C11. SQL Data Aggregation

The purpose of data aggregation is to extract global information or to summarize a set of data. SQL aggregate functions are functions that take a collection of values as input and return a single value.

Main characteristics of the aggregation functions:

- they operate on a single column or expression from a group of rows;
- they return a single value for a group of rows
- could be used only in the SELECT list and/or in the HAVING clause;
- they produce an anonymous result, therefore it is necessary to use an alias to name the result column.

Most aggregate functions (except MIN, MAX, COUNT) work on a single column or expression of numeric data. Projection list may contain only aggregation functions and distinct values per set.

The SQL standard defines the following functions:

- COUNT: the function returns the number of rows from the result set;
- SUM: the function returns the sum of the entries in a column or expression;
- AVG: the function returns the average entry in a column;
- MIN, MAX: the function return the minimum / maximum of a column/expression value;
- VARIANCE: the function measures how far a set of numbers is spread out from the mean value of an expression;

- STDDEV: the function returns the sample standard deviation of a set of numerical values.

11.1. COUNT

Returns the number of values for a specified column or expression. DISTINCT can be used to avoid counting duplicate values. Default, all values are considered, excepting NULL. The marker '*' can be used to counts all the result entries regardless of whether nulls or the duplicate occur.

The general syntax for count is:

```
SELECT COUNT( [DISTINCT] exp)
                        AS nrOfSomething
        FROM Relation …
        WHERE condition …;
```

The expression could be by any type or could be the '*' marker. The COUNT function can be used also over join result.

ACTIVITY 1: Using SQL Workshop -> SQL Commands run the following SQL command that find the number of sailors younger than 30:

```
SELECT COUNT(*) AS NrOfYoungSailors
        FROM Sailors
        WHERE age<30;
```

Modify the query in order to consider just the young sailors (<30) that reserved sometimes a 'red' Boat.

11.2. SUM

Returns the sum of the values in a specified column and ignores the NULL values. However, if all values are NULL, the function will return NULL.

The general syntax is:

```
SELECT SUM(exp) AS totalExp
    FROM Relation …
    WHERE condition …;
```

The expression has to be numerical. The function could be applied also over joins.

ACTIVITY 2: Using SQL Workshop -> SQL Commands run the following SQL command to compute the sum of ages for the rank three sailors:

```
SELECT SUM(age) AS sumOfAges
        FROM Sailors
        WHERE rank=3;
```

11.3. AVG.

The AVG function returns the average of the values of a specified column or expression. It also ignores the NULL values. If all values are NULL the function will return NULL.

The syntax for SQL AVG is:

```
SELECT AVG(exp) AS avgExp
FROM Relation …
WHERE condition …;
```

ACTIVITY 3: Using SQL Workshop -> SQL Commands run the following SQL command in order to compare the result of computing average by using SUM/COUNT as an alternative of AVG:

```
SELECT SUM(age)/COUNT(*) AS sumCountAvg,
       AVG(age)  AS avgAvg
FROM Sailors;
```

Note: What happed in case of NULL values? Modify the query in order to obtain the same result.

11.4. MIN and MAX.

The MIN function returns the smallest value of a column or expression. On the contrary, the MAX function returns the largest value of a column or expression. Both ignore the NULL values. If all values are NULL, both of them will return NULL.

In Oracle string (characters) comparison is case sensitive by default.

The syntax for SQL MIN/MAX is:

```
SELECT MIN(exp1) AS minExp1,
       MAX(exp2) AS maxExp2
FROM Relation …
WHERE condition …;
```

ACTIVITY 4: Using SQL Workshop -> SQL Commands run the following SQL command in order to find the ages of the youngest and oldest sailors:

```
SELECT MIN(age) AS ageOfYoungestSailor,
       MAX(age) AS ageOfOldestSailor
FROM Sailors;
```

Note: What happed in case of NULL values? Modify the query using 'case when' in order to consider 0 for NULL values.

11.5. VARIANCE and STDDEV.

The VARINACE function returns the variance of a set of values of a column or expression. It computes the average of the squared differences from the mean. In other words, it measures how far a set of numbers is spread out around the mean. The STDDEV function extends the previous function and allows obtaining more depth information about data. Mathematically, it computes the squared root of the variance. The result can be used to define the "normality" around the mean of a set of values.

Both functions return zero if the result contains just one row. They also ignore NULL values, but if all values are NULL, they will return NULL.

The general syntax is:

```
SELECT VARIANCE(exp1) AS varExp1,
       STDDEV(exp2) AS stddevExp2
FROM Relation …
WHERE condition …;
```

ACTIVITY 5: Using SQL Workshop -> SQL Commands run the following SQL statement in order to find variance and standard deviation for the age column:

```
SELECT VARIANCE(age) AS varAge,
       STDDEV(age) AS stddevAge
FROM Sailors;
```

ACTIVITY 6: Extend the previous query in order to find the "normal" ages interval for sailors.

11.6. Grouping data.

In many situations the summarization needs to be applied on specific group of entities. The GROUP BY clause extend the SQL SELECT command by adding the ability of grouping the data from the result and produce a single summary row for each group. In this case, the projection list has to contain only aggregation functions and expressions unique over each group.

The GROUP BY clause could contains more than one criterion, but they are not computed in a hierarchical manner. However, the result can be interpreted in some hierarchical style by using ORDER BY.

A special group will be created for NULL values.

To add additional fields to projection list, they must be declared invariant by including them in the GROUP BY clause.

The HAVING clause could be used in conjunction with GROUP BY in order to discard or restrict the groups that appear in the final result table.

The general syntax is:

```
SELECT group_by_expressions,
          aggregation_functions
    FROM Relation …
    WHERE selection_condition …
    GROUP BY exp1, exp2 …
    HAVING conditions;
```

ACTIVITY 7: Using SQL Workshop -> SQL Commands run the following SQL command in order to find the number of sailors on each rank, for all ranks that have more than one sailor:

```
SELECT rank, COUNT(*) AS NrSail
    FROM Sailors
    GROUP BY rank
    HAVING COUNT(*)>1;
```

ACTIVITY 8: Using SQL Workshop -> SQL Commands create a query that find the sailors older than the average age of sailors.

ACTIVITY 9: Create an interactive report based on aggregation functions to display all sailors who have reserved more than *N* boats. For reading the *N* please add one input field (noOfReservedBoats) and a validation button to the report.