’anu’,’cse’,50);
1 row created.
Savepoint A;
Savepoint created.
Insert into sri values(3,’anuja’,’cse’,50);
1 row created.
Savepoint B;
Savepoint created.
Insert into sri values(4,’uha’,’cse’,50);
1 row created.
Rollback to savepoint B;
Rollback complete.
Select * from sri;

Output



```
SQL> create table sri(roll number(10),name varchar2(10),branch varchar2(10),marks number(10));

Table created.

SQL> insert into sri values(1,'tanu','cse',50);

1 row created.

SQL> insert into sri values(2,'anu','cse',50);

1 row created.

SQL> savepoint A;

Savepoint created.

SQL> insert into sri values(3,'anuja','cse',50);

1 row created.

SQL> savepoint B;

Savepoint created.

SQL> insert into sri values(4,'uha','cse',50);

1 row created.

SQL> rollback to savepoint B;

Rollback complete.

SQL> select * from sri;

      ROLL NAME       BRANCH          MARKS
---------- ---------- ---------- ----------
         1 tanu       cse                50
         2 anu        cse                50
         3 anuja      cse                50
```

3. SAVEPOINT

- It is used to temporarily save a transaction so that you can rollback to that point when ever necessary.

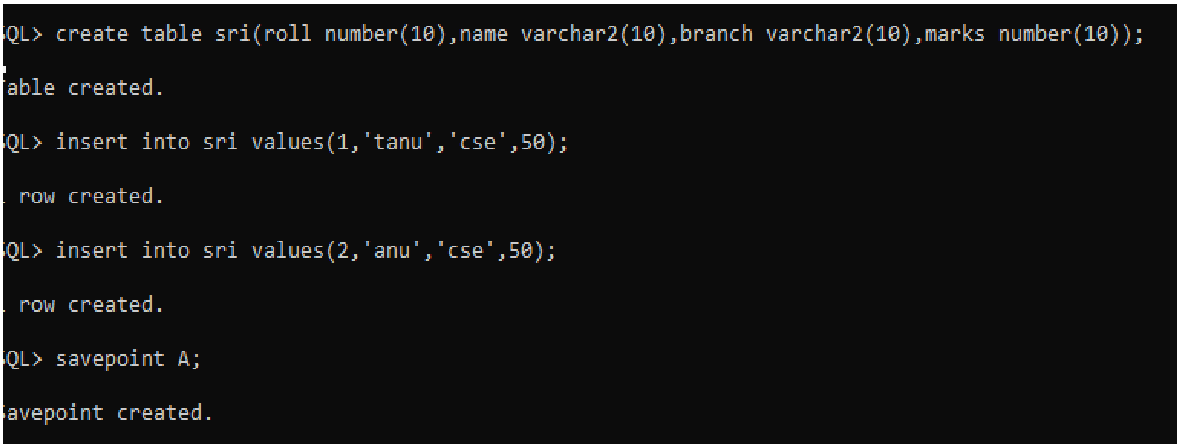
Syntax

Savepoint savepoint name;

Example

create table sri(id number(10),name varchar2(10));
table created.
Insert into sri values(1,’tanu’,’cse’,50);
1 row created.
Insert into sri values(2,’anu’,’cse’,50);
1 row created.
Savepoint A;
Savepoint created.

Output



```
QL> create table sri(roll number(10),name varchar2(10),branch varchar2(10),marks number(10));

able created.

QL> insert into sri values(1,'tanu','cse',50);

 row created.

QL> insert into sri values(2,'anu','cse',50);

 row created.

QL> savepoint A;

avepoint created.
```

DCL COMMANDS

1. **GRANT:**

- It gives acces privilege to data base

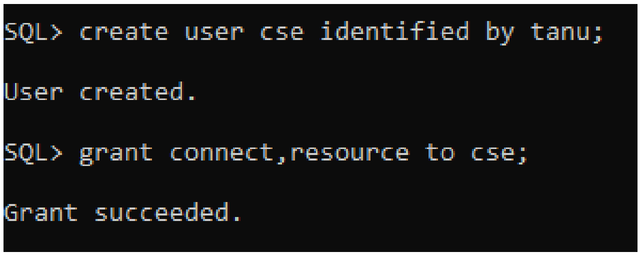
Syntax

grant create session to username;

Example

Create user cse identified by tanu;
User created.
Grant connect,resource to cse;
Grant succeded.

Output



```
SQL> create user cse identified by tanu;

User created.

SQL> grant connect,resource to cse;

Grant succeeded.
```

2. **REVOKE**

- Take back permissions from user.

Syntax

Revoke session from username;

Example

Revoke connect, resource from tanuja;
Revoke succeeded.

<u>VIVA QUESTIONS</u>

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

a) Primary key
b) Foreign key
c) Unique key
d) NOT NULL and Check

CONSTRAINTS

- KEY CONSTRAINTS
  - SUPER KEY – set of one or more attributes that uniquely identifies a tuple in a relation.
  - CANDIDATE KEY – minimal set of attributes that uniquely identifies a tuple in a relation.
  - PRIMARY KEY – is a key which uniquely identifies a tuple in a relation. the two properties of primary key are unique and not null.
  - FOREIGN KEY – Ensure the referential integrity of the data in one table to match values in another table.
- INTEGRITY CONSTRAINTS
  - CHECK - Ensures that the value in a column meets a specific condition
    E.g. check (account_balance>0)
  - NOT NULL - Indicates that a column cannot store NULL value.
    E.g. Account_number char(10) not null
  - UNIQUE - Ensures that each row for a column must have a unique value.
    E.g. unique(Name, DOB)

DEFINING DIFFERENT CONSTRAINTS ON A TABLE

A) PRIMARY KEY CONSTRAINT

- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly.
- **Syntax**

  create table <table name>(<attribute><datatype(<size>)> primary key,-------,---);

  Example: create table students(sid int primary key, name varchar(20),age int);

  OR

  create table students(sid int,name varchar(20),age int,primary key(id));

(i) WRITE A QUERY TO CREATE STD_MSTR TABLE BY APPLYING PRIMARY KEY CONSTRAINT

SQL> CREATE TABLE STD_MSTR(SNO VARCHAR2(10) PRIMARY KEY,SNAME VARCHAR2(20),DEPARTMENT VARCHAR2(10));

OUTPUT:-

Table Created

(ii) WRITE A QUERY TO DESCRIBE THE STRUCTURE OF STD-MSTR TABLE

SQL> DESC STD_MSTR;

OUTPUT:-

Name	Null?	Type
SNO	NOT NULL	VARCHAR2(10)
SNAME		VARCHAR2(20)

DEPARTMENT		VARCHAR2(10)

(iii) WRITE A QUERY TO INSERT VALUES INTO STD_MSTR TABLE

SQL> INSERT INTO STD_MSTR VALUES('08541F0043','SRINU','MCA');
OUTPUT:- 1 Row Created
SQL> INSERT INTO STD_MSTR VALUES('08541F0042','SIVA','MCA');
OUTPUT:- 1 Row Created
SQL> INSERT INTO STD_MSTR VALUES('08541F0041','TIGER','MBA');
OUTPUT:- 1 Row Created
SQL> INSERT INTO STD_MSTR VALUES('08541F0042','PANDU','EEE');
OUTPUT:-
ERROR at line 1:
ORA-00001: unique constraint (08541F0041.SYS_C003624) violated
SQL> INSERT INTO STD_MSTR VALUES(",'PANDU','ECE');
OUTPUT:-
ERROR at line 1:
ORA-01400: cannot insert NULL into ("08541F0041"."STD_MSTR"."SNO")

(iv) WRITE A QUERY TO DISPLAY STD_MSTR TABLE INFORMATION

SQL> SELECT * FROM STD_MSTR;
OUTPUT:-

SNO	SNAME	DEPARTMENT
08541F0043	SRINU	MCA
08541F0042	SIVA	MCA
08541F0041	TIGER	MBA

B) FOREIGN KEY CONSTRAINT

- FOREIGN KEY - Ensure the referential integrity of the data in one table to match values in another table.
- **Syntax:** create table <tablename>(<attribute><datatype(<size>)>,-----,foreign key(<attribute>) references <tablename>(<attribute>));
- **Example**

  Create table student (id int,name varchar(9));
  Create table enrolled (eid int,course varchar(20),foreign key(eid)references student(id));

(i) WRITE A QUERY TO CREATE STD_LIB TABLE BY APPLYING FOREIGN KEY CONSTRAINT

SQL> CREATE TABLE STD_LIB(STDNO VARCHAR2(10) REFERENCES STD_MSTR(SNO), BOOKNO VARCHAR2(10),BOOKNAME VARCHAR2(15),AUTHOR VARCHAR2(10));
OUTPUT:-
Table Created

(ii) WRITE A QUERY TO DESCRIBE THE STRUCTURE OF STD-LIB TABLE

SQL> DESC STD_LIB;
OUTPUT:-

Name	Null?	Type

STDNO		VARCHAR2(10)
BOOKNO		VARCHAR2(10)
BOOKNAME		VARCHAR2(15)
AUTHOR		VARCHAR2(10)

(iii) WRITE A QUERY TO INSERT VALUES INTO STD_LIB TABLE

SQL> INSERT INTO STD_LIB VALUES('08541F0042','CP43','C++','BAVE');

OUTPUT:- 1 Row Created

SQL> INSERT INTO STD_LIB VALUES('08541A0541','100','JAVA','SIVA NAGA');

OUTPUT:- 1 Row Created

SQL> INSERT INTO STD_LIB VALUES('08541A0542','255','C++','RAMS');

OUTPUT:-

ERROR at line 1:
ORA-02291: integrity constraint (08541F0041.SYS_C003656) violated - parent key not found

(iv) WRITE A QUERY TO DISPLAY TABLE INFORMATION

SQL> SELECT * FROM STD_LIB;

OUTPUT:-

STDNO	BOOKNO	BOOKNAME	AUTHOR
08541A0542	100	C++	RAMS
08541A0541	255	JAVA	SIVA NAGA

C) UNIQUE KEY CONSTRAINT

- UNIQUE - Ensures that each row for a column must have a unique value
  Create table student (sid int unique, name varchar(20), age int);

(i) WRITE A QUERY TO CREATE PRODUCT TABLE BY APPLYING UNIQUE KEY CONSTRAINT

SQL> CREATE TABLE PRODUCT(PRODUCTID NUMBER(3),STANDARDPRICE NUMBER(5),STARTDATE DATE,ENDDATE DATE UNIQUE);

OUTPUT:-

Table Created

(ii) WRITE A QUERY TO DESCRIBE THE STRUCTURE OF PRODUCT TABLE

SQL> DESC PRODUCT;

OUTPUT:-

Name	Null?	Type
PRODUCTID		NUMBER(3)
STANDARDPRICE		NUMBER(5)
STARTDATE		DATE

ENDDATE		DATE

(iii) WRITE A QUERY TO INSERT VALUES INTO PRODUCT TABLE

SQL> INSERT INTO PRODUCT VALUES(1,25,'12-SEP-08','12-AUG-09');

OUTPUT:- 1 Row Created

SQL> INSERT INTO PRODUCT VALUES(2,35,'12-OCT-08','12-SEP-09');

OUTPUT:- 1 Row Created

SQL> INSERT INTO PRODUCT VALUES(107,125,'17-JUL-08','12-AUG-09');

OUTPUT:-

ERROR at line 1:
ORA-00001: unique constraint (08541F0041.SYS_C003561) violated

(iv) WRITE A QUERY TO DISPLAY PRODUCT TABLE INFORMATION

SQL> SELECT * FROM PRODUCT;

OUTPUT:-

PRODUCTID	STANDARDPRICE	STARTDATE	ENDDATE
1	25	12-SEP-08	12-AUG-09
2	35	12-OCT-08	12-SEP-09

D) NOT NULL CONSTRAINT

- NOT NULL - Indicates that a column cannot store NULL value
  - Syntax:<column name><type>(<size>) not null
    Create table student (sid int not null,name varchar(20),age int);

(i) WRITE A QUERY TO CRATE ACCOUNTINFO TABLE BY APPLYING NOT NULL CONSTRAINT ON ACCNO FIELD

SQL> CREATE TABLE ACCOUNTINFO(ACCNO NUMBER(10) NOT NULL,NAME VARCHAR2(20),ACCTYPE VARCHAR2(20),TRANSACTION VARCHAR2(20), TRAN_DATE DATE,AMOUNT NUMBER(8,2));

OUTPUT:-

Table Created

(ii) WRITE A QUERY TO DESCRIBE THE STRUCTURE OF ACCOUNTINFO TABLE

SQL> DESC ACCOUNTINFO;

OUTPUT:-

Name	Null?	Type
ACCNO	NOT NULL	NUMBER(10)
NAME		VARCHAR2(20)
ACCTYPE		VARCHAR2(20)
TRANSACTION		VARCHAR2(20)
TRAN_DATE		DATE
AMOUNT		NUMBER(8,2)

(iii) WRITE A QUERY TO INSERT VALUES INTO ACCOUNTINFO TABLE

SQL> INSERT INTO ACCOUNTINFO VALUES(1092018805,'SRINU', 'SAVINGS', 'DEPOSIT','18-AUG-2009',15000);

OUTPUT:- 1 Row Created

```
SQL> INSERT INTO ACCOUNTINFO VALUES(1092017705,'SIVA',' ',
'DEPOSIT',    '29-AUG-2009',35000);
```

OUTPUT:- 1 Row Created

```
SQL> INSERT INTO ACCOUNTINFO
VALUES('','SAIBABA','CURRENT','WITHDRAW', '05-AUG-2009',15000);
```

OUTPUT:-

```
ERROR at line 1:
ORA-01400: cannot insert NULL into
("08541F0041"."ACCOUNTINFO"."ACCNO")
```

(iv) WRITE A QUERY TO DISPLAY ACCOUNINFO TABLE INFORMATION

```
SQL>  SELECT * FROM  ACCOUNTINFO;
```

OUTPUT:-

ACCNO	NAME	ACCTYPE	TRANSACTION	TRAN_DATE	AMOUNT
1092018805	SRINU	SAVINGS	DEPOSIT	18-AUG-09	15000
1092017705	SIVA		DEPOSIT	29-AUG-09	35000

E) CHECK CONSTRAINT

- CHECK - Ensures that the value in a column meets a specific condition
- Syntax:<column name><type>(<size>)check(<logical expression>)
  E.g. create table Student (s_id int NOT NULL CHECK(s_id > 0), Name varchar(60) NOT NULL, Age int);

(i) WRITE A QUERY TO CREATE A ORDERINFO1 TABLE BY APPLYING CHECK CONSTRAINT

```
SQL> CREATE TABLE ORDER(ORD_ID NUMBER(5)
CHECK(ORD_ID>100), ORD_DATE DATE,CUST_ID
NUMBER(5),QUANTITY NUMBER(5));
```

OUTPUT:-

Table Created

(ii) WRITE A QUERY TO DESCRIBE THE STRUCTURE OF ORDERINFO1 TABLE

```
SQL>  DESC  ORDERINFO1;
```

OUTPUT:-

Name	Null?	Type
ORD_ID		NUMBER(5)
ORD_DATE		DATE
CUST_ID		NUMBER(5)
QUANTITY		NUMBER(5)

(iii) WRITE A QUERY TO INSERT VALUES INTO ORDERINFO1 TABLE

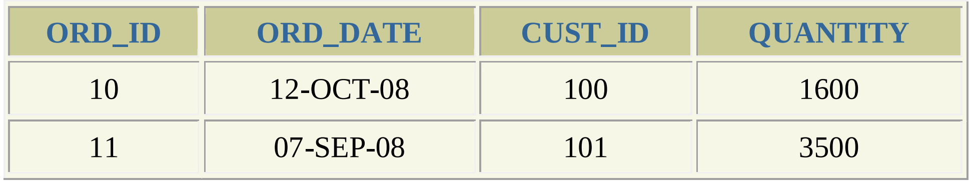
```
SQL>  INSERT INTO ORDER VALUES(101,'12-OCT-2008',100,1600);
```

OUTPUT:- 1 Row Created

```
SQL> INSERT INTO ORDER VALUES(111,'07-SEP-2008',200,3500);
OUTPUT:-        1    Row    Created
SQL> INSERT INTO ORDER VALUES(15,'07-NOV-2007',403,2500);
OUTPUT:-
ERROR at line 1:
ORA-02290: check constraint (08541F0041.SYS_C003525) violated
```

(iv) WRITE A QUERY TO DISPLAY ORDERINFO1 TABLE INFORMATION

```
SQL>  SELECT * FROM ORDER;
OUTPUT:-
```



ORD_ID	ORD_DATE	CUST_ID	QUANTITY
10	12-OCT-08	100	1600
11	07-SEP-08	101	3500

VIVA QUESTIONS

1) Define primary key.
2) Define foreign key.
3) What is the purpose of check and not null constraints.
4) How the primary key does differs from a candidate key? How they are similar?

