J.N.T.U.H. COLLEGE OF ENGINEERING

KUKATPALLY, HYDERABAD – 500 085

CERTIFICATE

This is to certify that ______________________of B.Tech III year I Semester bearing the Hall-Ticket number ___________ has fulfilled his/her DATABASE MANAGEMENT SYSTEMS LAB record for the academic year 2018-2019.

Signature of the Head of the Department　　　　Signature of the staff member

Date of Examination_________________________

Internal Examiner　　　　External Examiner

TABLE OF CONTENTS

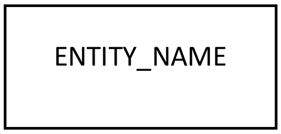
Topic 1: Conceptual Design with E-R Model

- Entity-relationship model is a model used for design and representation of relationships between data.
- The main data objects are termed as Entities, with their details defined as attributes, some of these attributes are important and are used to identity the entity, and different entities are related using relationships.
- An **Entity** is generally a real-world object which has characteristics and holds relationships in a DBMS.
- If a Student is an Entity, then the complete dataset of all the students will be the **Entity Set.**
- An **attribute** is a property or descriptor of an entity. **Attributes Define** Entities.

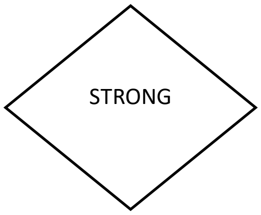
- When an Entity is related to another Entity, they are said to have a relationship. For example, A **Class** Entity is related to **Student** entity, because students study in classes, hence this is a relationship.

Components Of E-R diagram.

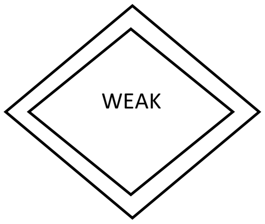
- **Entity: Represented by a Rectangle.**



- **Relationships between Entities - Weak and Strong**



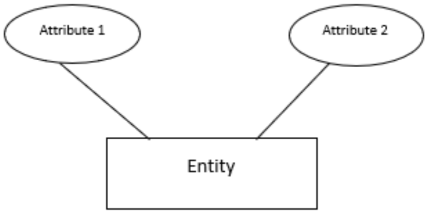
STRONG ENTITY



WEAK ENTITY

- **Attributes for any Entity**

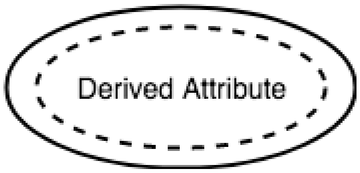
Ellipse is used to represent attributes of any entity. It is connected to the entity.



- **Derived Attribute for any Entity**

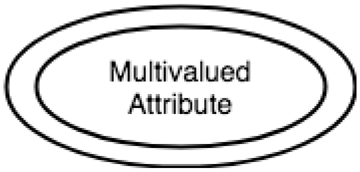
Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth.

To represent a derived attribute, another dotted ellipse is created inside the main ellipse.



- **Multivalued Attribute for any Entity**

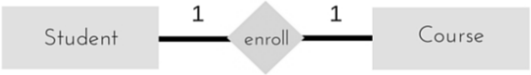
Double Ellipse, one inside another, represents the attribute which can have multiple values.



TYPES OF RELATIONSHIPS:

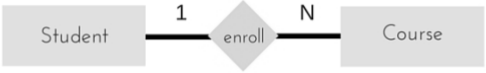
- **One to One Relationship**

This type of relationship is rarely seen in real world. The above example describes that one student can enroll only for one course and a course will also have only one Student. This is not what you will usually see in real-world relationships.



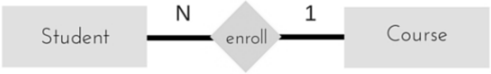
- **One to Many Relationship**

The below example showcases this relationship, which means that 1 student can opt for many courses, but a course can only have 1 student. Sounds weird! This is how it is.

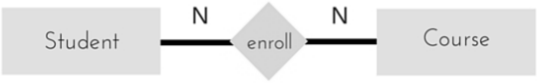


- **Many to One Relationship**

  It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.

