EXPERIMENT 3

3) Queries using Aggregate functions (COUNT, SUM, AVG, MAX and MIN), GROUP BY, HAVING and Creation and dropping of Views.

Solution:

Aggregate Functions

1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
4. MAX (A): The maximum value in the A column.
5. MIN (A): The minimum value in the A column.

1.Count the number of sailors.

```
SELECT COUNT (*)
FROM SAILORS S;
```

2.Count the number of different sailor names.

```
SELECT COUNT (DISTINCT S.sname );
```

3. Sum the rating of sailors

```
SELECT SUM (RATING)
FROM SAILORS;
```

4. Find the average age of all sailors.

```
SELECT AVG (S.age)
FROM Sailors S;
```

5. Find the average age of sailors with a rating of 10.

```
SELECT AVG (S.age)
FROM Sailors S
WHERE S.rating = 10;
```

6. Find the age of the oldest sailor.

```
SELECT MAX (AGE) FROM
SAILORS S;
```

6. Find the age of the youngest sailor.

```
SELECT MIN (AGE) FROM
SAILORS S;
```

7. Find the age of the youngest sailor for each rating level.

```
SELECT    S.RATING MIN (S.AGE)
FROM      SAILORS S
GROUP BY  S.RATING;
```

8. Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.

```
SELECT    S.RATING MIN (S.AGE) as MINAGE
FROM      SAILORS S
WHERE     S.AGE>=18
GROUP BY  S.RATING
HAVING    COUNT (*) > 1;
```

VIEWS:

A view is a SELECT statement with a name, stored in the database, and accessible as though it were a table

Views are useful for a variety of reasons.

One benefit is security.

For example, consider a typical scenario where you have a large table that contains a combination of some sensitive information along with some information that is of a general interest. You might create a view that queries the general interest columns of the large table, then grant privileges on that view to the general population of users. Those users may now query the view, and get direct access to the general information without having access to more sensitive data that exists in the underlying table. Views are a great way to mask certain data while giving access to other data in the same table.

Another benefit to views is their ability to make a complex query easier to work with.

For example, you might create a view that is built on a complex join, so that the complexity is built into the view. The result is a view object that appears to be a single table, which you may now query as though it were a table. You can even join the view with other tables and other views. In this situation, a view can be used to simplify the complexity of a commonly used join.

Creating Views

The syntax rules for creating a view are

- **The keywords CREATE VIEW**
- **The optional keywords OR REPLACE**
- **A name for the view, specified according to the rules for naming database objects**
- **The keyword AS**
- **Finally, a valid SELECT statement, with a few restrictions**

Example:

CREATE VIEW VW_EXPERTSAILORS AS

SELECT SID, SNAME, RATING

FROM SAILORS WHERE S.RATING>9;

If we execute this statement in SQL, we'll get the following message:

CREATE VIEW succeeded

Now that we have this view, we can work with it just as if it were a table.

For example, we can DESCRIBE it:

DESC VW_EXPERTSAILORS;

Dropping Views:

If we decide that we no longer need a view and want to destroy it (i.e. removing the definition of view) we can drop the view. A view can be dropped using the **DROP VIEW** command. To drop the Expert Sailors view.

DROP VIEW VW_EXPERTSAILORS**;**