Module 2-2

Aggregate Functions

Additional SELECT options

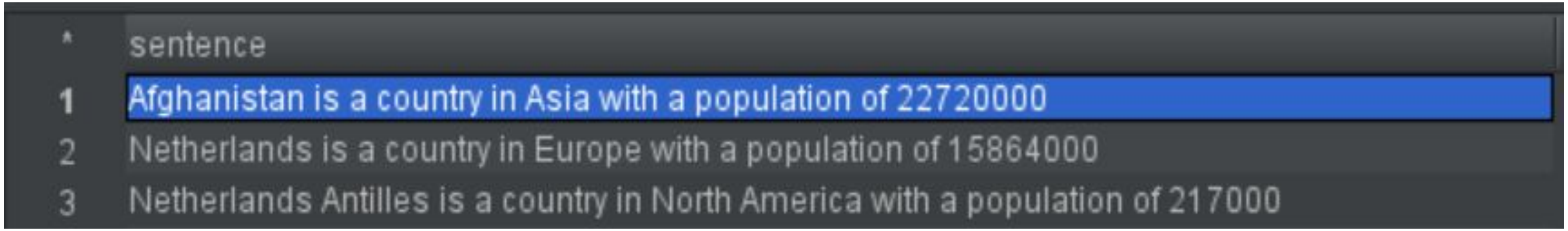
Data Concatenation

Several columns can be concatenated into a single derive column using || .

- Consider the following example:

```
SELECT name || ' is a country in ' || continent || ' with a population of ' || population AS sentence
FROM country;
```

- The first three rows of output:



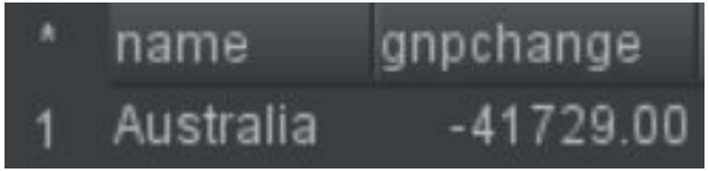
```
*   sentence
1   Afghanistan is a country in Asia with a population of 22720000
2   Netherlands is a country in Europe with a population of 15864000
3   Netherlands Antilles is a country in North America with a population of 217000
```

Absolute Value

The absolute value can be calculated by using the ABS(...) function.

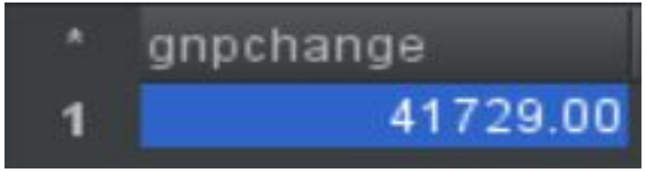
- Consider the following example:

```
SELECT gnp - gnpold AS gnpchange
FROM country;
```



```
*   name        gnpchange
1   Australia   -41729.00
```

```
SELECT ABS(gnp - gnpold) AS gnpchange
FROM country;
```



```
*   gnpchange
1    41729.00
```

Note that the results will never be negative now.

Limiting Results

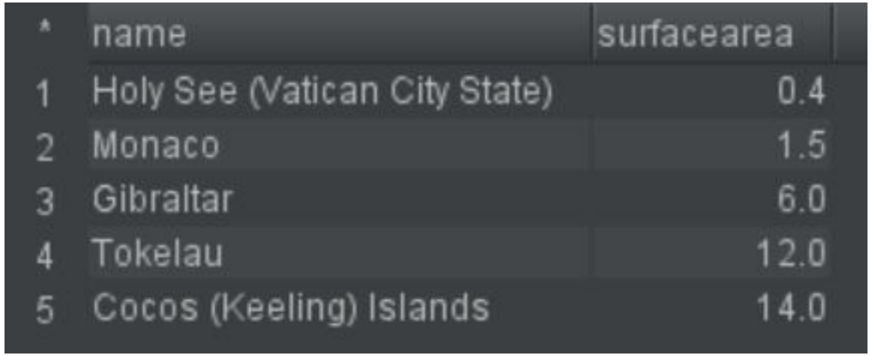
You can limit the number of rows from your query with **LIMIT x**. For instance LIMIT 10 limits the results to 10 rows.

