Aggregate Functions

Aggregate Functions

- Also called set functions
- They operate on group of values to produce a single, summarizing value
- Aggregates are applied to a set of rows that can be:
  - All the rows in a table
  - Only those rows specified by a `WHERE` clause
  - Those rows created by a `GROUP BY` clause

Non-aggregate queries process the rows one by one. Each row is processed independently and put into the result.

Aggregate queries do something completely differently – it takes a tables as a whole and constructs new rows from it.

Aggregate Functions

- We can count the number of records that match a certain criteria –
- `COUNT()` – the number of rows in a table
- `COUNT(value)` – the number of non-null values in *value*

Works with all datatypes

- We can find the lowest (minimum) value or the highest (maximum) value –`MIN(value)` / `MAX(value)`

Works with character, numeric and datetime

- We can add the values in a specific column that match a certain criteria—`SUM(value)`
- We can calculate the average -- `AVG(value)`

Only work with Numeric Types

All aggregate functions except Count(value) ignore NULLS

Aggregate Functions

- Aggregates return new result sets
- The result sets have no defined name
- The DBMS will return a name that is defined in the SELECT clause

```
SELECT COUNT( DISTINCT price )
FROM titles
```

COUNT(DISTINCT price)
10

- Use **ALIAS** to provide more meaningful names to your result set

Aggregate functions

- Aggregate functions ignore NULLS!
- An aggregate function can **NOT** appear in a WHERE clause

Aggregate

```
SELECT title_id
FROM books
WHERE page_count = MAX (page_count);
```

- You can **NOT** mix non-aggregate with aggregate in a SELECT clause

Non-Aggregate Aggregate

```
SELECT title_id, MAX(page_count)
FROM books;
```

Aggregate functions

- You can **NOT** nest aggregate functions

Aggregate Nested Aggregate

```
SELECT SUM(AVG(sales))
FROM titles;
```

- You can not use subqueries in aggregate expressions

Aggregate

```
SELECT AVG(SELECT price FROM titles);
```

Subquery

Combining Aggregate Functions

- SELECT statements **can** include as many of the aggregate functions as required

```
SELECT
COUNT(*) AS 'Num_Books',
AVG(price) AS 'Avg_Price',
MAX(price) AS 'Highest_Price'
FROM titles;
```

Min & Max

MIN

- Returns the minimum value of the selected column
- It works with **character**, **numeric** and **datetime** types

```
SELECT
MIN(numberOfPackages)
FROM orders;
```

MIN(numberOfPackages)
1

```
SELECT MIN(shippedto)
FROM orders;
```

MIN(shippedto)
Altenwerth and Sons

```
SELECT MIN(dateOrdered)
FROM orders;
```

MIN(dateOrdered)
2020-03-05 13:32:21

- MIN works with character, numeric and datetime datatypes
- DISTINCT has no meaning in MIN

MAX

- Finds the maximum value
- It works with character, numeric and datetime types

```
SELECT
MAX(numberOfPackages)
FROM orders;
```

MAX(numberofpackages)
4

```
SELECT MAX(shippedTo)
FROM orders;
```

MAX(shippedTo)
Zulauf, Hahn and Legros

```
SELECT MAX(dateOrdered)
FROM orders;
```

MAX(dateOrdered)
2021-02-26 05:41:15

- MAX works with character, numeric and datetime datatypes
- DISTINCT has no meaning in MAX

Average

AVG

- Finds the average of a set of values
- The result is at least as precise as the most precise datatype being averaged

```
SELECT AVG(price)
FROM carInventory
```

AVG(price)
30425.200940

```
SELECT AVG(price)
FROM carInventory
WHERE make = 'Mazda';
```

AVG(price)
31345.935714

AVG ***ONLY*** works on numeric data types

Average of no rows is null – NOT zero

SUM

SUM

- Works only with numeric types

```
SELECT SUM(price)
FROM carInventory
WHERE qtyInStock = 1;
```