EXERCISE: 3

AIM: Execute a single line and group (Aggregate) functions on Relation.

Description

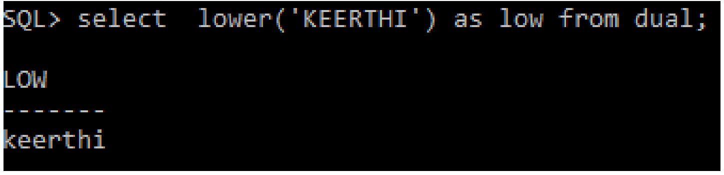
Single Row Functions

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

Syntax: lower(message)

Example

select lower('KEERTHI') as low from dual;



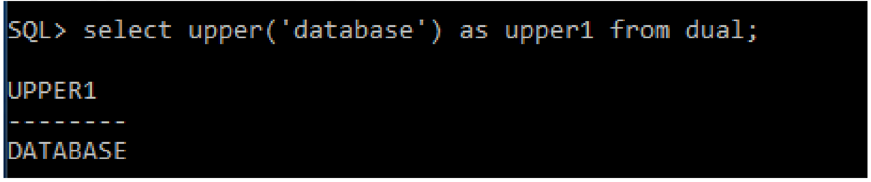
```
SQL> select  lower('KEERTHI') as low from dual;

LOW
-------
keerthi
```

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;



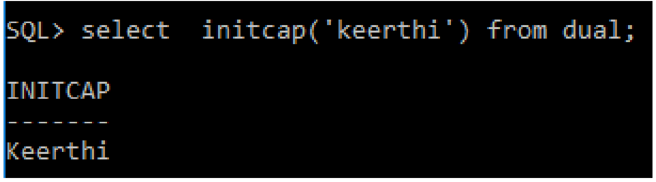
```
SQL> select upper('database') as upper1 from dual;

UPPER1
--------
DATABASE
```

3. initcap():

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

Syntax: initcap(message)

select initcap('keerthi') from dual;



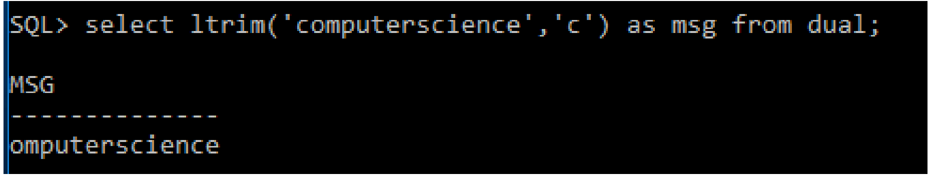
```
SQL> select  initcap('keerthi') from dual;

INITCAP
-------
Keerthi
```

4. 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;



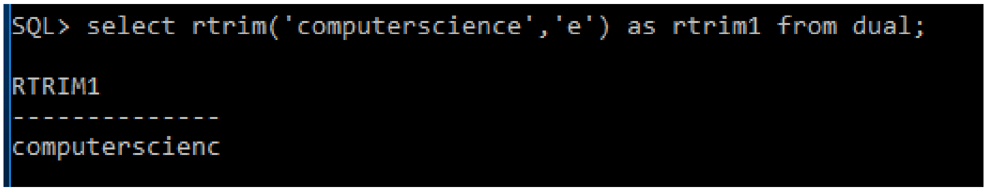
```
SQL> select ltrim('computerscience','c') as msg from dual;

MSG
--------------
omputerscience
```

5. rtrim()

This function is used for right trimming.

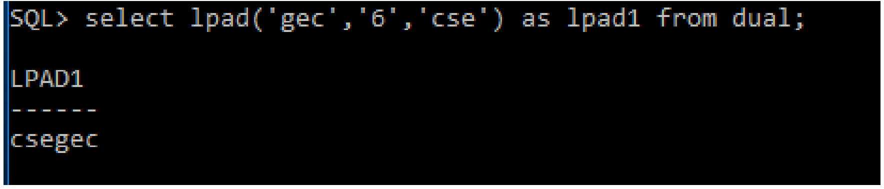
Syntax: rtrim('message','character')

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



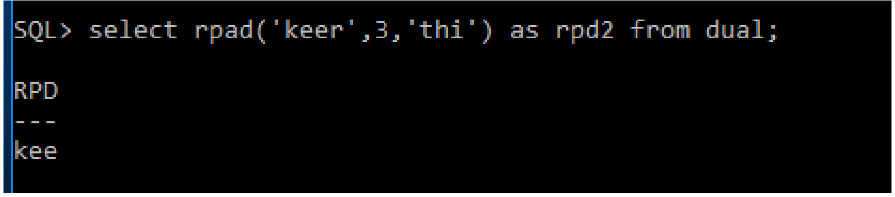
```
SQL> select rtrim('computerscience','e') as rtrim1 from dual;

RTRIM1
--------------
computerscienc
```

6. 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;



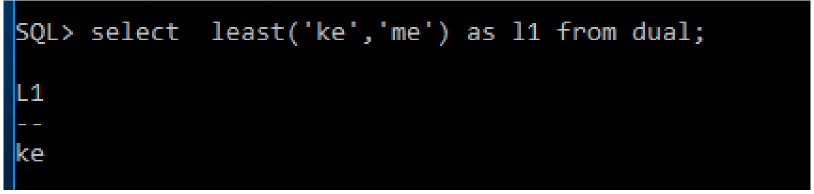
```
SQL> select lpad('gec','6','cse') as lpad1 from dual;

LPAD1
------
csegec
```

7. 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('keer',3,'thi') as rpd2 from dual;



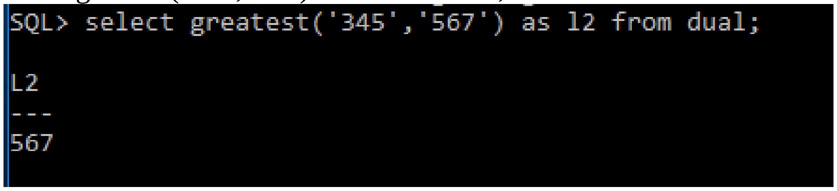
```
SQL> select rpad('keer',3,'thi') as rpd2 from dual;

RPD
---
kee
```

8. least(): this function is used to print the least value.
Syntax: least(string1,string2)
Example: select least('ke','me') as l1 from dual;
(or)
select least('345','567') as l2 from dual;



```
SQL> select  least('ke','me') as l1 from dual;

L1
--
ke
```

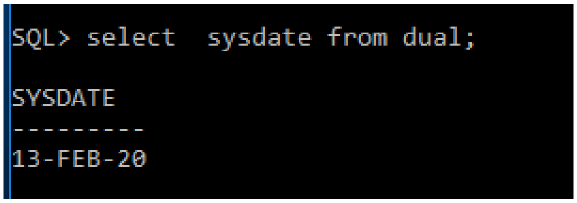
9. greatest(): this function is used to get maximum value .
Syntax: greatest(string1,string2)
Example: select greatest('ke','me') as l1 from dual;
(or)
select greatest('345','567') as l2 from dual;



```
SQL> select greatest('345','567') as l2 from dual;

L2
---
567
```

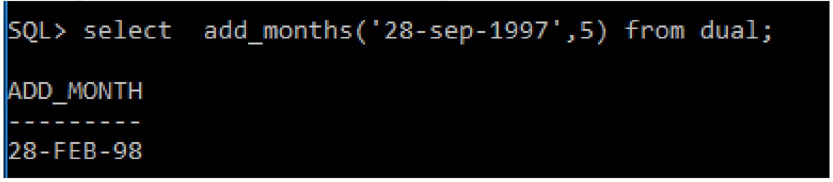
Dual Functions/Date Functions

1. current date: to get the current date.
Example: select sysdate from dual;



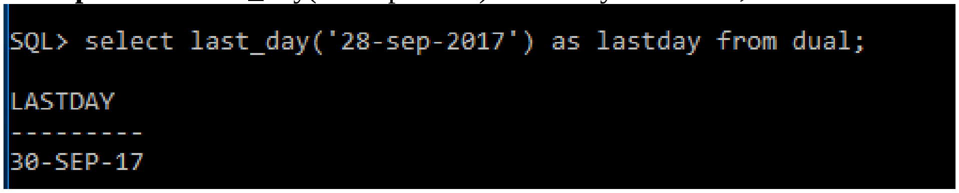
```
SQL> select  sysdate from dual;

SYSDATE
---------
13-FEB-20
```

2. 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;



```
SQL> select  add_months('28-sep-1997',5) from dual;

ADD_MONTH
---------
28-FEB-98
```

3. 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;

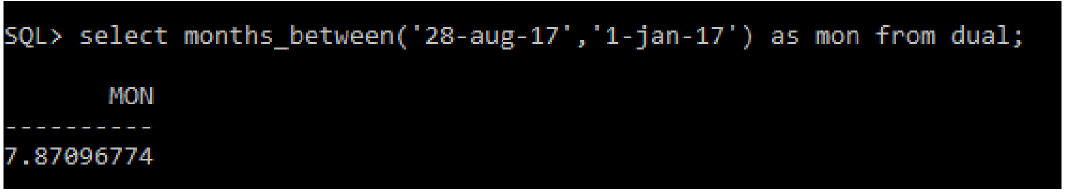


```
SQL> select last_day('28-sep-2017') as lastday from dual;

LASTDAY
---------
30-SEP-17
```

4. months_between(): it gives the number of months between specified two dates.

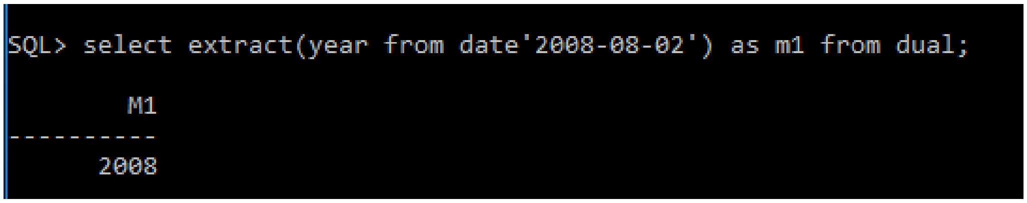
Result value	Months_between(date-exp1,date-exp2)
Negative result	If date-exp1 is earlier than date-exp2
Integer result	If date-exp1 and date-exp2 have the same day,or both specify the last day of the month.
Decimal result	If days are different and they are not both specify the last day of the month
Fractional part	Always calcilated as the difference between days divided by 31 despite the 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;



```
SQL> select months_between('28-aug-17','1-jan-17') as mon from dual;

       MON
----------
7.87096774
```

5. extract(): it is used to extract time component from date expression.
select extract(year from date'2008-08-02') as m1 from dual;

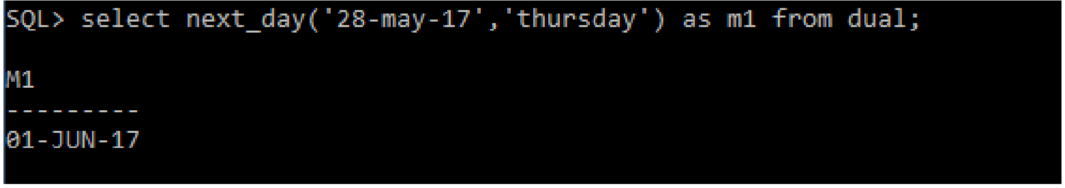


```
SQL> select extract(year from date'2008-08-02') as m1 from dual;

        M1
----------
      2008
```

6. next day:

next_day(date,dayname)

select next_day('28-may-17','thursday') as m1 from dual;



```
SQL> select next_day('28-may-17','thursday') as m1 from dual;

M1
---------
01-JUN-17
```

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).

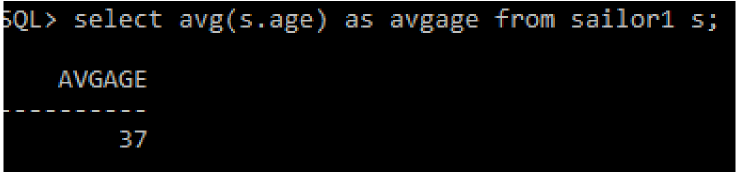
6) COUNT(*):It returns the number of all rows in the given table(including null values).

1) Find the average age of all sailors.

Query

select avg(s.age) as avgage from sailor1 s;

Output



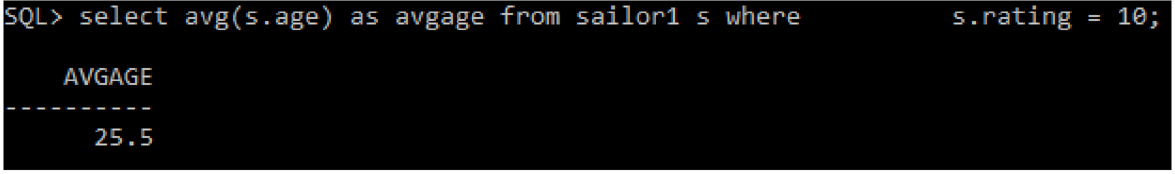
```
SQL> select avg(s.age) as avgage from sailor1 s;

    AVGAGE
----------
        37
```

2) Find the average age of all sailors with rating of 10.

