Ordering, Limiting, Grouping, and Aggregate functions

Ordering Data

A result set can be sorted using the **`ORDER BY`** syntax

- Sort columns must exist in the table being queried or can be aliased columns

```
SELECT census_region, state_name FROM state
ORDER BY census_region
```

Ordering Data

A result set can be sorted using the **`ORDER BY`** syntax

- Sort columns must exist in the table being queried or can be aliased columns
- Multiple column names can be provided which assigns a priority sort

```
SELECT census_region, state_name, population
FROM state ORDER BY census_region, population
```

Ordering Data

A result set can be sorted using the **`ORDER BY`** syntax

- Sort columns must exist in the table being queried or can be aliased columns
- Multiple column names can be provided which assigns a priority sort.
- Each column in **`ORDER BY`** clause can be specified as ascending (**`ASC`**) or descending (**`DESC`**). If not specified the default is **`ASC`**.

```
SELECT census_region, state_name, population
FROM state ORDER BY census_region, population DESC;
```

Limiting Results

We can reduce the size of our result set to N results. By using **`LIMIT N`** at the end of a query. Where **`N`** is the result size.

```
SELECT state_name FROM state LIMIT 10;
```

String Operations

We can concatenate the values across multiple columns into a single field using the `||` operator to concatenate strings.

```
SELECT city_name || ', ' || state_abbreviation
FROM city WHERE state_abbreviation IN
('PA', 'OH', 'NY');
```

String Operations

We can use the **AS** keyword to give the concatenated column a name.

```
SELECT city_name || ', ' || state_abbreviation
AS city_state FROM city
WHERE state_abbreviation IN
('PA', 'OH', 'NY');
```

Grouping Results

In order to use aggregate functions such as **`SUM`**, data must be grouped. Grouping data is the process of combining columns with common values.

For example if our database contains city information, including state abbreviation and population, we can report the total population in each state abbreviation by using **`SUM`** and **`GROUP BY`** the state abbreviation .

Grouping Results

`GROUP BY` statements group records into summary rows and return one record for each group.

- The **`GROUP BY`** clause can be used in conjunction with a **`SELECT`** statement and aggregate functions to collect data across multiple records.

```
SELECT state_abbreviation, SUM(population)
FROM city GROUP BY state_abbreviation
```

Aggregate Functions

- **`AVG`** returns the average value of a numeric column
- **`SUM`** returns the total sum of a numeric column
- **`COUNT`** returns the number of rows matching criteria
- **`MIN`** returns the smallest value of the selected column
- **`MAX`** returns the largest value of the selected column

Count(column) Vs. Count(*)

It's important to be aware of the **difference between `COUNT(*)` and `COUNT`ing a specific field**.

If you specify a column name to count, only the rows in the table that have a value for that column will be returned. For instance, in our state data, this query returns 51 rows, while changing it to use **`COUNT(*)`** instead returns 56. This is because only 51 rows have a value (that is do not have a **`NULL`** value) for **`state_nickname`**.

```
SELECT COUNT(state_nickname) FROM state;
```

Subqueries

A **subquery** is referred to as an inner query and can provide the results of one query as input to another.

- Often used in the WHERE clause
- Can only return one item in the SELECT clause when using = in main query so you must be sure you will only get **one result** or use **IN** in your **WHERE** clause.

```
SELECT city_name, state_abbreviation FROM city
WHERE state_abbreviation IN
(SELECT state_abbreviation FROM state WHERE
census_region = 'Northeast');
```

Query Execution Order

SQL is a declarative language and does *not* run from top to bottom and left to right. The order it runs is:

1. **`FROM`** clause - The database needs to know which table you're selecting from first of all
2. **`WHERE`** clause - The database then needs to know which rows you'll work with
3. **`GROUP BY`** clause - The database then groups those rows according to your GROUP BY clause
4. **`SELECT`** clause - The database then collapses those rows down and selects the columns that you want data from
5. **`ORDER BY`** clause - The database orders the rows in the order that you ask for
6. **`LIMIT / TOP`** clause - The database only returns the number of resulting rows that you want