# EXERCISE – VII

Expressing Queries using Aggregate Functions of SQL on Relational Database Schema

**SQL SERVER AGGREGATE FUNCTIONS**

SQL Server aggregate functions perform a calculation on a set of values and return a single value.  With the exception of the COUNT aggregate function, all other aggregate functions ignore NULL values.  Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement.

All aggregate functions are deterministic, which means they return the same value any time that they are called by using a specific set of input values

\* In order to express the aggregate functions we are using sailors, boat and reserves relational schema which was created in our lab exercise II.

* **SAILORS (*SID: INTEGER, SNAME: STRING, RATING: FLOAT, AGE: REAL*)**
* **BOATS (*BID: INTEGER, BNAME: STRING, COLOR: STRING*)**
* **RESERVES (*SID: INTEGER, BID: INTEGER, DAY: DATE*)**

1. **COUNT (A):-** The number of values in the A column.

COUNT (DISTINCT A): The number of unique values in the A column.

Examples:

1. To count number SIDs of sailors in Sailors table

SQL> SELECT COUNT (SID) FROM SAILORS;

1. To count numbers of boats booked in Reserves table.

SQL> SELECT COUNT (DISTINCT BID) FROM RESERVES;

1. To count number of Boats in Boats table.

SQL> SELECT COUNT (\*) FROM BOATS;

1. **SUM (A):-** The sum of all values in the A column.

SUM (DISTINCT A): The sum of all unique values in the A column.

Examples:

1. To find sum of rating from Sailors

SQL> SELECT SUM (RATING) FROM SAILORS;

1. To find sum of distinct age of Sailors (Duplicate ages are eliminated).

SQL> SELECT SUM (DISTINCT AGE) FROM SAILORS;

1. **AVG (A**):- The average of all values in the A column.

AVG (DISTINCT A): The average of all unique values in the A column.

Examples:

1. To display average age of Sailors.

SQL> SELECT AVG (AGE) FROM SAILORS;

1. To find average of distinct age of Sailors (Duplicate ages are eliminated).

SQL> SELECT AVG (DISTINCT AGE) FROM SAILORS;

1. **MAX (A):-** The maximum value in the A column.

MAX (DISTINCT A): The maximum of all unique values in the A column.

Examples:

1. To find age of Oldest Sailor.

SQL> SELECT MAX (AGE) FROM SAILORS;

1. **MIN (A):-** The minimum value in the A column.

MAX (DISTINCT A): The maximum of all unique values in the A column.

Examples:

1. To find age of Youngest Sailor.

SQL> SELECT MIN (AGE) FROM SAILORS;

Note that it does not make sense to specify DISTINCT in conjunction with MIN or MAX

(Although SQL does not preclude this).

1. **Write the following queries using Aggregate Functions.**
2. Find the average age of sailors with a rating of 10.
3. Count the number of different sailor names.
4. Find the name and age of the oldest sailor.
5. Count the number of Sailors.
6. Find the names of sailors who are older than the oldest sailor with a rating of 10.

# EXERCISE – VIII

Specify Queries on Relational Database Schema using GROUP BY, ORDER BY and HAVING Clause of SQL

1. **CLAUSE FUNCTIONS:**

**ORDER BY Clause:-** The ORDER BY keyword is used to sort the result-set by a specified column. The ORDER BY keyword sorts the records in ascending order by default (we can even use ASC keyword). If we want to sort the records in a descending order, we can use the DESC keyword. The general syntax is

* 1. **ORDER BY :**

SELECT ATT\_LIST FROM TABLE\_LIST ORDER BY ATT\_NAMES [ASC | DESC];

Examples:

1. Display all the sailors according to their ages.

SQL> SELECT \* FROM SAILORS ORDER BY AGE;

1. Display all the sailors according to their ratings (topper first).

SQL> SELECT \* FROM SAILORS ORDER BY RATING DESC;

1. Displays all the sailors according to rating, if rating is same then sort according to age.

SQL> SELECT \* FROM SAILORS ORDER BY RATING, AGE;

* + 1. **Write the following queries using ORDER BY Clause.**

1. To display names of sailors according to alphabetical order.
2. Displays all the sailors according to rating (Topper First), if rating is same then sort according to age (Older First).
3. Displays all the sailors according to rating (Topper First), if rating is same then sort according to age (Younger First).
4. Displays all the sailors according to rating (Lower Rating First), if rating is same then sort according to age (Younger First).
   1. **GROUP BY Clause**:- Group by is used to make each a number of groups of rows in a relation, where the number of groups depends on the relation instances. The general syntax is

**GROUP BY:**

SELECT [DISTINCT] ATT\_LIST FROM TABLE\_LIST WHERE CONDITION GROUP BY GROUPING\_LIST;