Query

select avg(s.age) as avgage from sailor1 s where s.rating = 10;

Output



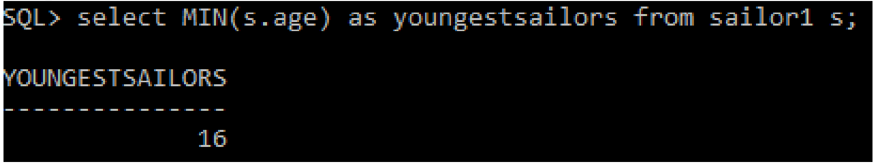
```
SQL> select avg(s.age) as avgage from sailor1 s where         s.rating = 10;

    AVGAGE
----------
      25.5
```

3) Find the age of youngest sailors

Query

select MIN(s.age) as youngestsailors from sailor1 s;

Output



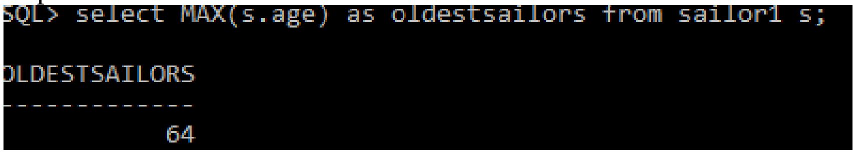
```
SQL> select MIN(s.age) as youngestsailors from sailor1 s;

YOUNGESTSAILORS
---------------
             16
```

4) Find the age of oldest sailors.

Query

select MAX(s.age) as oldestsailors from sailor1 s;

Output



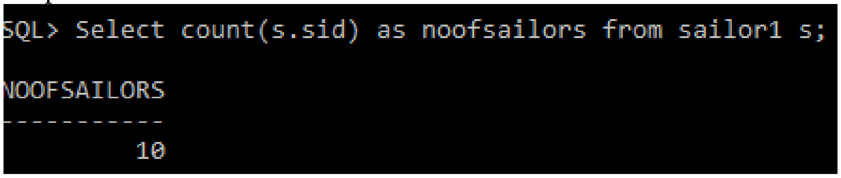
```
SQL> select MAX(s.age) as oldestsailors from sailor1 s;

OLDESTSAILORS
-------------
           64
```

5) Find the total number of sailors.

Query

Select count(s.sid) as noofsailors from sailor1 s;

Output



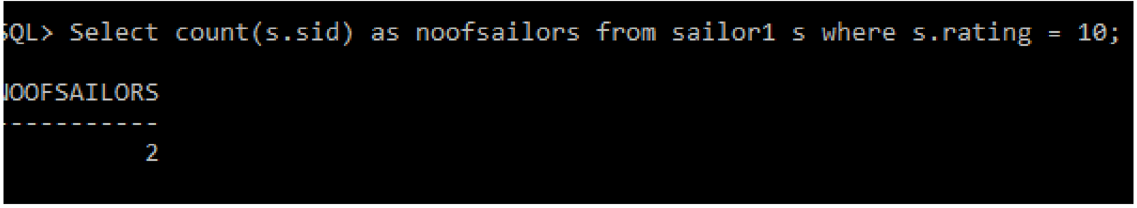
```
SQL> Select count(s.sid) as noofsailors from sailor1 s;

NOOFSAILORS
-----------
         10
```

6) Find the number of sailors with rating 10.

Query

Select count(s.sid) as noofsailors from sailor1 s where s.rating = 10;

Output



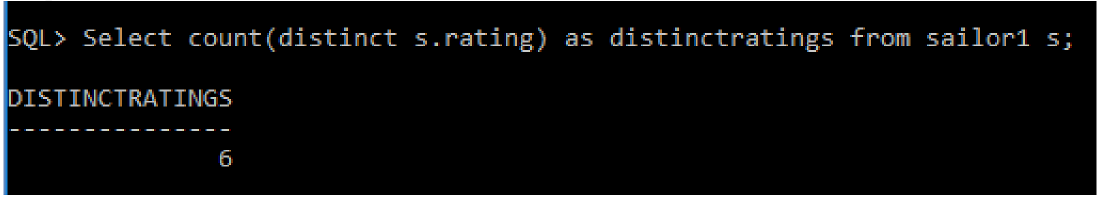
```
SQL> Select count(s.sid) as noofsailors from sailor1 s where s.rating = 10;

NOOFSAILORS
-----------
          2
```

7) Find the count of distinct ratings.

Query

Select count(distinct s.rating) as distinctratings from sailor1 s;

Output



```
SQL> Select count(distinct s.rating) as distinctratings from sailor1 s;

DISTINCTRATINGS
---------------
              6
```

VIVA QUESTIONS

1) What is difference between count() and count(*).
2) List out Aggregate functions.
3) List the single line functions.

EXERCISE: 4

AIM: Execute Set operations on various Relations.

Description:

Set operations 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.

1. Find the names of sailors who have reserved red or green boat.

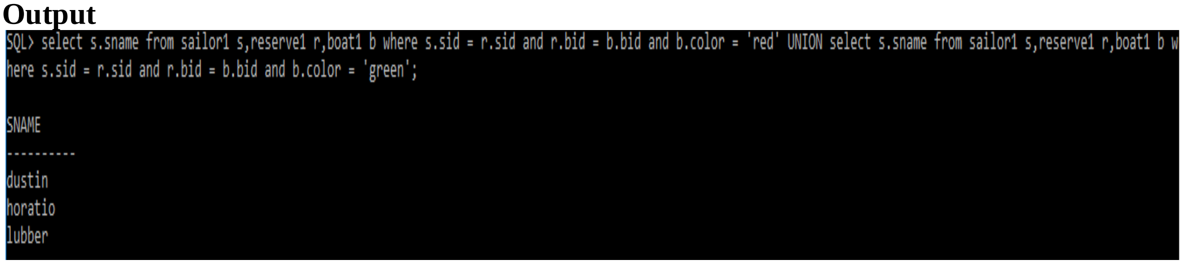
Query

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

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



```
SQL> 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 w
here s.sid = r.sid and r.bid = b.bid and b.color = 'green';

SNAME
---------
dustin
horatio
lubber
```

2. Find all sid's of sailors who have rating of 10 or reserved boat no.104.

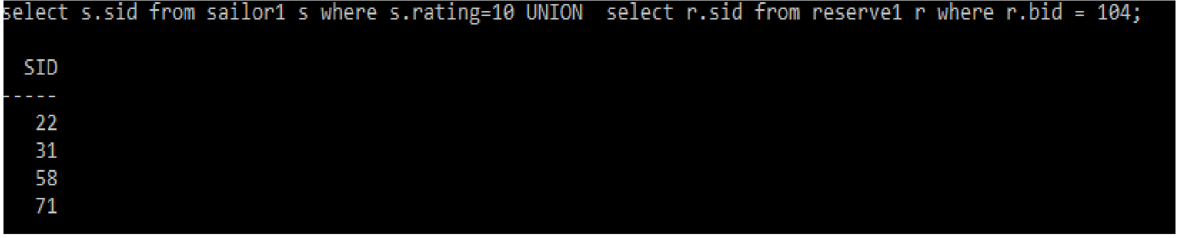
Query

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

UNION

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

Output



```
select s.sid from sailor1 s where s.rating=10 UNION  select r.sid from reserve1 r where r.bid = 104;

  SID
-----
   22
   31
   58
   71
```

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.

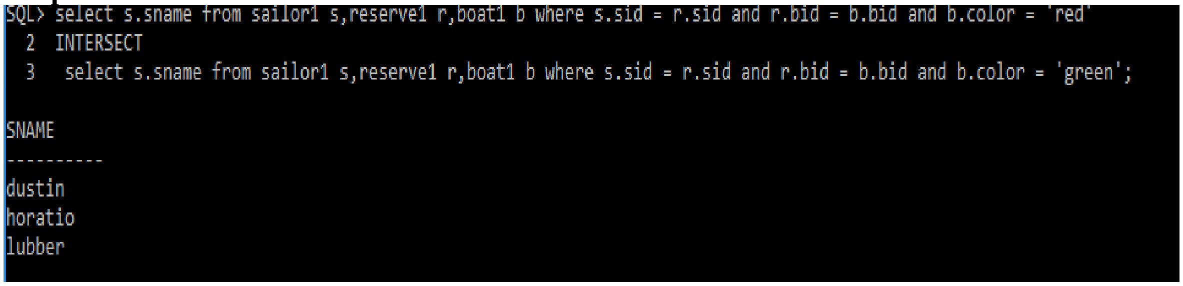
Query

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



```
SQL> select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'
  2  INTERSECT
  3   select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'green';

SNAME
----------
dustin
horatio
lubber
```

MINUS

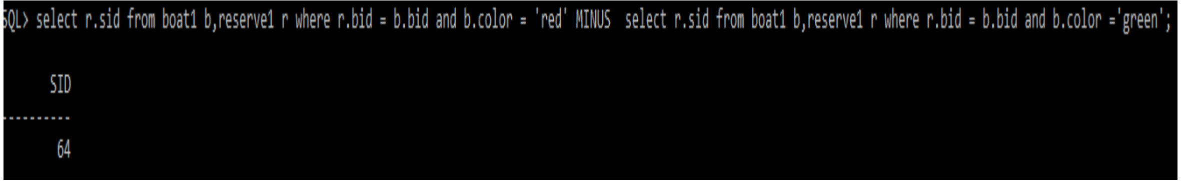
Let R and S are two union compatible relations then, intersect operation returns the tuples that are present in R but not in S.

1. Find the sid’s of sailors who have reserved red but not green boat.

Query

select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color = 'red' MINUS select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color ='green';

Output



```
SQL> select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color = 'red' MINUS  select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color ='green';

       SID
----------
        64
```

<u>VIVA QUESTIONS</u>

1) List out the SET operations.
2) What is Union?
3) What is Intersection and Minus?

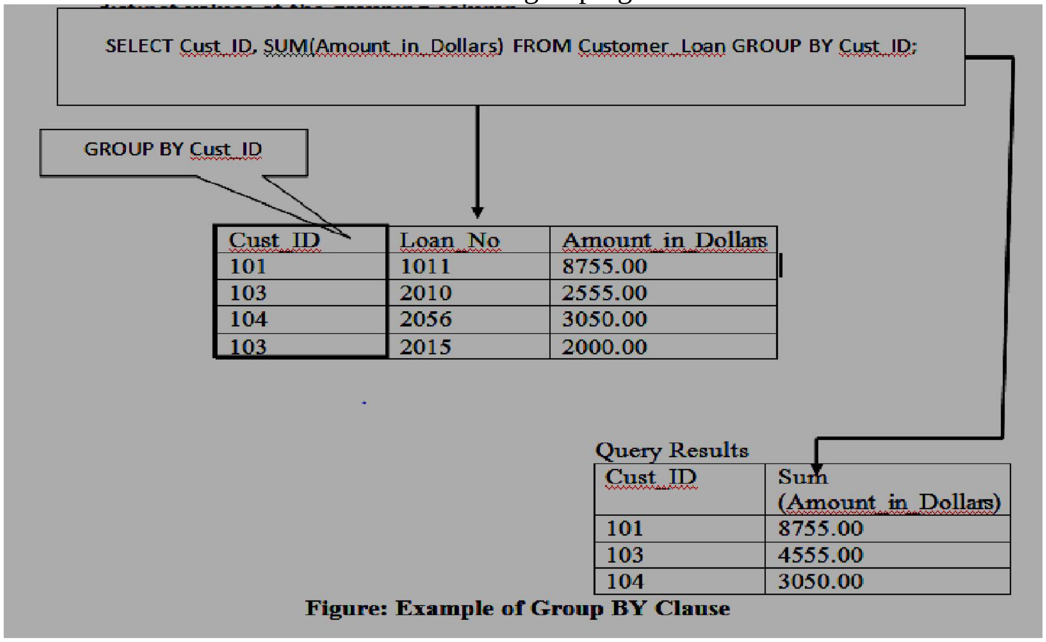
EXERCISE: 5

AIM: Execute Group by, Order by clause on Relations.

DESCRIPTION

GROUP BY

- 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.
- Example: consider the Customer_Loan table that has data about all the loans taken by all the customers of the bank. Assume that we want to retrieve the total loan-amount of all loans taken by each customer.
- Related rows can be grouped together by the GROUP BY clause by specifying a column as a grouping column.
- In the below example, the Cust_ID will be the 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.



```
SELECT Cust_ID, SUM(Amount_in_Dollars) FROM Customer_Loan GROUP BY Cust_ID;
```

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

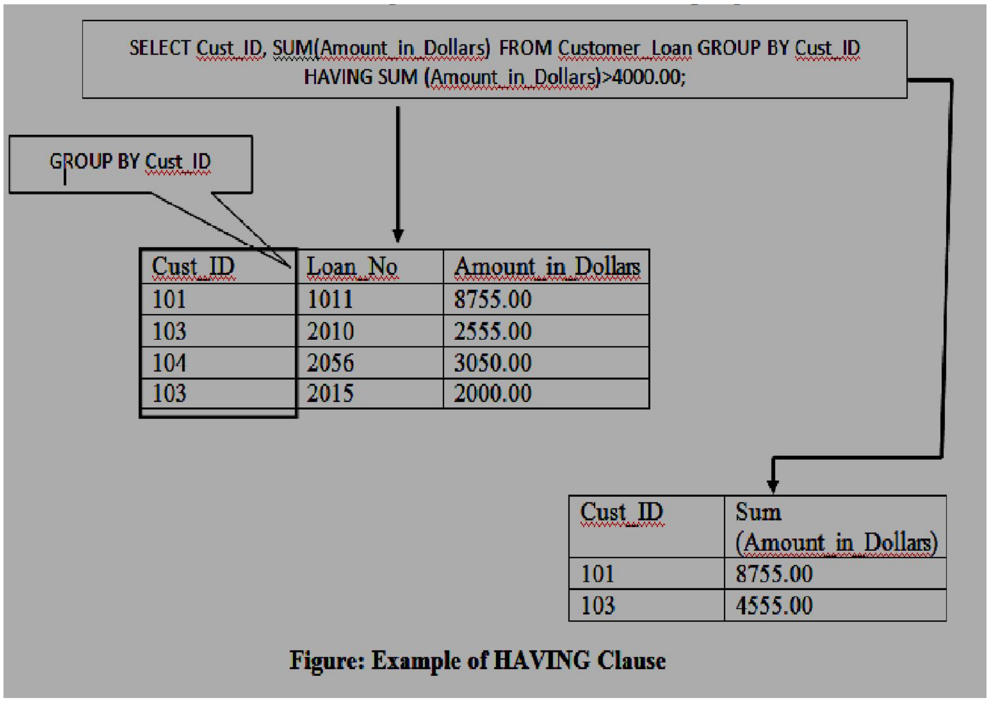
Query Results

Cust_ID	Sum (Amount_in_Dollars)
101	8755.00
103	4555.00
104	3050.00

Figure: Example of Group BY Clause

HAVING

- 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.