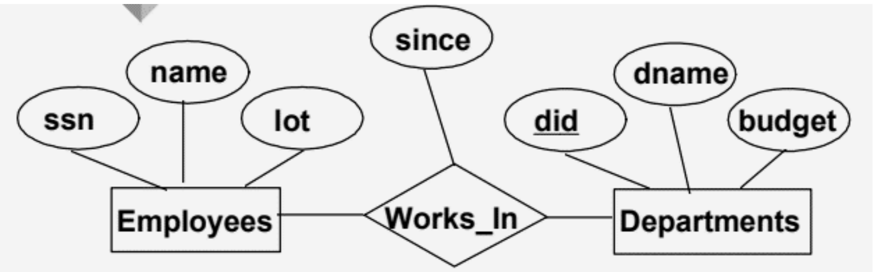


- **Many to Many Relationship**



The above diagram represents that one student can enroll for more than one courses. And a course can have more than 1 student enrolled in it.

Example:


since
name
ssn
lot
did
dname
budget
Employees
Works_In
Departments

Topic 2: Relational Model

Relational data model is one the models widely used for storing and data processing.

Concepts Used in Relational Model

Tables − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple − A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema − A relation schema describes the relation name (table name), attributes, and their names.

Relation key − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain − Every attribute has some pre-defined value scope, known as attribute domain.

Advantages of using Relational model:

- **Simplicity**: A relational data model is simpler than the hierarchical and network model.
- **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use**: The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- **Query capability**: It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence**: The structure of a database can be changed without having to change any application.
- **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of using Relational model

- Few relational databases have limits on field lengths which can't be exceeded.

- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

Example:

Consider three Relations

Relation 1: Sailors(sailor_id,sailor_name,rating,age)

Relation 2: Reserves(sailor_id,boat_id,day)

Relation 3: Boat (boat_id,boat_name,boat_color)

Here the three relations are relation through sailor id and boat id.

Sailors ,reserves and boat are entities. The column names are mentioned In parenthesis.

Initially, we create a database. We create three relations by the name sailors, reserves and boat ,Now, we can insert,delete, update, modify the data in them using various DDL and DML commands.In addition we also have constraints such as keys to uniquely identify tuples of relations.

Topic 3: Normalization

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

It divides larger tables to smaller tables and links them using relationships.

1NF (First Normal Form) Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

Student	Age	Subject
Adam	15	Biology,Maths
Alex	14	Maths
Stuart	17	Maths

Here the first column has two values. So we need to convert into 1st normal form which is:

Student	Age	Subject
Adam	15	Biology
Alex	14	Maths
Stuart	17	Maths
Adam	15	Maths

2NF (Second Normal Form) Rules

- Rule 1- Be in 1NF
- Rule 2- There should not be any partial dependency

S_id	S_name	c_id	c_name
101	A	234	Computers
102	B	345	Finance
198	R	343	Mechanics

After converting into 2nd normal form:

S_id	S_name	c_id
101	A	234
102	B	345
198	R	343

c_id	c_name
234	Computers
345	Finance
343	Mechanics

3RD (Third Normal Form) Rules

- Rule 1- Be in 2NF
- Rule 2- There should not be any transitive dependency

S_id	S_name	City	Pincode
101	A	ABC	234343
102	B	BAC	593489
198	R	CAB	500034

After converting into 3rd normal form.

S_id	S_name	City
101	A	ABC
102	B	BAC
198	R	CAB

Pincode	City
234343	ABC
593489	BAC
500034	CAB

BOYCE-CODD NORMAL FORM(BCNF) Rules:

- Rule 1- Be in 3NF
- Rule 2- For any non-trivial functional dependency, X → A, X must be a super-key.

The above relations are already in boyce codd normal form.

The functional dependencies are s_id → s_name,city

Pincode →city

Here s_id and Pincode are both super keys.

Topic 4: Practicing DDL commands

DDL (Data Definition Language) : DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

Examples of DDL commands:

1.CREATE: There are two CREATE statements available in SQL:

- **CREATE DATABASE**

  Syntax: CREATE DATABASE database_name;

- **CREATE TABLE**

  Syntax :

```
CREATE TABLE table_name
(
column1 data_type(size),
column2 data_type(size),
column3 data_type(size),
....
);
```

Example 1:

```
create table reserve_u509
(
sailor_id integer unique,
boat_id integer unique,
day date
);
```

Example 2:

```
create table reserve_u509
(
sailor_id integer unique,
boat_id integer unique,
day date
);
```

Output:

```
Query returned successfully with no result in 170 msec.
```

2.DROP

DROP is used to delete a whole database or just a table.The DROP statement destroys the objects like an existing database, table, index, or view.
A DROP statement in SQL removes a component from a relational database management system (RDBMS).

