SQL: Structured Query Language
Aggregate Operations

Aggregate Operations

Aggregate database operations are operations that return individual values (aggregates) instead of tuples. These aggregates can be used in the SELECT clause of the SELECT statement.

There are several aggregate operations in SQL/MySQL:

<table>
<thead>
<tr>
<th>Aggregate Operation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT()</td>
<td>returns the number of rows</td>
</tr>
<tr>
<td>AVG()</td>
<td>returns the average value in the column</td>
</tr>
<tr>
<td>MIN()</td>
<td>returns the minimum value in the column</td>
</tr>
<tr>
<td>MAX()</td>
<td>returns the maximum value in the column</td>
</tr>
<tr>
<td>SUM()</td>
<td>returns the sum of the values in the column</td>
</tr>
<tr>
<td>STD()</td>
<td>returns the population standard deviation for the values in the column</td>
</tr>
<tr>
<td>STDEV_SAMP()</td>
<td>returns the sample standard deviation for the values in the column</td>
</tr>
<tr>
<td>VAR_POP()</td>
<td>returns the population standard variance for the values in the column</td>
</tr>
<tr>
<td>VAR_SAMP()</td>
<td>returns the sample standard variance for the values in the columns</td>
</tr>
<tr>
<td>GROUP_CONCAT()</td>
<td>concatenates all string values in the column</td>
</tr>
</tbody>
</table>

**AVG, SUM, MAX, MIN, STD, STDEV_SAMP, VAR_POP, VAR_SAMP.**

For these operations the syntax is

- AVG(<Expression>)
- SUM(<Expression>)
- MAX(<Expression>)
- MIN(<Expression>)
- STD(<Expression>)
- STDEV_SAMP(<Expression>)
- VAR_POP(<Expression>)
- VAR_SAMP(<Expression>)

Here, <Expression> is either a column name from any of the tables found in the FROM clause of the SELECT statement, or a SQL expression that uses only column names from those tables (plus any other operations and built-in functions). The <Expression> is expected to return a numeric value (i.e., the columns referenced in it are supposed to have numeric types).
E.g.

SELECT AVG(Salary) FROM Employee;

returns the average of the Salary field in a relational table Employee(ID, Name, Salary, Dept). Similarly,

SELECT SUM(Salary + Bonus) FROM Employee;

returns the sum of Salary + Bonus (think - total compensation) for all records in the same relational table.

COUNT. COUNT has more interesting syntax. The two traditional forms of the COUNT() operator are

- `COUNT(*)`
- `COUNT([DISTINCT] <ColumnName>)`

`COUNT(*)` form counts the number of tuples returned by the rest of the query.

SELECT COUNT(*) FROM Employee;

returns the number of employee records.

If we replace the * with a column name:

SELECT COUNT(Name) FROM Employee;

the result of the query is the number of rows in the Employee table that have a non-NULL value in the Name column. For table instances w/o NULL values this is equivalent to `SELECT COUNT(*)`, but if a column has NULL values, the results of these two queries are different.

The DISTINCT qualifier allows us to count only unique values found in the column:

SELECT COUNT(DISTINCT Dept) FROM Employee;

returns the number of different departments recorded in the Employee table.

GROUP_CONCAT. The GROUP_CONCAT aggregation operation has the following syntax:

GROUP_CONCAT( [DISTINCT] <Expression> [, <Expression>, ...] [ORDER BY <order-list>] [SEPARATOR <string>] )

Here, <Expression> is either a column name of a column from one of the tables found in the FROM clause, or a SQL expression that uses only columns from the tables in the FROM clause and/or SQL operations and built-in functions. The <Expression> can has any return type, but the GROUP_CONCAT operation converts all values into strings.
The result of this operation is a string that concatenates all values of the `<Expression>` across the rows of the table constructed in the FROM and WHERE clauses of the SQL statement.

The optional SEPARATOR parameter controls the string value that separates the values of the expression in the result. By default they are separated by a comma (`,`) but it can be overridden.

The optional DISTINCT keyword specifies that duplicate values shall be eliminated from the output (i.e., each value is concatenated only once).

The optional ORDER BY clause specifies the order in which the values are concatenated. By default they are concatenated in whatever order the rows of the table are processed (and thus may be somewhat non-deterministic). ORDER BY clause has the same syntax as the ORDER BY clause in the SELECT statement and establishes the order in the same way.

E.g., the following query:

```
SELECT GROUP_CONCAT(Name, ORDER BY Dept, Name SEPARATOR ', ') 
FROM Employee 
WHERE Salary > 50000;
```

returns in one row the string that concatenates the names of all employees from the Employee table with salary higher than $50,000 separated by a comma-and-a-space (`, `) and ordered in alphabetical order by department name, and then by last name within each department.

The query

```
SELECT GROUP_CONCAT(DISTINCT Dept ORDER BY Dept) 
FROM Employee;
```

returns a concatenated list of department names ordered alphabetically.

**Other operations that use DISTINCT.** Several other aggregate operations can use DISTINCT keyword. The full list of operations is below:

- `COUNT(DISTINCT <ColumnName>)`
- `AVG(DISTINCT <Expression>)`
- `MAX(DISTINCT <Expression>)`
- `MIN(DISTINCT <Expression>)`
- `SUM(DISTINCT <Expression>)`

In all cases, presence of the DISTINCT keyword triggers the duplicate elimination operation before the aggregation operation takes place. Note that for MAX and MIN the presence of DISTINCT is redundant, while for other operations, this keyword can affect the actual output.