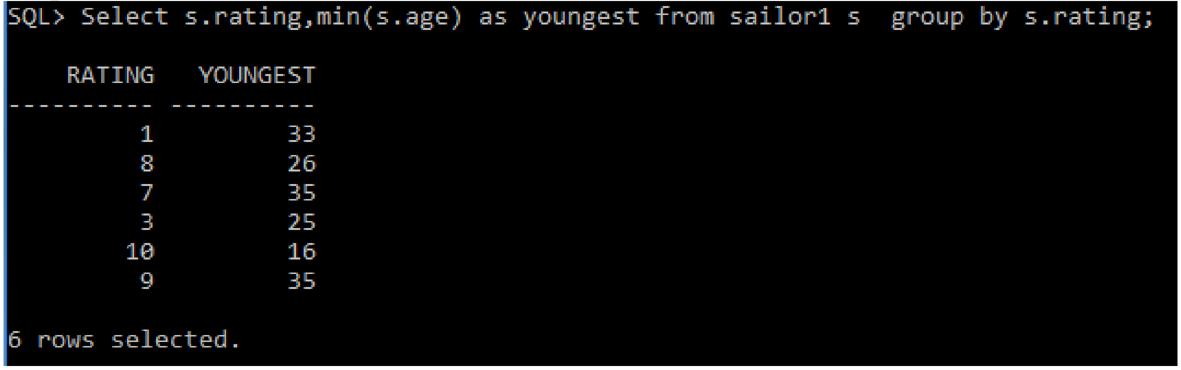
```
SELECT Cust_ID, SUM(Amount_in_Dollars) FROM Customer_Loan GROUP BY Cust_ID
HAVING SUM (Amount_in_Dollars)>4000.00;
```

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Cust_ID	Sum (Amount_in_Dollars)
101	8755.00
103	4555.00

Figure: Example of HAVING Clause

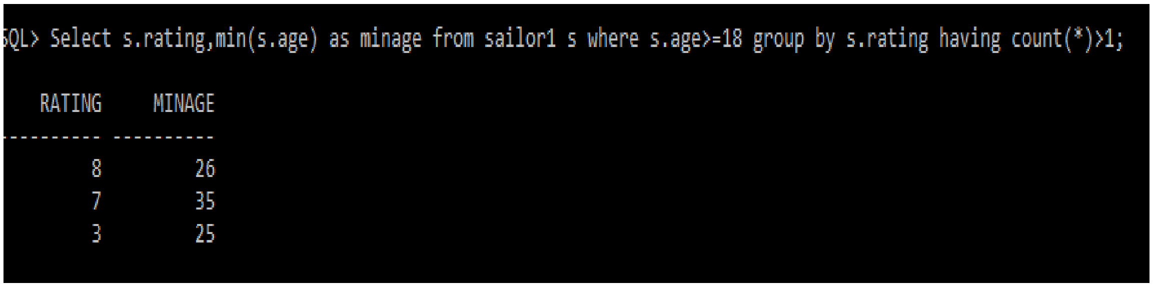
1) Find the age of the youngest sailor for each rating level.
Query
Select s.rating,min(s.age) as youngest from sailor1 s group by s.rating;
Output:



```
SQL> Select s.rating,min(s.age) as youngest from sailor1 s  group by s.rating;

    RATING   YOUNGEST
---------- ----------
         1         33
         8         26
         7         35
         3         25
        10         16
         9         35

6 rows selected.
```

2) Find the age of the youngest sailor who is eligible to vote(i.eatleast 18 years old) for each rating level with atleast two such sailors.
Query
Select s.rating,min(s.age) as minage from sailor1 s where s.age>=18 group by s.rating having count(*)>1;

Output



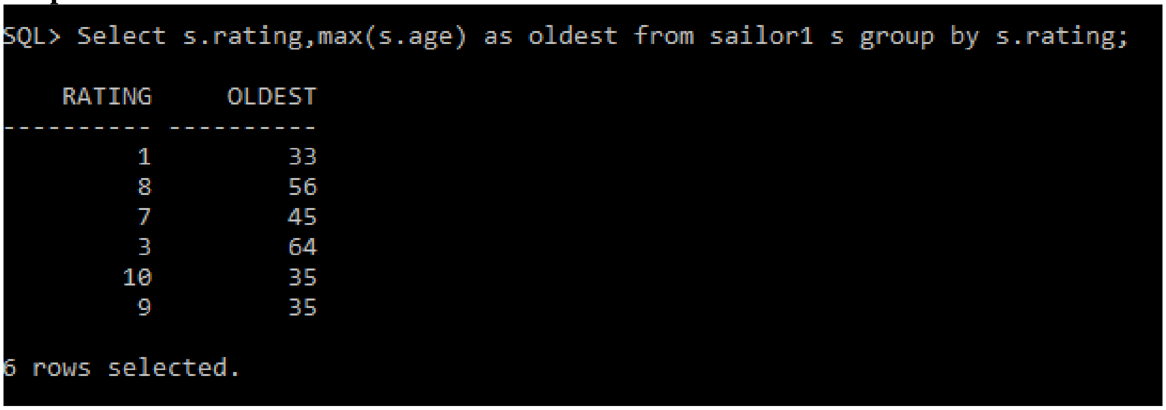
```
SQL> Select s.rating,min(s.age) as minage from sailor1 s where s.age>=18 group by s.rating having count(*)>1;

   RATING     MINAGE
--------- ---------
        8         26
        7         35
        3         25
```

3) **Find the age of the oldest sailor for each rating level.**

Query

Select s.rating,max(s.age) as oldest from sailor1 s group by s.rating;

Output



```
SQL> Select s.rating,max(s.age) as oldest from sailor1 s group by s.rating;

    RATING     OLDEST
---------- ----------
         1         33
         8         56
         7         45
         3         64
        10         35
         9         35

6 rows selected.
```

ORDER BY

The ORDER BY statement in sql is used to sort the fetched data in either ascending or descending according to one or more columns.

- By default ORDER BY sorts the data in ascending order.
- We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

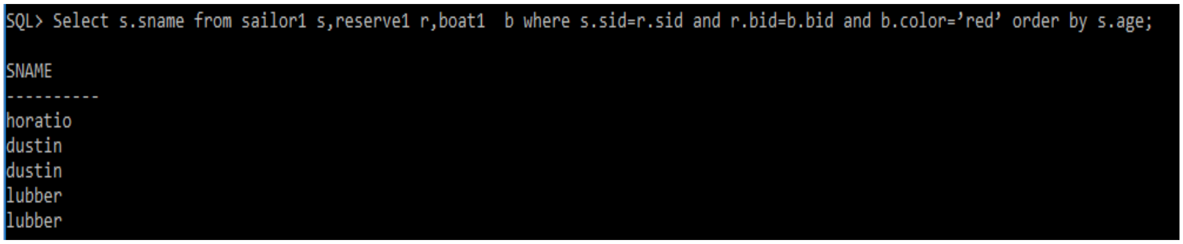
Syntax of all ways of using ORDER BY is shown below:

- Sort according to one column: To sort in ascending or descending order we can use the keywords ASC or DESC respectively.
  Syntax:
  SELECT * FROM table_name ORDER BY column_name ASC|DESC;
- Sort according to multiple columns: To sort in ascending or descending order we can use the keywords ASC or DESC respectively. To sort according to multiple columns, separate the names of columns by (,) operator.
  Syntax:
- SELECT * FROM table_name ORDER BY column1 ASC|DESC , column2 ASC|DESC;

1) Find the names of sailors who have reserved a red boat listing in order of age.

Select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid=r.sid and r.bid=b.bid and b.color=’red’ order by s.age; ((or)for descending s.age desc)

Output

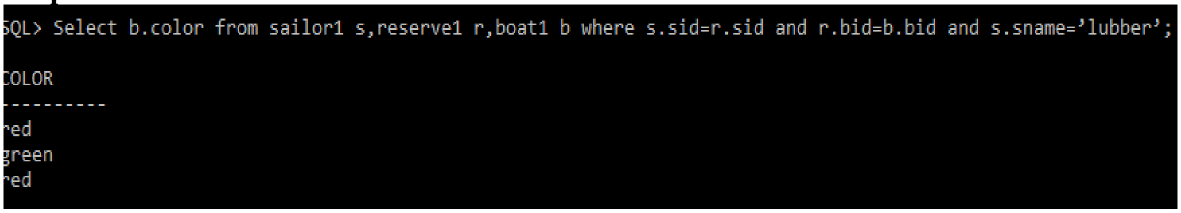


```
SQL> Select s.sname from sailor1 s,reserve1 r,boat1  b where s.sid=r.sid and r.bid=b.bid and b.color=’red’ order by s.age;

SNAME
---------
horatio
dustin
dustin
lubber
lubber
```

2) Find the colors of boats reserved by lubber.

Select b.color from sailor1 s,reserve1 r,boat1 b where s.sid=r.sid and r.bid=b.bid and s.sname=’lubber’;

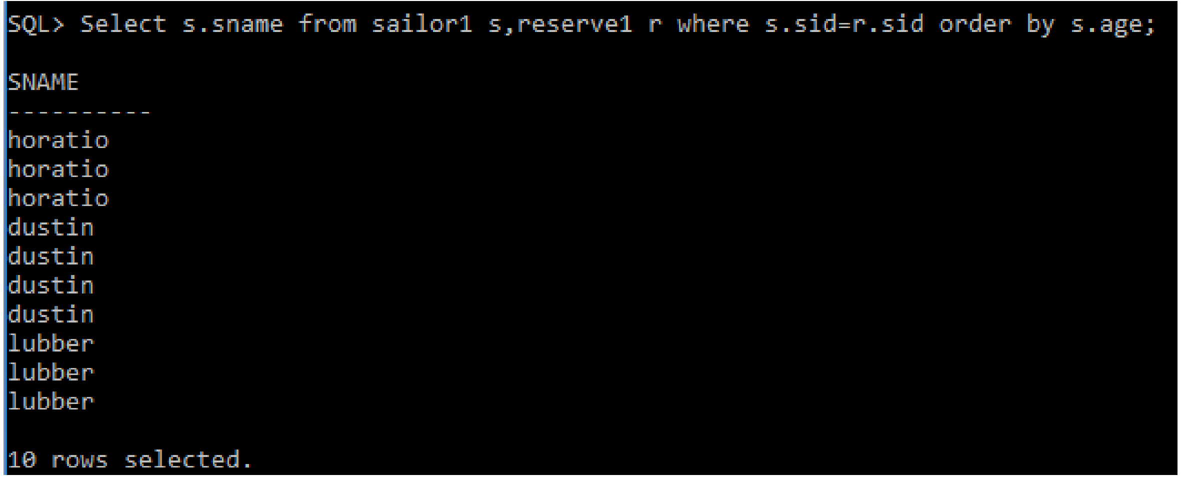
Output



```
SQL> Select b.color from sailor1 s,reserve1 r,boat1 b where s.sid=r.sid and r.bid=b.bid and s.sname=’lubber’;

COLOR
---------
red
green
red
```

3) Find the names of sailors who have reserved at least one boat in the order of age.

Select s.sname from sailor1 s,reserve1 r where s.sid=r.sid order by s.age;



```
SQL> Select s.sname from sailor1 s,reserve1 r where s.sid=r.sid order by s.age;

SNAME
----------
horatio
horatio
horatio
dustin
dustin
dustin
dustin
lubber
lubber
lubber

10 rows selected.
```

<u>VIVA QUESTIONS</u>

1. What is group by?
2. What is order by?
3. When we use order by command default output will be in which order?

EXERCISE: 6

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

DESCRIPTION

To execute queries and nested queries on sailors, boats, and reserves database.

Create table sailor1(sid number(10) primary key, sname varchar2(10),rating number(10),age number(8,2));

Output: Table created.

Insert into sailor1 values(22,’dustin’,7,45);

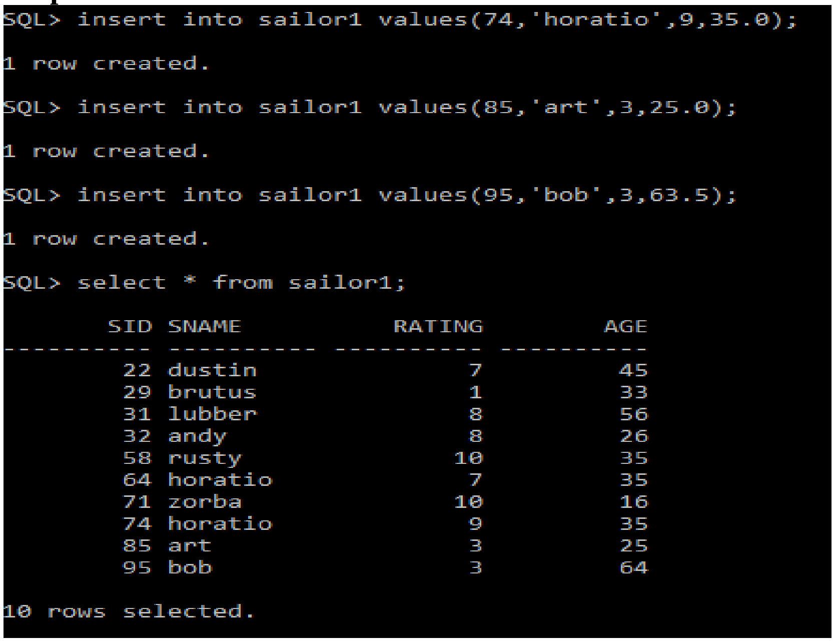
Output: 1 row created.

Insert into sailor1 values(29,’brutus’,1,133);

Output: 1 row created.

Select * from sailor1;

Output:



```
SQL> insert into sailor1 values(74,'horatio',9,35.0);

1 row created.

SQL> insert into sailor1 values(85,'art',3,25.0);

1 row created.

SQL> insert into sailor1 values(95,'bob',3,63.5);

1 row created.

SQL> select * from sailor1;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 dustin              7         45
        29 brutus              1         33
        31 lubber              8         56
        32 andy                8         26
        58 rusty              10         35
        64 horatio             7         35
        71 zorba              10         16
        74 horatio             9         35
        85 art                 3         25
        95 bob                 3         64

10 rows selected.
```

create table boat1(bid number(10) primary key,bname varchar2(10),color varchar2(10));

output: Table created.

insert into boat1 values(101,'interlake','blue');

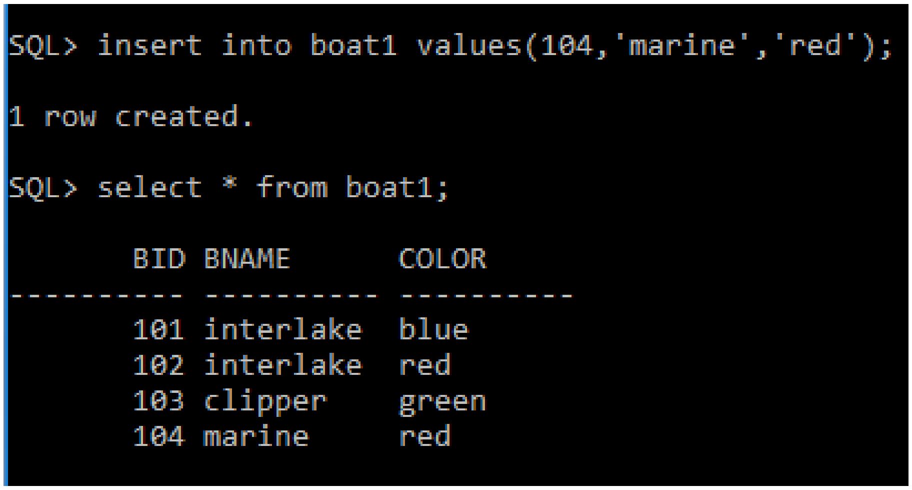
output: 1 row created.