- **DROP DATABASE**

Syntax:

```
DROP DATABASE database_name;
```

- **DROP TABLE**

Syntax:

```
DROP TABLE table_name
```

Example:

```
DROP TABLE STUDENT;
```

Output:

```
Query returned successfully with no result in 170 msec.
```

3.ALTER

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

Syntax 1:

```
ALTER TABLE table_name

ADD (Columnname_1  datatype,
Columnname_2  datatype,
  …
Columnname_n  datatype);
```

Syntax 2:
ALTER TABLE table_name

DROP COLUMN column_name;

Example:

STUDENT TABLE

STUDENT ID	STUDENT NAME	STUDENT AGE
10101	Anna	22
18272	Lisa	23
23455	Maria	19

ALTER TABLE Student DROP COLUMN STUDENT AGE;

STUDENT ID	STUDENT NAME
10101	Anna
18272	Lisa
23455	Maria

Topic 5: Practicing DML Commands

DML (Data Manipulation Language) : The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements**.**

Examples of DML commands:

1.<u>INSERT</u> – is used to insert data into a table

Syntax:

```
INSERT INTO table_name VALUES (value1, value2, value3 ,...);
```

table_name: name of the table.
value1, value2,.. : value of first column, second column,... for the new record

(OR)

```
INSERT INTO table_name (column1, column2, column3,..) VALUES ( value1,
value2, value3,..);
```

table_name: name of the table.
column1: name of first column, second column ...
value1, value2, value3 : value of first column, second column,... for the new record

Examples:

```
insert into author values(100,'Shreya','Bangalore','India');
insert into author values(101,'Suchi','Hyderabad','India');
insert into author values(102,'Lisa','Mexico','USA');
insert into author values(103,'Anna','Vatican city','Europe');
insert into author values(104,'seo jyung joon','Seoul','South korea');
```

```
insert into reserve_u509 (sailor_id,boat_id,day)
values(101,10000,'2018-06-29');
insert into reserve_u509 (sailor_id,boat_id,day)
values(102,10001,'2018-07-01');
insert into reserve_u509 (sailor_id,boat_id,day)
values(103,10002,'2018-07-05');
insert into reserve_u509 (sailor_id,boat_id,day)
```

```
values(104,10003,'2018-07-10');
insert into reserve_u509 (sailor_id,boat_id,day)
values(105,10004,'2018-07-15');
```

<u>2.SELECT</u>: is used to retrieve data from the a database.

Syntax 1: To fetch all the columns of a table.

SELECT * FROM table_name;

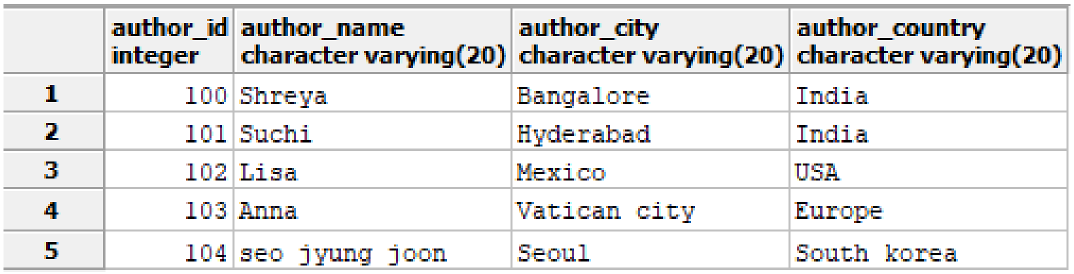
Syntax 2: To fetch particular columns of a table

SELECT column1,column2 FROM table_name ;

column1 , column2: names of the fields of the table
table_name: from where we want to fetch

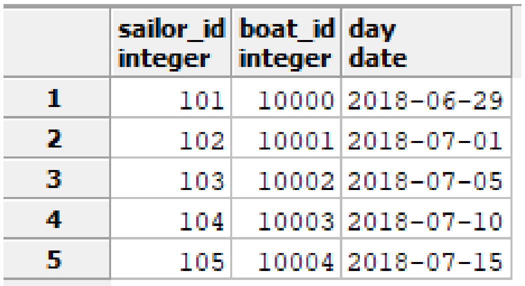
Examples:

SELECT * from author;



	author_id integer	author_name character varying(20)	author_city character varying(20)	author_country character varying(20)
1	100	Shreya	Bangalore	India
2	101	Suchi	Hyderabad	India
3	102	Lisa	Mexico	USA
4	103	Anna	Vatican city	Europe
5	104	seo jyung joon	Seoul	South korea

SELECT * from reserve;



	sailor_id integer	boat_id integer	day date
1	101	10000	2018-06-29
2	102	10001	2018-07-01
3	103	10002	2018-07-05
4	104	10003	2018-07-10
5	105	10004	2018-07-15

3.UPDATE:

The UPDATE statement in SQL is used to update the data of an existing table in database. We can update single columns as well as multiple columns using UPDATE statement as per our requirement.