SUM(price)
4824371.14

- SUM() works only on numeric data types
- SUM of no rows is null (NOT zero)
- Columns with NULL values are ignored

Counting

COUNT

- Count the number of rows in a set of values

```
SELECT COUNT(make)
FROM carInventory;
```

COUNT(make)
1000

```
SELECT COUNT(make)
FROM carInventory
WHERE make = 'Porsche';
```

COUNT(make)
11

`COUNT(value)` returns the number of rows in value that **ARE NOT NULL** and returns an integer greater than or equal to zero

`COUNT(*)` returns the count of all rows ***including nulls*** and duplicates and returns an integer greater than or equal to zero

```
SELECT COUNT(*)
FROM carInventory
```

Distinct and Aggregate Functions

DISTINCT

- Can be used to eliminate duplicates
- Works with SUM, AVG, and COUNT – not meaningful with MIN, MAX
- Can't use with COUNT(*)- can only use with a named column

ALL argument is the default – does not have to specified

```
SELECT COUNT(make)
FROM carInventory;
```

COUNT(make)
1000

```
SELECT COUNT(DISTINCT make)
FROM carInventory;
```

COUNT(DISTINCT make)
58

Group By

GROUP BY

- **Used with aggregate functions** to group the result set
- Can divide the table into logical groups or categories and perform calculations for each group

```
SELECT make, COUNT(*) as numberOf
FROM carInventory
GROUP BY make;
```

make	numberOf
Acura	21
Aptera	1
Aston Martin	5
Audi	25
Bentley	12
BMW	37

GROUP BY

- The GROUP BY clause is used for aggregation of a result set
- The database returns rows of information that is filtered by the where clause – this result is set is then grouped by the values of one or more columns.
- The GROUP BY lists the columns that are used to determine the groups
- The returned (grouped) rows consists of the information from one or more rows – these are produced rows
- A grouped row is a new row representing each group of rows found during aggregation

GROUP BY

```
SELECT make, SUM(price),COUNT(*) as numberOf
FROM carInventory
GROUP BY make;
```

make	SUM(price)	numberOf
Acura	689700.13	21
Aptera	25580.07	1
Aston Martin	176896.22	5
Audi	743231.61	25
Bentley	315743.38	12
BMW	990397.44	37

GROUP BY

- Gathers all of the information from the FROM clause
- Filters the information by the WHERE clause (if present)
- Aggregates the remaining rows into groups.
- Only group rows can be used in the SELECT clause

GROUP BY

- The GROUP BY clause comes **AFTER** the WHERE clause and **BEFORE** the ORDER BY clause
- No columns can appear in the SELECT clause UNLESS they are also included in the GROUP BY clause

```
SELECT make, modelYear, COUNT(*) as numberOf
FROM carInventory
GROUP BY make;
```

```
SELECT make, modelYear, COUNT(*) as numberOf
FROM carInventory
GROUP BY make, modelYear;
```

Having Clause

MAD 202

Filtering Groups

- SQL provide you with a method to filter the returned grouping set
- The HAVING clause is similar to the WHERE clause
- It is optional
- The difference between a WHERE clause and HAVING is that the WHERE clause filters data before it is grouped and the HAVING **filters after**
  - Remember – rows removed by the WHERE clause will not be available
  - Some DBMS treat the WHERE and HAVING clause the same if no grouping is being performed.

However – use the HAVING clause only when using GROUP BY

HAVING clause

- Further filters the GROUP BY results

```
SELECT type, SUM(sales), COUNT(type)
FROM titles
GROUP BY type;
```

make	SUM(price)	numberOf
Acura	689700.13	21
Aptera	25580.07	1
Aston Martin	176896.22	5
Audi	743231.61	25
Bentley	315743.38	12
BMW	990397.44	37

```
SELECT make, SUM(price),COUNT(*) as numberOf
FROM carInventory
GROUP BY make
HAVING SUM(price) > 2000000;
```

make	SUM(price)	numberOf
Chevrolet	2450457.59	82
Ford	2245622.42	75