SQL> insert into boat1 values(102,'interlake','red');

Output : 1 row created.

Select * from boat1;

Output



```
SQL> insert into boat1 values(104,'marine','red');

1 row created.

SQL> select * from boat1;

       BID BNAME      COLOR
---------- ---------- ----------
       101 interlake  blue
       102 interlake  red
       103 clipper    green
       104 marine     red
```

Create table reserve1 (sid number(10) references sailor1(sid),bid number(10) references boat1(bid),day date);

Output

TABLE CREATED.

insert into reserve1 values(22,101,'10-oct-98');

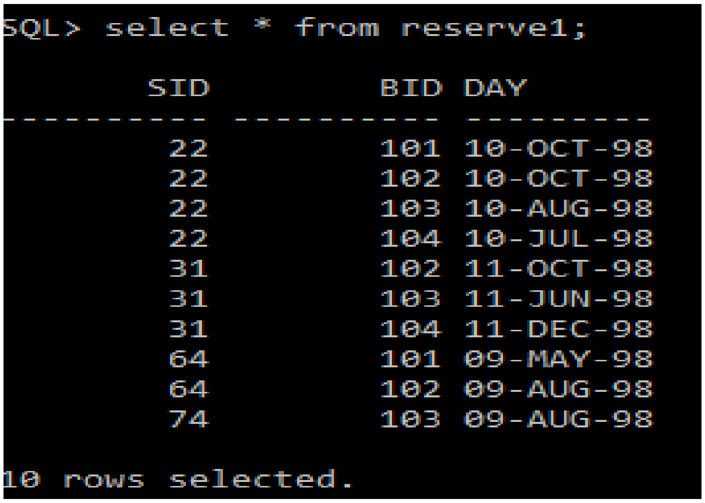
1 row created.

insert into reserve1 values(22,102,'10-oct-98');

1 row created.

insert into reserve1 values(22,103,'10-aug-98');

1   row created.



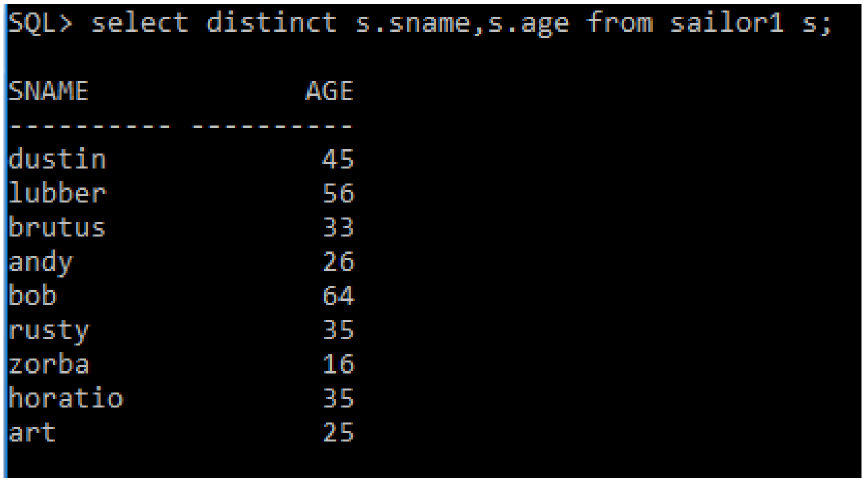
```
SQL> select * from reserve1;

       SID        BID DAY
---------- ---------- ---------
        22        101 10-OCT-98
        22        102 10-OCT-98
        22        103 10-AUG-98
        22        104 10-JUL-98
        31        102 11-OCT-98
        31        103 11-JUN-98
        31        104 11-DEC-98
        64        101 09-MAY-98
        64        102 09-AUG-98
        74        103 09-AUG-98

10 rows selected.
```

1) Find names and ages of all sailors

Query

Select distinct s.sname, s.age from sailor1 s;

Output:



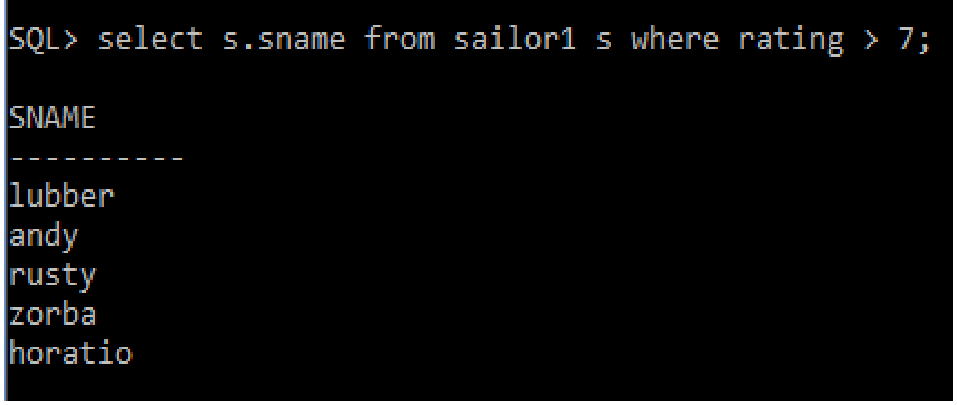
```
SQL> select distinct s.sname,s.age from sailor1 s;

SNAME             AGE
---------- ----------
dustin             45
lubber             56
brutus             33
andy               26
bob                64
rusty              35
zorba              16
horatio            35
art                25
```

2) Find all sailors with rating above 7.

Query:

Select s.sid from sailor1 s where rating>7;

Output:



```
SQL> select s.sname from sailor1 s where rating > 7;

SNAME
----------
lubber
andy
rusty
zorba
horatio
```

3) Find the names of sailors who have reserved boat no.103

Query:

Select s.sname from sailor1 s,reserve1 r where s.sid = r.sid and r.bid = 103;

4) find the sid’s of sailors who reserved a red boat.

Query:

Select s.sid from reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = ‘red’;

5) Find the names of sailors who reserved a red boat.

Query:

Select s.sname from sialor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = ‘red’;

6) find the colors of boat reserved by lubber.

Query

Select b.color from sailor1 s,boat1 b,reserve1 s where s.sid r.sid and r.bid = b.bid and s.sname = ‘lubber’;

7) find the names of sailors who have reserved atleast one boat.

Query:

Select s.sname from sialor1 s, reserve1 r where s.sid = r.sid;

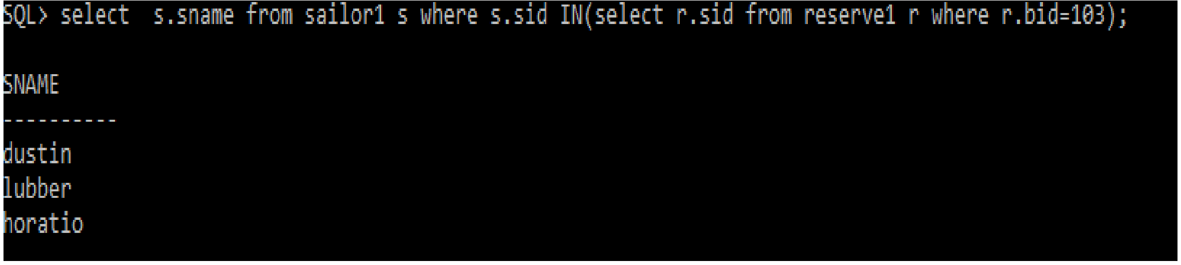
NESTED QUERIES

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.

1. Find the names of sailors who have reserved boat103

SQL> select s.sname from sailor1 s where s.sid IN(select r.sid from reserve1 r where r.bid=103);

Output:

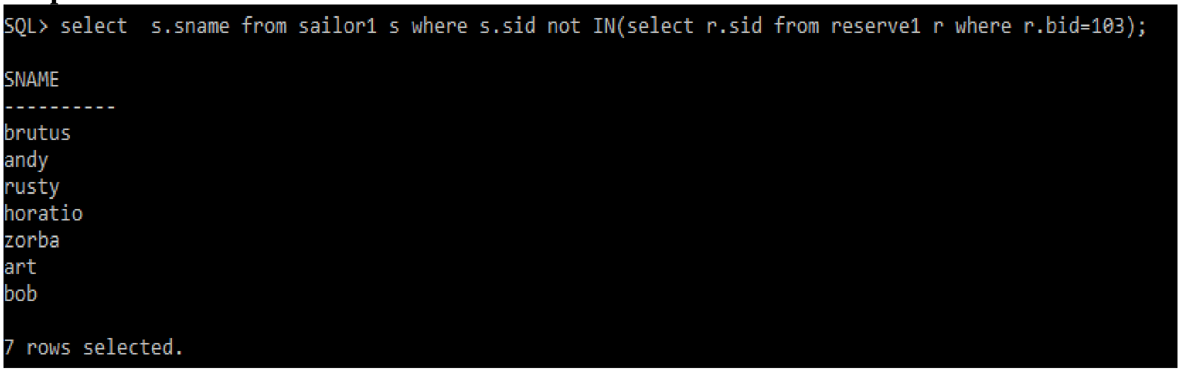


```
SQL> select  s.sname from sailor1 s where s.sid IN(select r.sid from reserve1 r where r.bid=103);

SNAME
----------
dustin
lubber
horatio
```

2. Find the names of sailors who have not reserved boat103

SQL> select s.sname from sailor1 s where s.sid not IN(select r.sid from reserve1 r where r.bid=103);

Output:

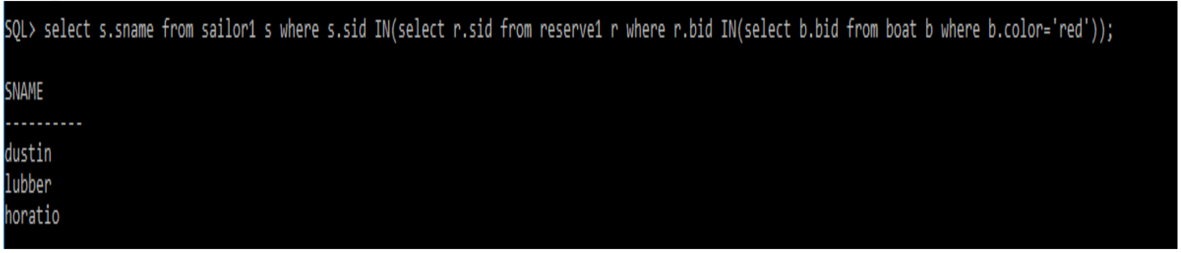


```
SQL> select  s.sname from sailor1 s where s.sid not IN(select r.sid from reserve1 r where r.bid=103);

SNAME
----------
brutus
andy
rusty
horatio
zorba
art
bob

7 rows selected.
```

3. Find the names of sailors who have reserved a red boat

SQL> select s.sname from sailor1 s where s.sid IN(select r.sid from reserve1 r where r.bid IN(select b.bid from boat b where b.color='red'));

Output:

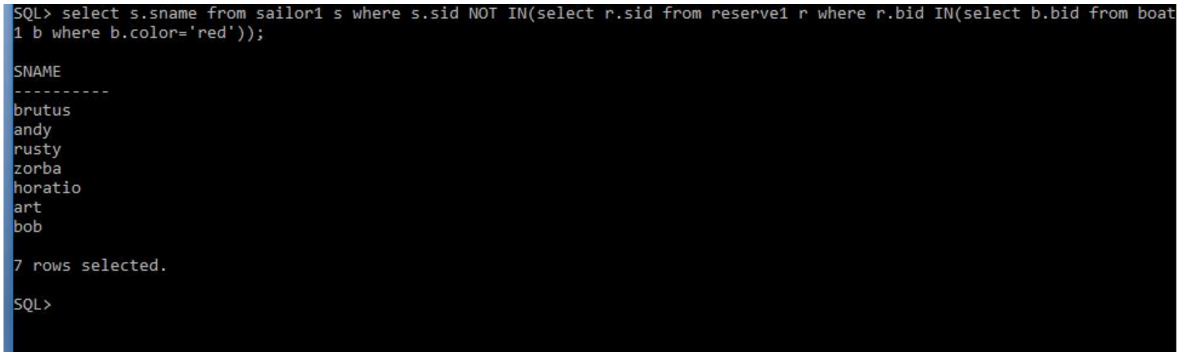


```
SQL> select s.sname from sailor1 s where s.sid IN(select r.sid from reserve1 r where r.bid IN(select b.bid from boat b where b.color='red'));

SNAME
---------
dustin
lubber
horatio
```

4. Find the names of sailors who have not reserved a red boat

SQL> select s.sname from sailor1 s where s.sid NOT IN(select r.sid from reserve1 r where r.bid IN(select b.bid from boat1 b where b.color='red'));

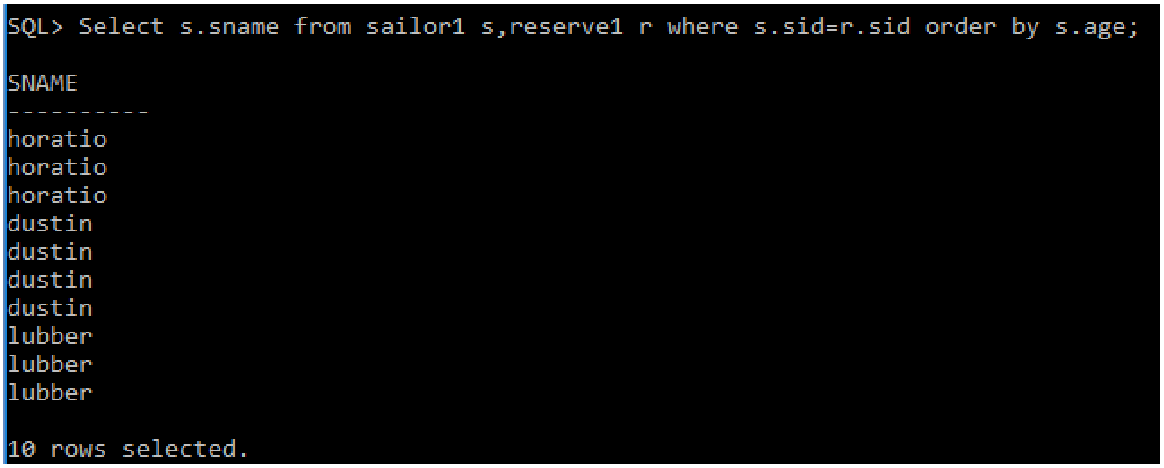
Output:



```
SQL> select s.sname from sailor1 s where s.sid NOT IN(select r.sid from reserve1 r where r.bid IN(select b.bid from boat
1 b where b.color='red'));

SNAME
----------
brutus
andy
rusty
zorba
horatio
art
bob

7 rows selected.

SQL>
```

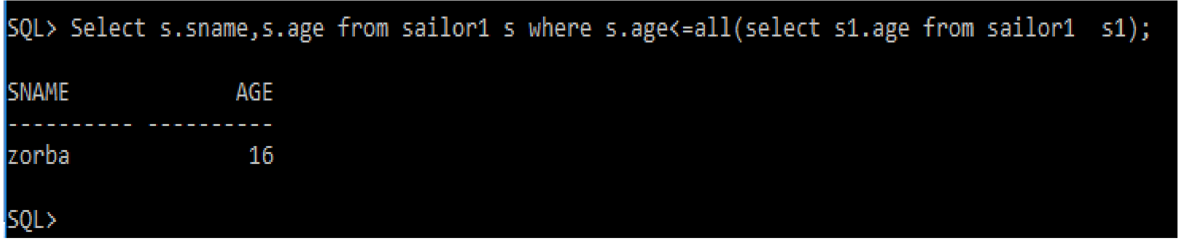
Co-Related nested queries: Correlated subquery is a query in which the inner query is executed for each row of the outer query.