Examples:

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

SQL> SELECT S.RATING, MIN (S.AGE) FROM SAILORS S GROUP BY S.RATING;

* 1. **HAVING Clause** :- The extension of GROUP BY is HAVING clause which can be used to specify the qualification over group. The general syntax is

**HAVING:**

SELECT [DISTINCT] ATT\_LIST FROM TABLE\_LIST WHERE CONDITION GROUP BY GROUPING\_LIST HAVING GROUP\_CONDITIION;

Examples:

1. Find the age of youngest sailor with age >= 18 for each rating with at least 2 such sailors.

SQL> SELECT S.RATING, MIN (S.AGE) AS MINAGE FROM SAILORS S WHERE S.AGE >= 18 GROUP BY S.RATING HAVING COUNT (\*) > 1;

**# GROUP BY and HAVING** Clauses:-Thus far, we have applied aggregate operations to all (qualifying) rows in a relation. Often we want to apply aggregate operations to each of a number of groups of rows in a relation, where the number of groups depends on the relation instance. For this purpose we can use Group by clause.

432/a1

# EXERCISE – IX

Design and Development of Shipping Database and Expressing Nested Queries

1. **NESTED QUERIES**:- For retrieving data from the tables we have seen the simple & basic queries. These queries extract the data from one or more tables. Here we are going to see some complex & powerful queries that enables us to retrieve the data in desired manner. One of the most powerful features of SQL is nested queries. A nested query is a query that has another query embedded within it; the embedded query is called a sub-query.

A MySQL sub-query is a query nested within another query such as SELECT, INSERT, UPDATE or DELETE. In addition, a MySQL sub-query can be nested inside another sub-query.

A MySQL sub-query is called an inner query while the query that contains the sub-query is called an outer query. A sub-query can be used anywhere that expression is used and must be closed in parentheses.

**IN Operator**:- The IN operator allows us to test whether a value is in a given set of elements an SQL query is used to generate the set to be tested.

Example:- Find the names of sailors who have reserved boat 103.

SQL> SELECT S.SNAME FROM SAILORS S WHERE S.SID **IN** (SELECT R.SID FROM

RESERVES R WHERE R.BID = 103);

**NOT IN Operator**:- The NOT IN is used in a opposite manner to IN.

Example:- Find the names of sailors who have not reserved boat 103.

SQL> SELECT S.SNAME FROM SAILORS S WHERE S.SID **NOT IN** (SELECT R.SID FROM RESERVES R WHERE R.BID = 103);

**EXISTS Operator**:- This is a Correlated Nested Queries operator. The EXISTS operator is another set comparison operator, such as IN. It allows us to test whether a set is nonempty, an implicit comparison with the empty set.

Example:- Find the names of sailors who have reserved boat number 103.

SQL> SELECT S.SNAME FROM SAILORS S WHERE **EXISTS** (SELECT \* FROM RESERVES R WHERE R.BID = 103 AND R.SID = S.SID);

**NOT EXISTS Operator**:- The NOT EXISTS is used in a opposite manner to EXISTS.

Example:- Find the names of sailors who have not reserved boat number 103.

SQL> SELECT S.SNAME FROM SAILORS S WHERE **NOT EXISTS** ( SELECT \* FROM RESERVES R WHERE R.BID = 103 AND R.SID = S.SID );

**Set-Comparison Operators**:- We have already seen the set-comparison operators EXISTS, IN along with their negated versions. SQL also supports **op ANY** and **op ALL**, where **op** is one of the arithmetic comparison operators {<, <=, =, <>, >=, >}. Following are the example which illustrates the use of these Set-Comparison Operators.

**OP ANY Operator**:- It is a comparison operator. It is used to compare a value with any of element in a given set.

Example:- Find sailors whose rating is better than some sailor called Rajesh.

SQL> SELECT S.SID FROM SAILORS S WHERE S.RATING **> ANY** (SELECT S2.RATING FROM SAILORS S2 WHERE S2.SNAME = ' RAJESH ' );

**Note** that IN and NOT IN are equivalent to = ANY and <> ALL, respectively.

**ALL Operator**:- It is a comparison operator. It is used to compare a value with all the elements in a given set.

Example:- Find the sailor's with the highest rating using ALL.

SQL> SELECT S.SID FROM SAILORS S WHERE S.RATING **>= ALL** (SELECT S2.RATING FROM SAILORS S2);

Write following queries in SQL.

1) For each red boat; find the number of reservations for this boat.