Syntax:

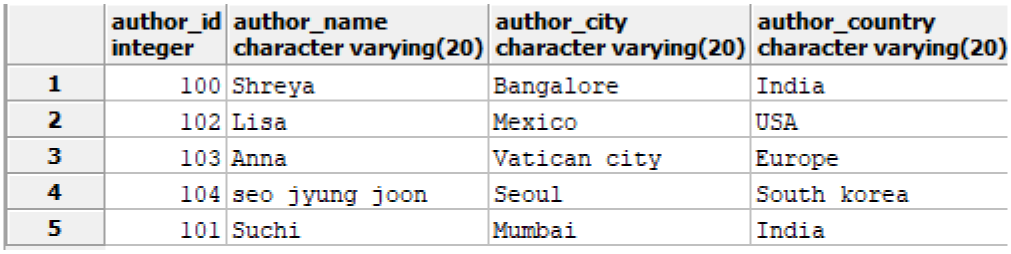
UPDATE table_name SET column1 = value1, column2 = value2,...

WHERE condition;

table_name: name of the table
column1: name of first , second, third column....
value1: new value for first, second, third column....
condition: condition to select the rows for which the
values of columns needs to be updated.

Example:

update author
set author_city='Mumbai'
where author_id=101



	author_id integer	author_name character varying(20)	author_city character varying(20)	author_country character varying(20)
1	100	Shreya	Bangalore	India
2	102	Lisa	Mexico	USA
3	103	Anna	Vatican city	Europe
4	104	seo jyung joon	Seoul	South korea
5	101	Suchi	Mumbai	India

4.DELETE

The DELETE Statement in SQL is used to delete existing records from a table. We can delete a single record or multiple records depending on the condition we specify in the WHERE clause.

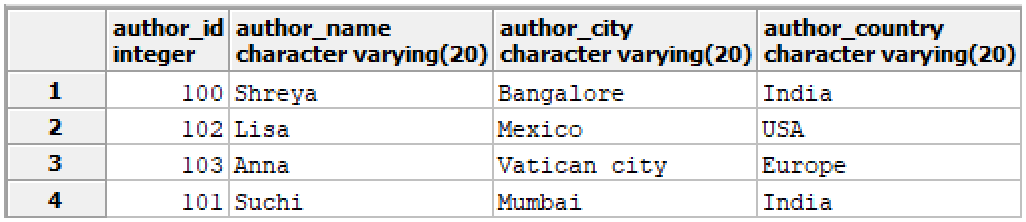
Syntax:

DELETE FROM table_name WHERE some_condition;

table_name: name of the table
some_condition: condition to choose particular record.

Example:

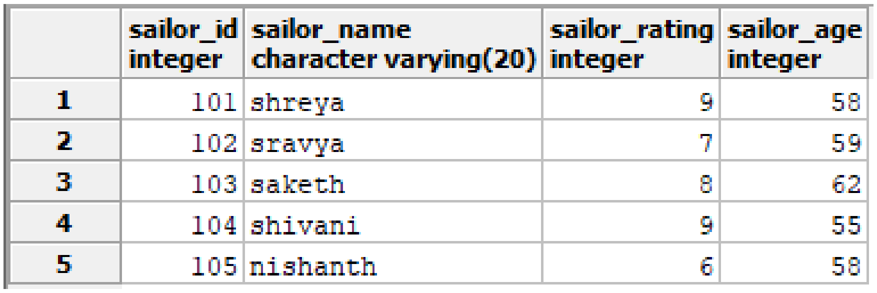
```
DELETE FROM author
WHERE author_is=104;
```



	author_id integer	author_name character varying(20)	author_city character varying(20)	author_country character varying(20)
1	100	Shreya	Bangalore	India
2	102	Lisa	Mexico	USA
3	103	Anna	Vatican city	Europe
4	101	Suchi	Mumbai	India

Topic 6: Querying using ANY, ALL, IN, EXISTS, NOT EXISTS, UNION, INTERSECT, CONSTRAINTS etc.