1) Find the names of sailors who have reserved boat no 103.
Select s.sname from sailor1 s where exists(select * from reserve1 r where s.sid=r.sid and r.bid=103);



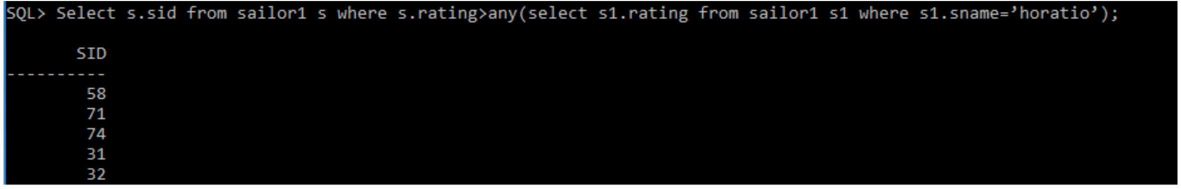
```
SQL> Select s.sname from sailor1 s,reserve1 r where s.sid=r.sid order by s.age;

SNAME
----------
horatio
horatio
horatio
dustin
dustin
dustin
dustin
lubber
lubber
lubber

10 rows selected.
```

2) Find the names and ages of youngest sailor.
Select s.sname,s.age from sailor1 s where s.age<=all(select s1.age from sailor1 s1);



```
SQL> Select s.sname,s.age from sailor1 s where s.age<=all(select s1.age from sailor1  s1);

SNAME             AGE
---------- ----------
zorba              16

SQL>
```

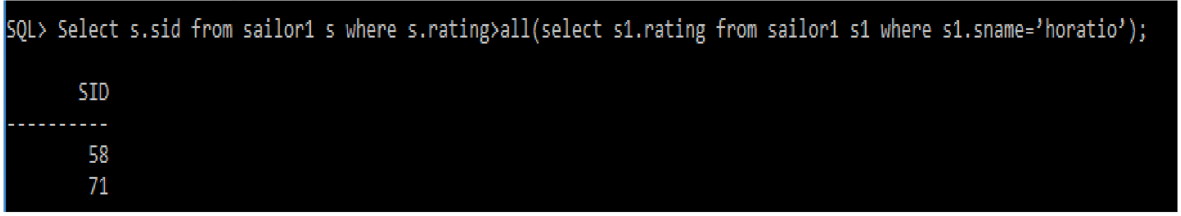
3) Find the sailors whose rating is better than some sailors called horatio.
Select s.sid from sailor1 s where s.rating>any(select s1.rating from sailor1 s1 where s1.sname=’horatio’);



```
SQL> Select s.sid from sailor1 s where s.rating>any(select s1.rating from sailor1 s1 where s1.sname=’horatio’);

       SID
----------
        58
        71
        74
        31
        32
```

4) Find the sailors whose rating is better than every sailor called horatio.

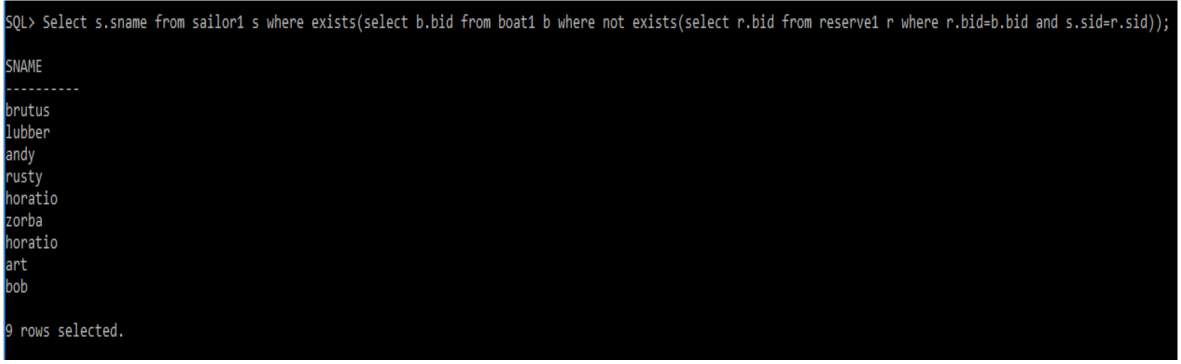
Select s.sid from sailor1 s where s.rating>all(select s1.rating from sailor1 s1 where s1.sname='horatio');



```
SQL> Select s.sid from sailor1 s where s.rating>all(select s1.rating from sailor1 s1 where s1.sname='horatio');

       SID
----------
        58
        71
```

5) Find the names who reserved all boats

Select s.sname from sailor1 s where exists(select b.bid from boat1 b where not exists(select r.bid from reserve1 r where r.bid=b.bid and s.sid=r.sid));



```
SQL> Select s.sname from sailor1 s where exists(select b.bid from boat1 b where not exists(select r.bid from reserve1 r where r.bid=b.bid and s.sid=r.sid));

SNAME
----------
brutus
lubber
andy
rusty
horatio
zorba
horatio
art
bob

9 rows selected.
```

Employee table

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

1. Find the employees whose salary is greater than the average salary of their department.

SELECT *
FROM Employee1 E1
WHERE Salary > (SELECT AVG(Salary)
FROM Employee1 E2
WHERE E1.Dept = E2.Dept);

E1

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE

x

E2

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE

104	Scott	30000	ECE
105	Warner	50000	CSE

104	Scott	30000	ECE
105	Warner	50000	CSE

20000 > 33333 F

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

40000 > 35000 T

102	Smith	40000	ECE

Included in the result

E1

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

x

E2

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

30000 > 33333 F

E1

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

x

E2

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

30000 > 35000 F

E1

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

E2

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	30000	ECE
105	Warner	50000	CSE

50000 > 33333 T

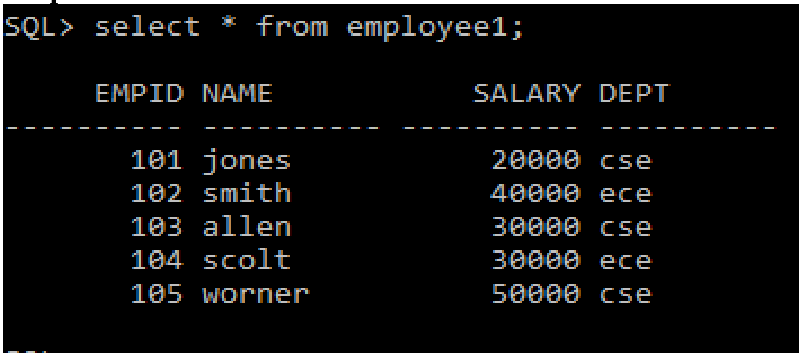
105	Warner	50000	CSE

Included in the result

Final Result

Empid	Name	Salary	Dept
102	Smith	40000	ECE
105	Warner	50000	CSE

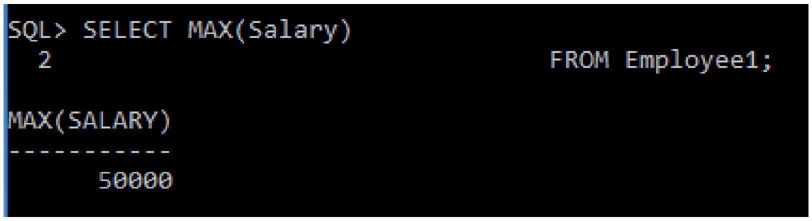
Output



```
SQL> select * from employee1;

     EMPID NAME                     SALARY DEPT
---------- ---------- ---------- ----------
       101 jones                     20000 cse
       102 smith                     40000 ece
       103 allen                     30000 cse
       104 scolt                     30000 ece
       105 worner                    50000 cse
```

2. Find the 1st highest salary employee details in the Employee table.

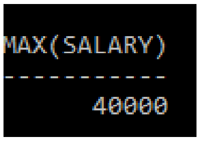
SELECT MAX(Salary)
FROM Employee1;



```
SQL> SELECT MAX(Salary)
  2                             FROM Employee1;

MAX(SALARY)
-----------
      50000
```

3. Find the 2nd highest salary employee details in the Employee table.

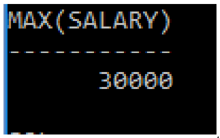
SELECT MAX(Salary)
FROM Employee1
WHERE Salary < (SELECT MAX(Salary)
From Employee1);



```
MAX(SALARY)
-----------
      40000
```

4. Find the 3rd highest salary employee details in the Employee table.

```
 SELECT MAX(Salary)
                         FROM Employee1
                         WHERE Salary < (SELECT MAX(Salary)
                                         From Employee1
                                         WHERE Salary < (SELECT MAX(Salary)
                                                         FROM Employee1));
```

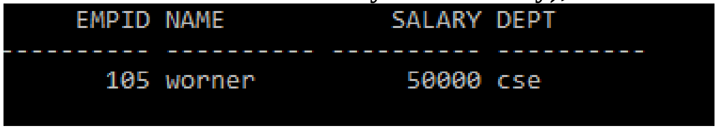


```
MAX(SALARY)
-----------
      30000
```

So here, to find 1st Highest salary - we are writing 1 Query
to find 2nd Highest salary - we are writing 2 Queries
to find 3rd Highest salary - we are writing 3 Queries
......
to find Nth Highest salary - we need to write N Queries
Writing these many queries is tedious and inefficient.
One best solution to this problem is correlated sub query.

5. Find the 1st highest salary employee details in the Employee table.

```
SELECT *
FROM Employee1 E1
WHERE 0 = (SELECT COUNT(Salary)
                  FROM Employee1 E2
              WHERE E2. Salary > E1. Salary);
```



```
     EMPID NAME           SALARY DEPT
---------- ---------- ---------- ----------
       105 worner          50000 cse
```

E2

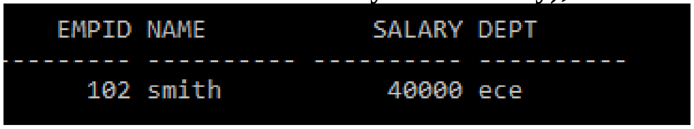
Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	35000	ECE
105	Warner	50000	CSE

E1

Empid	Name	Salary	Dept
101	Jones	20000	CSE
102	Smith	40000	ECE
103	Allen	30000	CSE
104	Scott	35000	ECE
105	Warner	50000	CSE

6. Find the 2nd highest salary employee details in the Employee table.

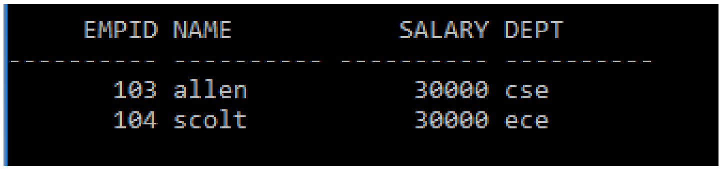
```
SELECT *
FROM Employee1 E1
WHERE 1 = (SELECT COUNT(Salary)
                   FROM Employee1 E2
              WHERE E2. Salary > E1. Salary);
```



```
     EMPID NAME                 SALARY DEPT
---------- ---------- ---------- ----------
       102 smith                 40000 ece
```

7. Find the 3rd highest salary employee details in the Employee table.

```
SELECT *
FROM Employee1 E1
WHERE 2 = (SELECT COUNT(Salary)
                   FROM Employee1 E2
              WHERE E2. Salary > E1. Salary);
```



```
     EMPID NAME                 SALARY DEPT
---------- ---------- ---------- ----------
       103 allen                 30000 cse
       104 scolt                 30000 ece
```

<u>VIVA QUESTIONS</u>

1. What is nested query?
2. What is co-related nested query?
3. How to find second highest salary?

EXERCISE: 7

AIM: Perform the following join operations

a. Cross b. Inner c. Outer (left, right, full) d. Self

DESCRIPTION:

JOIN: A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.

Example:

Student table

Sid	name	age
101	nihal	19
102	teja	18
103	rama	15
104	sita	16
105	siva	21

Enroll table

Sid	cid
101	CS1201
103	CS1203
105	CS1101

1. Natural join

Select * from student natural join enrol;

Output:

Sid	name	age	cid
101	nihal	19	CS1201
103	rama	15	CS1203
105	siva	21	CS1101

2. Equi join

It is a simple sql join condition which uses the eual sign as the comparison operator.it is divided into 2 types.

1) sql inner join

2) sql outer join

1) inner join: in this all rows returned by the sql query satisfy the sql conditions specified.

Example:

Select * from student inner join 44nrol on student.sid=enrol.sid;

3) sql outer join:

This sql join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables.

These are 3 types

a) Left outer join
b) Right outer join
c) Full outer join

Left outer join: in this we will get values of left side table along with the matching values of right side table.
Example: select * form student s left outer join enrol e on s.sid=e.sid;
Output:

Sid	Name	Age	Sid	cid
101	Nihal	19	101	CS1201
103	Rama	15	103	CS1203
105	Siva	21	105	CS1101
102	Teja	18		
104	Sita	16		

Right outer join:
In this we will get all values of right side table along with the matching values of left side table.
Example: select * from student s right outer join enroll e on s.sid=e.sid;
Output:

Sid	Name	Age	Sid	cid
101	Nihal	19	101	CS1201
103	Rama	15	103	CS1203
105	Siva	21	105	CS1101

Full outer join:
This type of join returns all rows from the left hand table and right hand table with null in place where the join condition is not met.
Example:
Select * from student s full outer join enrol e on s.sid=e.sid;

Output:

Sid	**Name**	**Age**	**Sid**	**cid**
101	Nihal	19	101	CS1201
103	Rama	15	103	CS1203
105	Siva	21	105	CS1101
102	Teja	18		
104	sita	16		

Theta(Ɵ) join:
Theta join is a conditional join that takes on two tables.
Example: select * from student s,enroll e where s.sid=e.sid;
Output:

Sid	**Name**	**Age**	**Sid**	**cid**

101	Nihal	19	101	CS1201
103	Rama	15	103	CS1203
105	Siva	21	105	CS1101

Example 2: select * from student s, enroll e where s.sid>e.sid;
Output:
Same query we can apply these symbols >=,<=,<>.