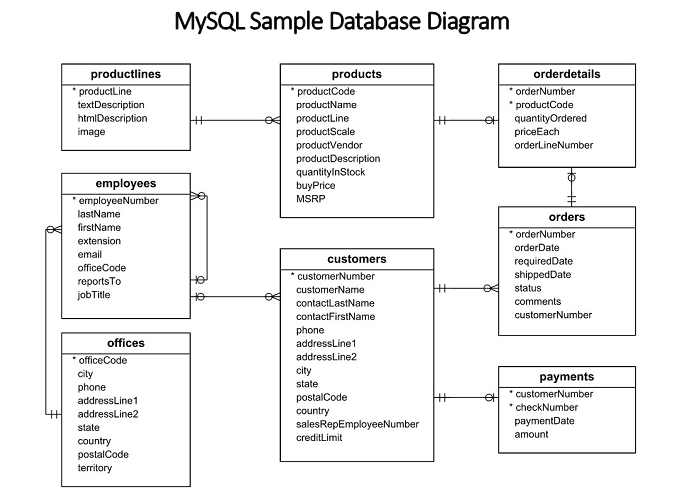
2) Find the average age of sailors for each rating level that has at least two sailors.

3) Find those ratings for which the average age of sailors is the minimum overall ratings.

## **MySQL Sample Database Schema**

The MySQL sample database schema consists of the following tables:

* **Customers**: stores customer’s data.
* **Products**: stores a list of scale model cars.
* **ProductLines**: stores a list of product line categories.
* **Orders**: stores sales orders placed by customers.
* **OrderDetails**: stores sales order line items for each sales order.
* **Payments**: stores payments made by customers based on their accounts.
* **Employees**: stores all employee information as well as the organization structure such as who reports to whom.
* **Offices**: stores sales office data.

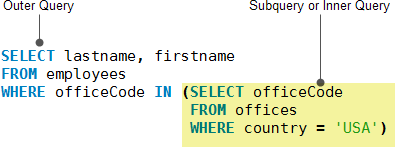


**Examples 2**

1. **SUB-QUERIES or COMPLEX QUERIES**
2. Employees who work in the offices located in the USA

SELECT lastName, firstName FROM employees WHERE officeCode

IN (SELECT officeCode FROM offices WHERE country = 'USA');



When the query is executed, the sub-query runs first and returns a result set. Then, this result set is used as an input of the outer query.

1. **MYSQL SUB-QUERY WITH COMPARISON OPERATORS**

You can use comparison operators e.g., =, >, <, etc., to compare a single value returned by the sub-query with the expression in the WHERE clause.

1. Find the customer who has the maximum payment.

SELECT customerNumber, checkNumber, amount FROM payments WHERE amount = (SELECT MAX(amount) FROM payments);

1. Find customers whose payments are greater than the average payment using a sub-query.

SELECT customerNumber, checkNumber, amount FROM payments WHERE amount > (SELECT AVG(amount) FROM payments);

1. **MYSQL SUBQUERY WITH IN AND NOT IN OPERATORS**
2. Find the customers who have not placed any orders

SELECT customerName FROM customers WHERE customerNumber NOT IN (SELECT DISTINCT customerNumber FROM orders);

1. **MYSQL SUB-QUERY IN THE FROM CLAUSE**
2. Finds the maximum, minimum and average number of items in sale orders

SELECT MAX(items), MIN(items), FLOOR(AVG(items)) FROM

(SELECT orderNumber, COUNT(orderNumber) AS items FROM orderdetails GROUP BY orderNumber) AS lineitems;

* Note that the FLOOR() is used to remove decimal places from the average values of item.

1. **MYSQL CORRELATED SUBQUERY**

Unlike a standalone sub-query, a correlated sub-query is a sub-query that uses the data from the outer query. In other words, a correlated sub-query depends on the outer query. A correlated sub-query is evaluated once for each row in the outer query.

1. Find the products whose buy prices are greater than the average buy price of all products in each product line.

SELECT productname, buyprice FROM products p1 WHERE buyprice >

(SELECT AVG(buyprice) FROM products WHERE productline = p1.productline)

1. **MYSQL SUBQUERY WITH EXISTS AND NOT EXISTS**
2. Selects sales orders whose total values are greater than 60K.

SELECT orderNumber, SUM(priceEach \* quantityOrdered) total FROM orderdetails INNER JOIN orders USING (orderNumber) GROUP BY orderNumber HAVING SUM(priceEach \* quantityOrdered) > 60000;

1. Find customers who placed at least one sales order with the total value greater than 60K by using the EXISTS operator (correlated subquery)

SELECT customerNumber, customerName FROM customers WHERE EXISTS

(SELECT orderNumber, SUM(priceEach \* quantityOrdered) FROM orderdetails INNER JOIN orders USING (orderNumber) WHERE customerNumber = customers.customerNumber GROUP BY orderNumber HAVING SUM(priceEach \* quantityOrdered) > 60000);