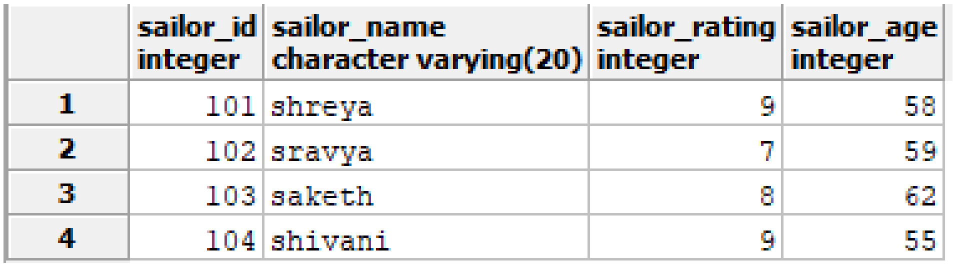
CONSIDER TABLE SAILORS(sailor_id, sailor_name, sailor_rating, sailor_age)



	sailor_id	sailor_name	sailor_rating	sailor_age
	integer	character varying(20)	integer	integer
1	101	shreya	9	58
2	102	sravya	7	59
3	103	saketh	8	62
4	104	shivani	9	55
5	105	nishanth	6	58

1.ANY:

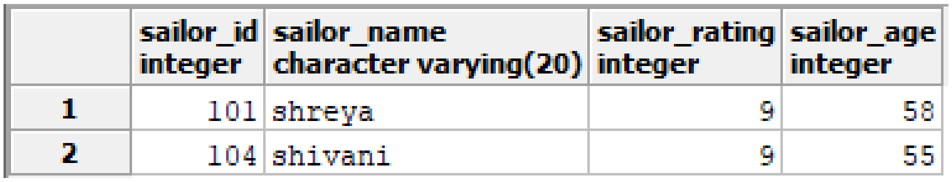
```
SELECT * FROM sailors_u509 s
WHERE s.sailor_rating > ANY (SELECT s2.sailor_rating FROM sailors_u509 s2
                             WHERE s2.sailor_name = 'nishanth'
                             )
```



	sailor_id	sailor_name	sailor_rating	sailor_age
	integer	character varying(20)	integer	integer
1	101	shreya	9	58
2	102	sravya	7	59
3	103	saketh	8	62
4	104	shivani	9	55

2.ALL:

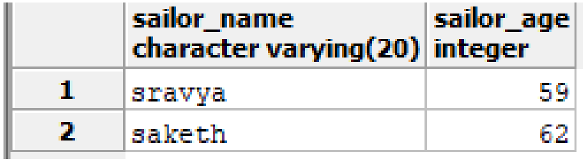
```
SELECT * FROM sailors_u509 s
WHERE s.sailor_rating >= ALL (SELECT s2.sailor_rating FROM sailors_u509 s2)
```



	sailor_id	sailor_name	sailor_rating	sailor_age
	integer	character varying(20)	integer	integer
1	101	shreya	9	58
2	104	shivani	9	55

3.IN:

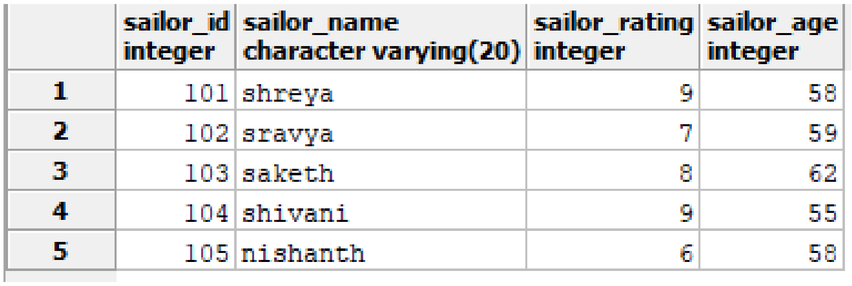
```
SELECT sailor_name,sailor_age FROM sailors_u509 s
WHERE s.sailor_rating  IN (select s2.sailor_rating from sailors_u509 s2
                           where s2.sailor_age>58
                           )
```



	sailor_name character varying(20)	sailor_age integer
1	sravya	59
2	saketh	62

3.EXISTS

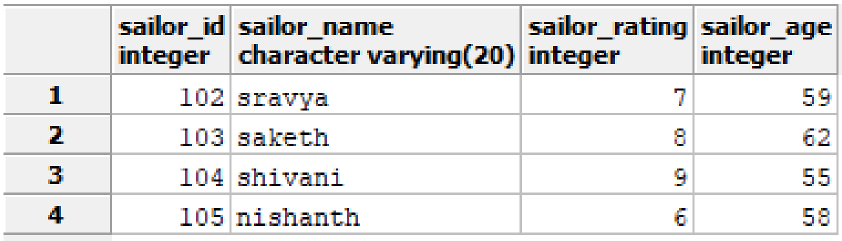
```
SELECT * FROM sailors_u509 s
WHERE EXISTS (SELECT s2.sailor_rating FROM sailors_u509 s2
                   WHERE s2.sailor_rating > 8)
```



	sailor_id integer	sailor_name character varying(20)	sailor_rating integer	sailor_age integer
1	101	shreya	9	58
2	102	sravya	7	59
3	103	saketh	8	62
4	104	shivani	9	55
5	105	nishanth	6	58