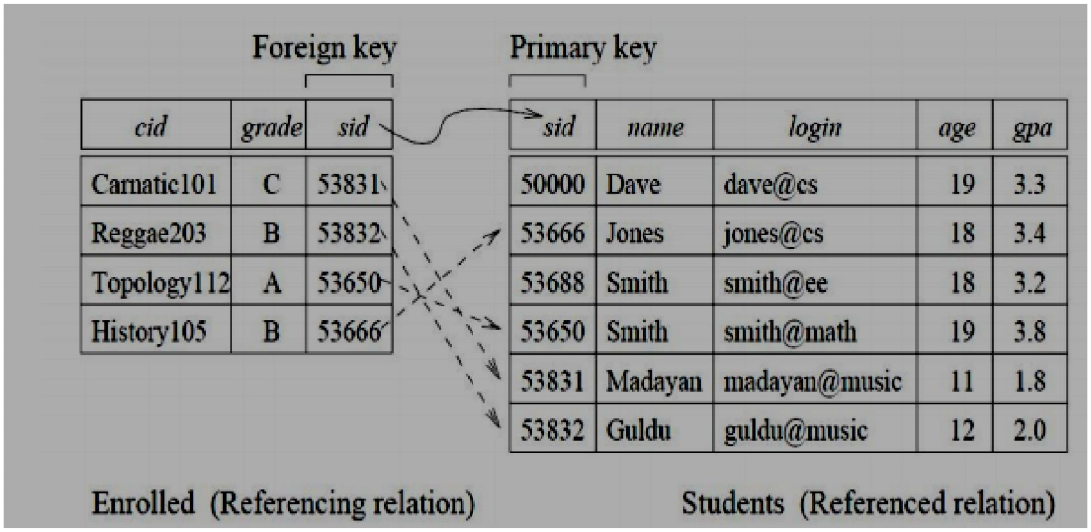
Cartesian product(X) or Cross product:
RXS return a relation instance whose schema contains all the fields of R followed by all fields of S.
Example:
Select * from student, enroll;
Output:

Sid	Name	Age	Sid	cid
101	Nihal	19	101	CS1201
103	Rama	15	103	CS1203
105	Siva	21	105	CS1101
102	Teja	18		
104	sita	16		

Viva Questions

1. What is join?
2. What are the different types of outer joins?
3. What is **ϴ** join?
4. Explain cross product.

EXERCISE: 8
AIM: Creating Views.
DESCRIPTION:
A view is a table whose rows are not explicitly stored in the database but are computed as needed from a view definition. Consider the Students and Enrolled relations.

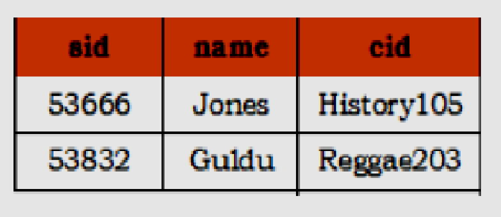


cid	grade	sid
Carnatic101	C	53831
Reggae203	B	53832
Topology112	A	53650
History105	B	53666

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Enrolled (Referencing relation) Students (Referenced relation)

From these two tables, we can create a view called BStudents showing B grade students as follows;
CREATE VIEW BStudents AS SELECT S.sid, S.name, E.cid FROM Students S, Enrolled E WHERE S.sid = E.sid AND E.grade = 'B';
This would produce the following view;
Output:
Select * from BStudents;

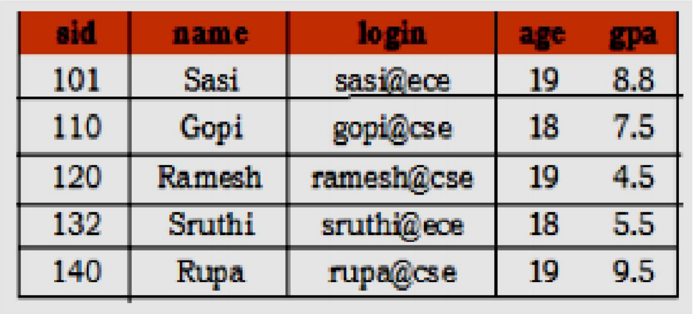


sid	name	cid
53666	Jones	History105
53832	Guldu	Reggae203

Syntax:
CREATE VIEW view name
AS SELECT attribute list
FROM table(s)
WHERE condition(s)

Updatable Views:
The SQL-92 standard allows updates to be specified only on views that are defined on a single base table using just selection and projection, with no use of aggregate operations and distinct clause. Such views are called updatable views.

For example, consider the following Students table;



sid	name	login	age	gpa
101	Sasi	sasi@ece	19	8.8
110	Gopi	gopi@cse	18	7.5
120	Ramesh	ramesh@cse	19	4.5
132	Sruthi	sruthi@ece	18	5.5
140	Rupa	rupa@cse	19	9.5

Consider the following view:
CREATE VIEW GoodStudents (sid, gpa)
AS SELECT sid, gpa
FROM Students
WHERE gpa >= 6.0;
Output: select * from GoodStudents;

sid	gpa
101	8.8
110	7.5
140	9.5

Example1:
first create sailors table or any other table. Create different views on that table.

1. Create view names as select s.name from sailors s;
   View created
   Select * from names;
2. Create view keerthi as select s.age from sailors s;
   View created
   Select * from keerthi;
3. Create view sony as select s.name,s.age from sailors s where s.age=35;
   View created
   Select * from sony;
4. Create view sam as select s.name, s.sid from sailors s where s.rating=10;
   View created
   Select * from sam;

Example2:
Create table student(sid number(10) primary key,name varchar2(20),gpa number(10),age number(10));
Table created.
Insert data
Create table enroll(sid number(20)references student(sid),eid number(20));
Table created.
Insert some rows.

1. Create view bstudent as select s.sname, s.sid from student where s.gpa=9.8;
   View created

Select * from bstudent;

2. Create view M as select s.sid,s.name,e.sid from student s,enrol e where s.sid=e.sid and e.grade=’A’;
   View created
   Select * from M;
3. Create view K as select s.name,s.sid from student s,enrol e where s.sid=e.sid and s.age=21;
   View created
   Select * from K;

Drop views:
Syntax: Drop view view_name;

VIVA QUESTIONS

1. What is view?
2. Is view updatable?
3. What are the advantages of views?

EXERCISE: 9

AIM:

Write PL/SQL basic programs.

DESCRIPTION:

PL/SQL stands for “Procedural Language extensions to the Structured Query Language”. SQL is a popular language for both querying and updating data in the relational database management systems (RDBMS). PL/SQL adds many procedural constructs to SQL language to overcome some limitations of SQL. Besides, PL/SQL provides a more comprehensive programming language solution for building mission-critical applications on Oracle Databases.

PL/SQL is a highly structured and readable language. Its constructs express the intent of the code clearly. Also, PL/SQL is a straightforward language to learn.

PL/SQL is a standard and portable language for Oracle Database development. If you develop a program that executes on an Oracle Database, you can quickly move it to another compatible Oracle Database without any changes.

```
SQL> set serveroutput on;
```

1. Aim:-Sum of two numbers

Sourcecode:-

```
SQL> declare
                x integer;
                y integer;
                z integer;
        begin
                x:=10;
                y:=20;
                z:=x+y;
         dbms_output.put_line('sum is' ||Z);
        end;
        /
```

Output:-

```
sum is30

PL/SQL procedure successfully completed.
```

2. Aim:-Sum of two numbers reading input from user

Sourcecode:-

```
SQL> declare
                x integer;
                y integer;
                z integer;
        begin
                x:=&x;
                y:=&y;
                z:=x+y;
        dbms_output.put_line(x||'+'||y||'='||z);
        end;
```

```
          /
```

Output:-

```
Enter value for x: 2
old   6: x:=&x;
new   6: x:=2;

Enter value for y: 2
old   7: y:=&y;
new   7: y:=2;

2+2=4
PL/SQL procedure successfully completed.
```

3. Aim:-TO PRINT NATURAL NUMBERS
Sourcecode:-

```
SQL> declare
                    a integer;
          begin
                    for a in 10 .. 20 loop
                    dbms_output.put_line('value of a:'||a);
                    end loop;
                    end;
                    /
```

Output:-

```
value of a:10
value of a:11
value of a:12
value of a:13
value of a:14
value of a:15
value of a:16
value of a:17
value of a:18
value of a:19
value of a:20
PL/SQL procedure successfully completed.
```

4. AIM:-SUM OF EVEN NUMBERS USER INPUT DYNAMICALLY
SOURCODE:-

```
SQL> declare
                    x integer:=2;
                    y integer;
                    s integer:=0;
          begin
                    y:=&y;
                    while x<=y loop
                              dbms_output.put_line(x);
                    s:=s+x;
                    x:=x+2;
                    end loop;
          dbms_output.put_line('sum of even numbers is' || s);
          end;
```

```
            /
```

Output:-

```
Enter value for y: 10
old   6:    y:=&y;
new   6:    y:=10;
2
4
6
8
10
sum of even numbers is30

PL/SQL procedure successfully completed.
```

5. Aim:-SWAPPING OF TWO NUMBERS using temp

SourceCode:-

```
SQL> declare
                    x integer;
                    y integer;
                     temp int;
          begin
                    x:=10;
                    y:=20;
                    dbms_output.put_line('before');
                    dbms_output.put_line('x='||x||'y='||y);
                    temp:=x;
                    x:=y;
                     y:=temp;
                    dbms_output.put_line('after');
                    dbms_output.put_line('x='||x||'y='||y);
                    end;
                    /
```

OUTPUT;-

```
before
x=10y=20
after
x=20y=10
PL/SQL procedure successfully completed.
```

6. Aim:-SWAPPING OF TWO NUMBERS without using temp

Sourcecode;-

```
SQL>  declare
                    x integer;
                    y integer;
          begin
                    x:=10;
                    y:=20;
                    dbms_output.put_line('before');
```

```
                dbms_output.put_line('x='||x||'y='||y);
                x:=x+y;
                y:=x-y;
                x:=x-y;
                dbms_output.put_line('after');
                dbms_output.put_line('x='||x||'y='||y);
                 end;
                /
```

Output:-

```
before
x=10y=20
after
x=20y=10
PL/SQL procedure successfully completed.
```

7. Aim:-Find GCD for two numbers

Sourcecode;-

```
SQL> declare
                x integer;
                 y integer;
                t integer;
        begin
                x:=8;
                y:=48;
                while mod(y,x)!=0 loop
                t:=mod(y,x);
                y:=x;
                x:=t;
                end loop;
                dbms_output.put_line('GCD of'||x||'and'||y||'is'||x);
                end;
                /
```

Output:

```
GCD of8and48is8
PL/SQL procedure successfully completed.
```

8. Aim:-Greatest of three numbers

Sourcecode;-

```
SQL> declare
                a number:=46;
                b number:=67;
                c number:=21;
        begin
                 if  a>b and a>c then
                dbms_output.put_line('greatest number is'||a);
                elsif b>a and b>c then
                dbms_output.put_line('greatest number is'||b);
                else
                dbms_output.put_line('greatest number is'||c);
                end if;
         end;
        /
```

OUTPUT:-

```
greatest number is67

PL/SQL procedure successfully completed.
```

Example 2:

```
SQL> create table std(sno int,sname varchar2(10),age int,cgpa real,grade
varchar2(10));
Table created.
SQL> insert into std values(1,'A',18,9.7,'A');
1 row created.
SQL> insert into std values(2,'B',17,8.8,'B');
1 row created.
SQL> insert into std values(3,'C',18,7.3,'C');
1 row created.
SQL> SELECT * FROM STD;
      SNO SNAME             AGE       CGPA GRADE
---------- ---------- ---------- ---------- ----------
        1 A                 18        9.7 A
        2 B                 17        8.8 B
        3 C                 18        7.3 C
SQL> set serveroutput on;
```

Inserting rows into a relation

```
declare
        stuid std.sno%type:=&stuid;
        stuname std.sname%type:=&stuname;
        stuage std.age%type:=&stuage;
        stucgpa std.cgpa%type:=&stucgpa;
        stugrade std.grade%type;

begin
        if stucgpa>=9 then
                stugrade:='A';
        elsif stucgpa>=8 then
                stugrade:='B';
        else stugrade:='C';
        end if;
insert into std values(stuid, stuname, stuage, stucgpa, stugrade);
end;
/
```

OUTPUT:-

```
SQL> /
Enter value for stuid: 12
old   2: stuid std.sno%type:=&stuid;
new   2: stuid std.sno%type:=12;
Enter value for stuname: 'jj'
old   3: stuname std.sname%type:=&stuname;
new   3: stuname std.sname%type:='jj';
Enter value for stuage: 22
old   4: stuage std.age%type:=&stuage;
new   4: stuage std.age%type:=22;
Enter value for stucgpa: 9.2
old   5: stucgpa std.cgpa%type:=&stucgpa;
```

```
new   5: stucgpa std.cgpa%type:=9.2;

PL/SQL procedure successfully completed.

SQL> select * from std;
      SNO SNAME           AGE       CGPA   GRADE
---------- ----------        ---------- ----------       ----------
        1 A               18                  9.7               A
        2 B               17                  8.8               B
        3 C               18                  7.3               C
       12 jj              22                  9.2               A
```

Update rows in a relation

```
declare
          stuid std.sno%type:=&stuid;
          stucgpa std.cgpa%type:=&stucgpa;
          stugrade std.grade%type;
begin
update std set cgpa=stucgpa where sno=stuid;
          if stucgpa>=9 then
                    stugrade:='A';
          elsif stucgpa>=8 then
                    stugrade:='B';
          else stugrade:='C';
          end if;
update std set grade=stugrade where sno=stuid;
end;
/
```

OUTPUT:-

```
Enter value for stuid: 12
old   2: stuid std.sno%type:=&stuid;
new   2: stuid std.sno%type:=12;
Enter value for stucgpa: 7.2
old   3: stucgpa std.cgpa%type:=&stucgpa;
new   3: stucgpa std.cgpa%type:=7.2;
PL/SQL procedure successfully completed.
SQL> select * from std;
      SNO SNAME           AGE       CGPA GRADE
---------- ---------- ---------- ---------- ----------
        1 A               18                  9.7               A
        2 B               17                  8.8               B
        3 C               18                  7.3               C
       12 jj              22                  7.2                C
```

delete TUPLES using pl/sql

```
declare
          stuid std.sno%type:=&stuid;
begin
delete from std where sno=stuid;
end;
/
Enter value for stuid: 12
old   2: stuid std.sno%type:=&stuid;
```

```
new   2: stuid std.sno%type:=12;

PL/SQL procedure successfully completed.

SQL> select * from std;
       SNO SNAME             AGE       CGPA GRADE
---------- ---------- ---------- ---------- ----------
         1 A                  18        9.7 A
         2 B                  17        8.8 B
         3 C                  18        7.3 C
```

Retrieving values from table using Pl/Sql

```
declare
        stuid std.sno%type:=&stuid;
        stuname std.sname%type;
        stuage std.age%type;
        stucgpa std.cgpa%type;
        stugrade std.grade%type;
begin
select sno,sname,age,cgpa,grade into stuid,stuname,stuage,stucgpa,stugrade from std
where sno=stuid;
dbms_output.put_line('student id is'||stuid);
dbms_output.put_line('student name  is'||stuname);
dbms_output.put_line('student age  is'||stuage);
dbms_output.put_line('cgpa is'||stucgpa||'grade is'||stugrade);
end;
/
```

Output:-

```
Enter value for stuid: 1
old   2: stuid std.sno%type:=&stuid;
new   2: stuid std.sno%type:=1;
student id is1
student name  isA
student age  is18
cgpa is9.7grade isA

PL/SQL procedure successfully completed.

Enter value for stuid: 66
old   2: stuid std.sno%type:=&stuid;
new   2: stuid std.sno%type:=66;
declare
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 8
```

Example 3:

Sailors relation

Inserting tuples into relation

```
declare
s_id sailors.sid%type:=&s_id;
```

```
s_name sailors.sname%type:=&s_name;
s_rating sailors.rating%type:=&s_rating;
s_age sailors.age%type:=&s_age;
begin
insert into sailors values(s_id,s_name,s_rating,s_age);
end;
/
```

Output:-

```
Enter value for s_id: 200
old   2: s_id sailors.sid%type:=&s_id;
new   2: s_id sailors.sid%type:=200;
Enter value for s_name: 'ksru'
old   3: s_name sailors.sname%type:=&s_name;
new   3: s_name sailors.sname%type:='ksru';
Enter value for s_rating: 20
old   4: s_rating sailors.rating%type:=&s_rating;
new   4: s_rating sailors.rating%type:=20;
Enter value for s_age: 58
old   5: s_age sailors.age%type:=&s_age;
new   5: s_age sailors.age%type:=58;

PL/SQL procedure successfully completed.
SQL> select * from sailors;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 dustin              7         45
        29 brutus              1         33
        31 lubber              8       55.5
        32 andy                8       25.5
        58 rusty              10         35
        64 horatio             7         35
        71 zorba              10         16
        74 horatio             9         35
        85 art                 3       25.5
        95 bob                 3       63.5
       200 ksru               20         58

11 rows selected.
```

Update tuples in a relation

```
declare
s_id sailors.sid%type:=&s_id;
s_name sailors.sname%type:=&s_name;
begin
update sailors set sname=s_name where sid=s_id;
end;
/
```

Output:-

```
Enter value for s_id: 200
old   2: s_id sailors.sid%type:=&s_id;
new   2: s_id sailors.sid%type:=200;
Enter value for s_name: 'jaya'
```

```
old   3: s_name sailors.sname%type:=&s_name;
new   3: s_name sailors.sname%type:='jaya';

PL/SQL procedure successfully completed.
SQL> select * from sailors;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 dustin              7         45
        29 brutus              1         33
        31 lubber              8       55.5
        32 andy                8       25.5
        58 rusty              10         35
        64 horatio             7         35
        71 zorba              10         16
        74 horatio             9         35
        85 art                 3       25.5
        95 bob                 3       63.5
       200 jaya               20         58

11 rows selected.
```

Delete tuples from a relation

```
declare
s_id sailors.sid%type:=&s_id;
begin
delete from sailors where sid=s_id;
end;
/
```

Output:-

```
Enter value for s_id: 200
old   2: s_id sailors.sid%type:=&s_id;
new   2: s_id sailors.sid%type:=200;

PL/SQL procedure successfully completed.
SQL> select * from sailors;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 dustin              7         45
        29 brutus              1         33
        31 lubber              8       55.5
        32 andy                8       25.5
        58 rusty              10         35
        64 horatio             7         35
        71 zorba              10         16
        74 horatio             9         35
        85 art                 3       25.5
        95 bob                 3       63.5
```

```
10 rows selected.
```

VIVA QUESTIONS:

1. What is the difference between SQL and PL/SQL?
2. Inserting rows in PL/SQL.
3. update and deletion of rows using PL/SQL.

EXERCISE: 10

AIM:

Write a PL/SQL block for transaction operations of a typical application using triggers. **DESCRIPTION:**

Trigger:

Trigger is a special kind of stored procedure i.e, automatically executed when an event occurs.

We have two types of triggers:

1. Row-level triggers—these are executed for each row.
2. Statement-level triggers—at one time, triggers are executed.

Syntax for triggers:

```
create  or replace trigger <trigger-name>
before or after
insert or delete or update on <table-name>
for  each row
when(condition)
declare
        declare statements;
begin
        executable statements;
        exception handling statements;
end;
/
```

1.Row –level Triggers:

```
create table emp(eid number(10),ename varchar(10),salary number(10));
output: table created.
create or replace Trigger display_sal_change
before
insert or delete or update on emp15
for each row
declare
sal_diff number;
begin
sal_diff:=:old.salary-:new.salary;
dbms_output.put_line(‘old salary is:’||:old.sal);
dbms_ output.put_line(‘new salary is:’||:new.sal);
dbms_ output.put_line(‘salary diff is:’||sal_diff);
end;
/
```

Output: trigger is created.

```
insert into emp15 values(101,’ram’,5000);
1 row(s)  inserted.
insert into emp values(102,’rama’,6000);
1 row(s) inserted
Select  * from emp;
```

Output:

```
update emp set sal=6000 where eid=101;
output:
1 row updated
Old salary is:5000
New salary is:6000
Salary diff is:-1000
```

Example 2:

```
create table account(acctno number(10),amount number(10));
create or replace trigger disp_notification
after
insert or update or delete on account
for each row
begin
if  :new.amount<100 then
dbms_output.put_line(‘account bal is low’);
else
dbms_output.put_line(‘transaction successful’);
end if;
end;
/
Output: trigger is created
Insert into account values(501,5000);
Output:1 row(s) inserted
Transaction successful
Insert into account values(512,3000);
Output: 1 row(s) inserted
Transaction successful
Select * from account;
Output:
Insert into account values(503,50);
Output: 1 row(s) inserted
Account bal is low.
```

Example:

```
create table std(id number(10) ,name varchar(10),marks number(5));
output: table created.
create table marks(id number(10), oldmarks number(10),newmarks
number(10),foreign key(id) references std(id));
Output: table created.
->create or replace trigger disp_marks_change
before
insert or delete or update on std
for each row
begin
insert into marks values(:old.id,:old.marks,:new.marks);
end;
/
```

Output: trigger created

```
Insert into std values(101,’ram’,90);
Output:1 row(s) inserted
Insert into std values(105,’raghu’,95);
Output: 1 row(S) inserted
update  std set marks=80 where id=105;
output: 1 row(s) updated
old.id:105
old.marks:90
new.marks:80
select * from std;
output:
select * from marks;
output:
```

example:

```
create table count(description varchar(10),id number(10),foreign key(id) references std(id));
output:table created
create or replace trigger stmt_level
after insert or update or delete on std
begin
insert into count values('stmt level trigger is fired');
end;
/
```

Output: trigger created.

VIVA QUESTIONS:

1. What is Trigger?
2. Syntax for creating a trigger.
3. How many types of triggers are there and what are they?

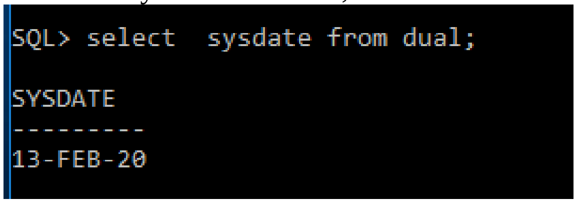
Additional Experiments

1. Execute Date functions.

dual functions/date functions:

1.current date: to get the current date.

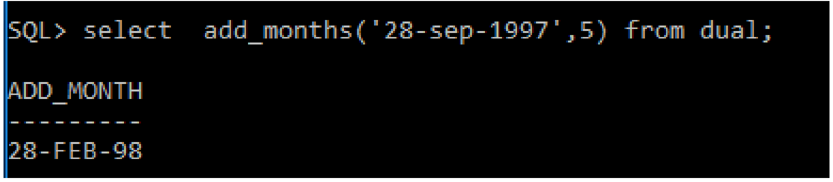
ex: select sysdate from dual;



```
SQL> select  sysdate from dual;

SYSDATE
---------
13-FEB-20
```

2. add_months(): this function is used to add the 'n' number of months to a given date.

ex: select add_months('28-sep-1997',5) from dual;



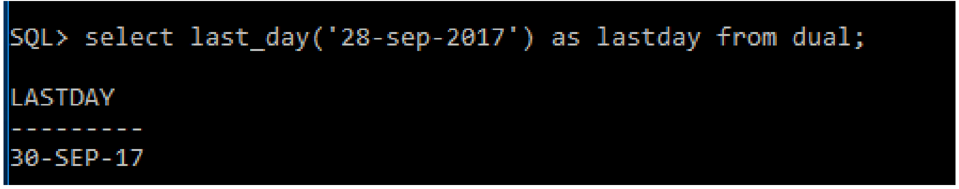
```
SQL> select  add_months('28-sep-1997',5) from dual;

ADD_MONTH
---------
28-FEB-98
```

3. last_day():

it gives the last day of the specified month in a date

syn: last_date(date)

ex: select last_day('28-sep-2017') as lastday from dual;



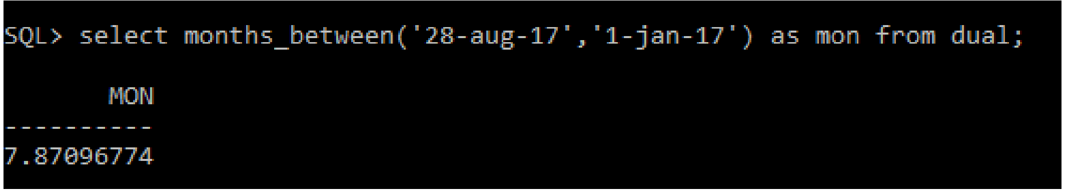
```
SQL> select last_day('28-sep-2017') as lastday from dual;

LASTDAY
---------
30-SEP-17
```

4. 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 the last day of the month.
Decimal result	If days are different and they are not both specify the last day of the month
Fractional part	Always calcilated as the difference between days divided by 31 despite the 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;

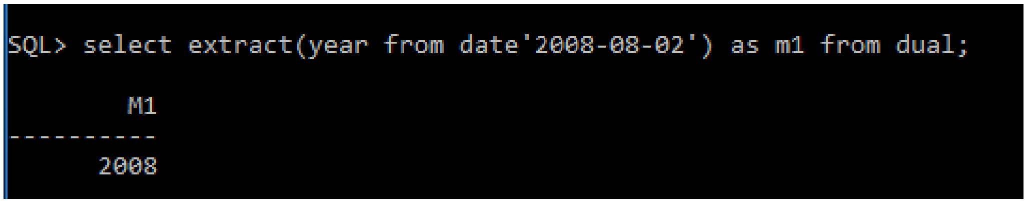


```
SQL> select months_between('28-aug-17','1-jan-17') as mon from dual;

       MON
----------
7.87096774
```

5. extract():

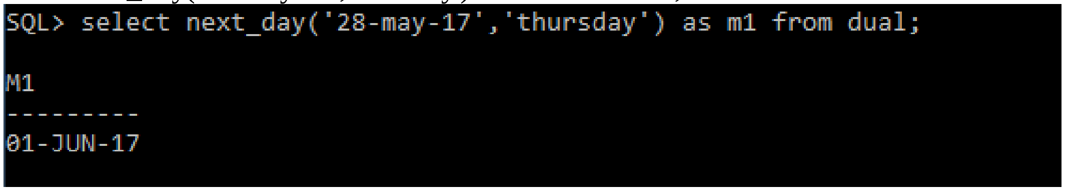
it is used to extract time component from date expression.

select extract(year from date'2008-08-02') as m1 from dual;



```
SQL> select extract(year from date'2008-08-02') as m1 from dual;

        M1
----------
      2008
```

6. next day:
next_day(date,dayname)
select next_day('28-may-17','thursday') as m1 from dual;



```
SQL> select next_day('28-may-17','thursday') as m1 from dual;

M1
---------
01-JUN-17
```

2. Execute Pl/SQL commands for exception handling.

Exception: any run time error is known as exception.
create table emp(id number(10)primary key, name varchar2(20), age number(5));
insert three rows into table.

System defined(predefined) exceptions:
These are built in exceptions and handled by system by using handler provided by the user.

```
declare
eid emp.id%type:=&eid;
ename emp.name%type;
eage emp.age%type;
begin
select id,name,age into eid,ename,eage from emp where id=eid;
dbms_output.put_line(eid);
dbms_output.put_line(ename);
dbms_output.put_line(eage);
exception
        when no_data_found
then dbms_output.put_line('no such employee found');
end;
/
```

Output: no such employee found

User defined exceptions:
Sql supports handling of user defined exceptions.

```
declare
eid emp.id%type:=&eid;
ename emp.name%type;
eage emp.age%type;
invalid_id exception;
begin
if eid<=0 then
raise invalid_id;
else
select id,name,age into eid,ename,eage from emp where id=eid;
dbms_output.put_line(eid);
dbms_output.put_line(ename);
dbms_output.put_line(eage);
```

```
end if;
exception
when invalid_id
then dbms_output.put_line(‘employee must be greater than zero’);
when no_data_found
then dbms_output.put_line(‘no such employee found’);
end;
/
Output: enter empid: -20
employee must be greater than zero.
```

3. Execute PL/SQL Procedures

PROCEDURES: Database Procedures (sometimes referred to as Stored Procedures or Procs) are subroutines that can contain one or more SQL statements that perform a specific task. They can be used for data validation, access control, or to reduce network traffic between clients and the DBMS servers.

```
SQL>  create or replace procedure high(a number,b number) is
   begin
   if a>b then
   dbms_output.put_line('max value iS:='||a);
   else
   dbms_output.put_line('max value iS:='||b);
   end if;
   end;
```

OUTPUT:

```
Procedure created.
SQL> exec high(20,10);
max value iS:=20
PL/SQL procedure successfully completed.
SQL> create or replace procedure fact(n in number) is
   fact number:=1;
   i number;
    begin
   for i in 1..n loop
    fact:=fact * i;
   end loop;
   dbms_output.put_line('the factorial value is'||fact);
   end;
```

OUTPUT:

```
Procedure created.
SQL> exec fact(10);
the factorial value is3628800
PL/SQL procedure successfully completed.
SQL> create or replace procedure fact(n in number,f out number) is
   f1 number:=1;
   i number;
   begin
   for i in 1..n loop
   f1:=f1 * i;
   end loop;
   f:=f1;
   end;
```

OUTPUT:

```
Procedure created.
SQL> declare
   n number:=&n;
   f number;
   begin
   fact(n,f);
   dbms_output.put_line('the factorial is'||f);
   end;
```

OUTPUT:

```
Enter value for n: 5
old   2: n number:=&n;
new   2: n number:=5;
the factorial is120
PL/SQL procedure successfully completed.
SQL> create or replace procedure fact(n in number,f in out number) is
   f1 number;
   i number;
   begin
   f1:=f;
   for i in 1..n loop
   f1:=f1 * i;
   end loop;
   f:=f1;
  end;
```

OUTPUT:

```
Procedure created.
SQL> declare
   n number:=&n;
   f number:=1;
   begin
   fact(n,f);
   dbms_output.put_line('factorial value is:'||f);
   end;
```

OUTPUT:

```
Enter value for n: 6
old   2: n number:=&n;
new   2: n number:=6;
factorial value is:720
PL/SQL procedure successfully completed.
```