4.NOT EXISTS

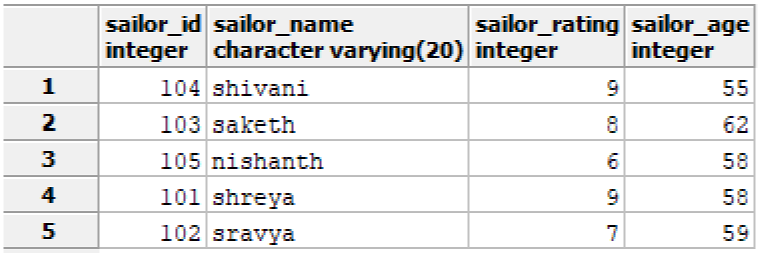
```
SELECT * FROM sailors_u509 s
WHERE NOT EXISTS (SELECT r.sailor_id FROM reserve_u509 r
                   WHERE r.sailor_id = 101 and r.sailor_id=s.sailor_id)
```



	sailor_id integer	sailor_name character varying(20)	sailor_rating integer	sailor_age integer
1	102	sravya	7	59
2	103	saketh	8	62
3	104	shivani	9	55
4	105	nishanth	6	58

5.UNION

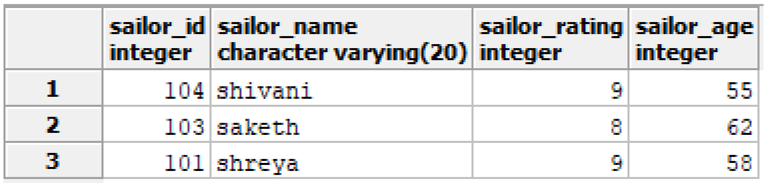
```
SELECT * FROM sailors_u509 s
WHERE s.sailor_rating >= 8
UNION
SELECT * FROM sailors_u509 s
WHERE s.sailor_rating <=10
```



	sailor_id integer	sailor_name character varying(20)	sailor_rating integer	sailor_age integer
1	104	shivani	9	55
2	103	saketh	8	62
3	105	nishanth	6	58
4	101	shreya	9	58
5	102	sravya	7	59

6.INTERSECT

```
SELECT * FROM sailors_u509 s
WHERE s.sailor_rating >= 8
INTERSECT
SELECT * FROM sailors_u509 s
WHERE s.sailor_rating <=10
```



	sailor_id integer	sailor_name character varying(20)	sailor_rating integer	sailor_age integer
1	104	shivani	9	55
2	103	saketh	8	62
3	101	shreya	9	58

Topic 7: Queries using Aggregate Functions , GROUP BY, HAVING

1.Aggregate Functions

- **COUNT**

  Count(*): Returns total number of records
  Count(column_name): Return number of Non Null values in that column.
  Count(Distinct Salary): Return number of distinct Non Null values int that column.

  Examples:

  ```
  SELECT COUNT(*) FROM sailors_u509;
  ```

  Output : 5

  ```
  SELECT COUNT(sailor_id) FROM sailors_u509;
  ```

  Output : 5

  ```
  SELECT COUNT(distinct sailor_rating) FROM sailors_u509;
  ```

  Output : 4

- **SUM**

  Example:

  ```
  SELECT sum(sailor_rating) FROM sailors_u509;
  ```

  Output : 39

  ```
  SELECT sum( distinct sailor_rating) FROM sailors_u509;
  ```

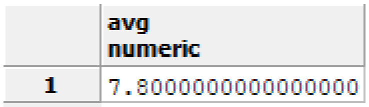
  Output: 30

- **AVG**

  Example

```
SELECT AVG(sailor_rating) FROM sailors_u509;
```

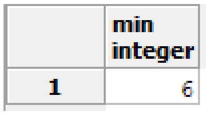
Output:



	avg numeric
1	7.8000000000000000

- **MIN**

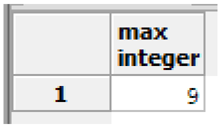
```
SELECT MIN(sailor_rating) FROM sailors_u509;
```



	min integer
1	6

- **MAX**

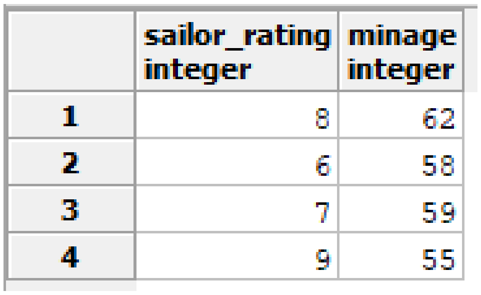
```
SELECT MAX(sailor_rating) FROM sailors_u509;
```



	max integer
1	9

2.GROUP BY:

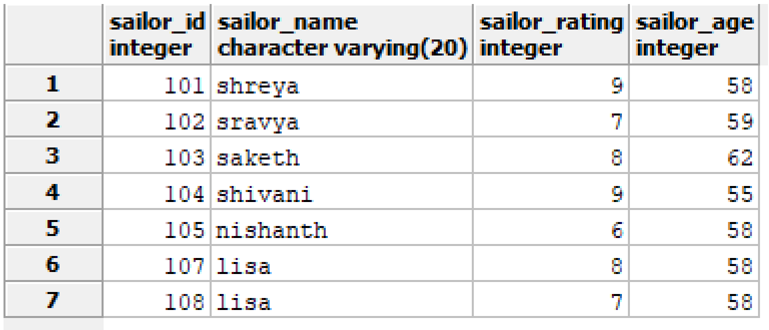
```
SELECT S.sailor_rating,
 MIN (S.sailor_age) AS minage
FROM sailors_u509 S
WHERE S.sailor_age >= 18
GROUP BY S.sailor_rating
```



	sailor_rating integer	minage integer
1	8	62
2	6	58
3	7	59
4	9	55

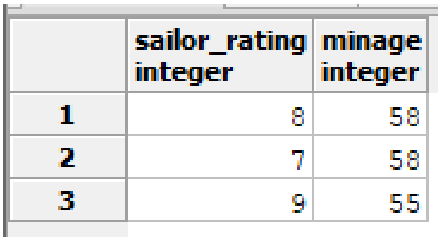
3.HAVING CLAUSE

Consider the following instance of the relation sailors



	sailor_id integer	sailor_name character varying(20)	sailor_rating integer	sailor_age integer
1	101	shreya	9	58
2	102	sravya	7	59
3	103	saketh	8	62
4	104	shivani	9	55
5	105	nishanth	6	58
6	107	lisa	8	58
7	108	lisa	7	58

```
SELECT S.sailor_rating,
 MIN (S.sailor_age) AS minage
FROM sailors_u509 S
WHERE S.sailor_age >= 18
GROUP BY S.sailor_rating
HAVING COUNT(*) > 1
```



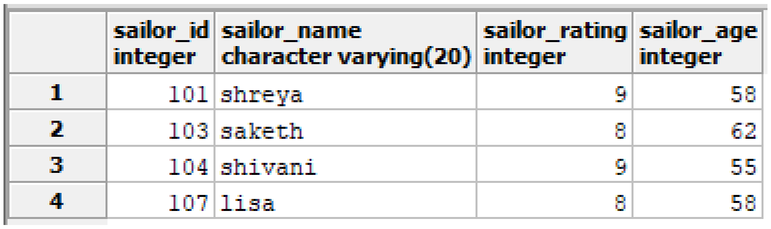
	sailor_rating integer	minage integer
1	8	58
2	7	58
3	9	55

4. VIEWS

CREATING VIEWS:

```
CREATE VIEW sailor_view
(sailor_id, sailor_name,sailor_rating,sailor_age) AS
SELECT sailor_id ,sailor_name,sailor_rating ,sailor_age
FROM sailors_u509
WHERE sailor_rating>7;
```

SELECT * FROM sailor_view;

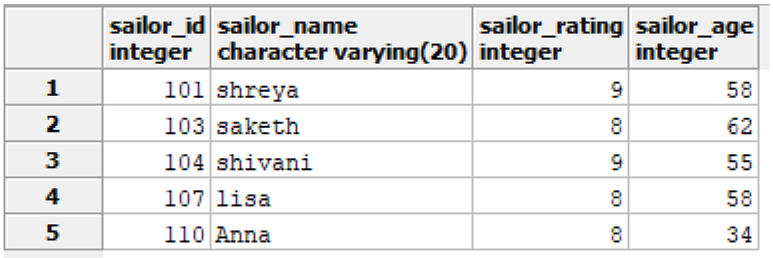


	sailor_id	sailor_name	sailor_rating	sailor_age
	integer	character varying(20)	integer	integer
1	101	shreya	9	58
2	103	saketh	8	62
3	104	shivani	9	55
4	107	lisa	8	58

INSERTING INTO VIEWS:

INSERT INTO sailor_view
VALUES (110,’Anna’,8,34)

SELECT * FROM sailor_view

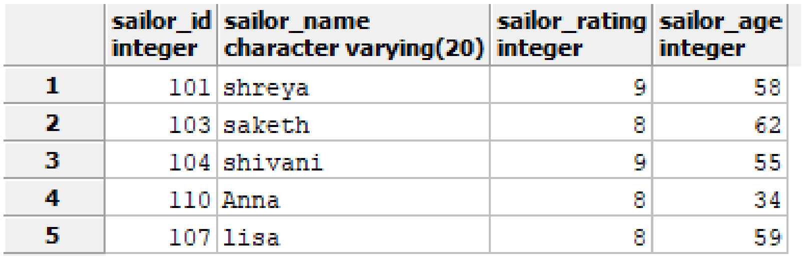


	sailor_id	sailor_name	sailor_rating	sailor_age
	integer	character varying(20)	integer	integer
1	101	shreya	9	58
2	103	saketh	8	62
3	104	shivani	9	55
4	107	lisa	8	58
5	110	Anna	8	34

UPDATING VIEWS

UPDATE sailor_view
SET sailor_age= sailor_age+1
WHERE sailor_name= ‘lisa’

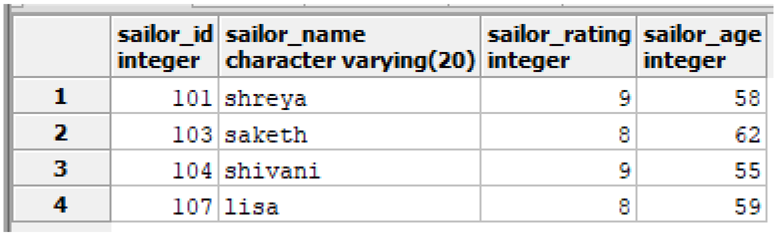
SELECT * FROM sailor_view



	sailor_id	sailor_name	sailor_rating	sailor_age
	integer	character varying(20)	integer	integer
1	101	shreya	9	58
2	103	saketh	8	62
3	104	shivani	9	55
4	110	Anna	8	34
5	107	lisa	8	59

DELETING FROM VIEWS.

```
DELETE FROM sailor_view
WHERE sailor_id=110
```



	sailor_id integer	sailor_name character varying(20)	sailor_rating integer	sailor_age integer
1	101	shreya	9	58
2	103	saketh	8	62
3	104	shivani	9	55
4	107	lisa	8	59

Topic 8 : Triggers Usage

Syntax:

```
CREATE TRIGGER incr_count
AFTER INSERT
ON STUDENT
FOR EACH ROW
WHEN (new.age > 18)
Declare
Count int;
Begin
Count := count+1;
End;
/
```

OUTPUT:

```
Trigger created Successfully.
```

Syntax 2:

```
DROP TRIGGER incr_count;
```

OUTPUT:

```
Trigger dropped successfully.
```

Topic 9 : Usage of cursors

IMPLICIT CURSORS:

Implicit cursors are automatically created whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Consider a relation student.

Student id	Student name	Student age
101	Abc	19
102	Xyz	20
103	Lmn	18

CURSOR CREATION:

```
DECLARE
  total_rows int

BEGIN
  UPDATE student
  SET age = age + 1;
  IF sql%notfound THEN
    dbms_output.put_line('no student selected');

  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' students selected ');
  END IF;
END;
/
```

OUTPUT:

```
3 students  selected.

PL/SQL procedure successfully completed.
```

```
SELECT * FROM student;
```

Student id	Student name	Student age
101	Abc	19
102	Xyz	20
103	Lmn	18

EXPLICIT CURSORS:

The syntax for creating an explicit cursor is –

```
CURSOR cursor_name IS select_statement;
```

```
DECLARE
  c_id student.student_id%type;
  c_name student.student_name%type;
  c_age student.student_age%type;

CURSOR c_student is
    SELECT student_id, student_name, student_age FROM student;
BEGIN
  OPEN c_student;
  LOOP
  FETCH c_student into c_id, c_name, c_age;
    EXIT WHEN c_student%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_age);
  END LOOP;
  CLOSE c_student;
END;
/
```

OUTPUT:

Student id	Student name	Student age
101	Abc	20
102	Xyz	30
103	Lmn	24