This tends to work best with ORDER BY as it allows you to construct lists like “top 10 of…”

Limiting Results Example

The following query gives you the “top 5” smallest countries by surface area:

```
SELECT name, surfacearea
FROM country
ORDER BY surfacearea ASC
LIMIT 5;
```



*	name	surfacearea
1	Holy See (Vatican City State)	0.4
2	Monaco	1.5
3	Gibraltar	6.0
4	Tokelau	12.0
5	Cocos (Keeling) Islands	14.0

Sorting & Aggregating

Sorting

- In SQL, sorting is achieved through the ORDER BY statement:

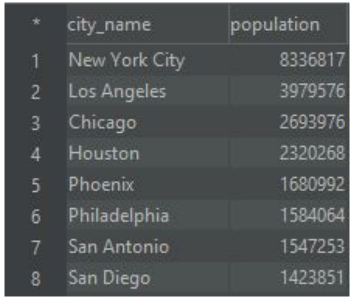
  ORDER BY [name of column] [direction]

- The ORDER BY section goes after the WHERE statement.
- You need to specify which column you want to sort by.
- You can optionally specify the direction of the sort:
  - **ASC** for ascending
  - **DESC** for descending.

Sorting Example

Consider the following example:

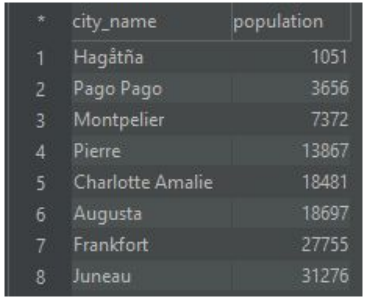
```
SELECT city_name, population
FROM city
ORDER BY population DESC;
```



*	city_name	population
1	New York City	8336817
2	Los Angeles	3979576
3	Chicago	2693976
4	Houston	2320268
5	Phoenix	1680992
6	Philadelphia	1584064
7	San Antonio	1547253
8	San Diego	1423851

Note that the records are now sorted in descending order with the largest population cities appearing first.

```
SELECT city_name, population
FROM city
ORDER BY population ASC;
```



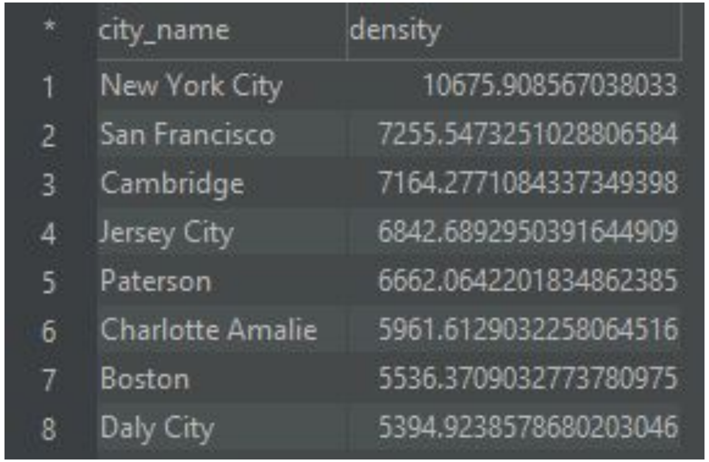
*	city_name	population
1	Hagåtña	1051
2	Pago Pago	3656
3	Montpelier	7372
4	Pierre	13867
5	Charlotte Amalie	18481
6	Augusta	18697
7	Frankfort	27755
8	Juneau	31276

Note that the records are now sorted in ascending order with the smallest population cities appearing first.

Sorting Example with Derived Fields

You can also sort by derived fields. Consider the following example:

```
SELECT city_name, population/area AS density
FROM city
ORDER BY density DESC;
```



*	city_name	density
1	New York City	10675.908567038033
2	San Francisco	7255.5473251028806584
3	Cambridge	7164.2771084337349398
4	Jersey City	6842.6892950391644909
5	Paterson	6662.0642201834862385
6	Charlotte Amalie	5961.6129032258064516
7	Boston	5536.3709032773780975
8	Daly City	5394.9238578680203046

Sorting, default behavior

If no direction is provided (ASC or DESC), the sort order is **assumed to be ascending**. The following 2 queries produce identical results:

```
SELECT city_name, population
FROM city
ORDER BY population ASC;
```

```
SELECT city_name, population
FROM city
ORDER BY population;
```

Let’s write some SQL!

Aggregate Functions

Aggregate data can be created by combining the value of one or more rows in a table. For example:

- The total population of the United States
- The total population of all cities in Ohio
- The average population of all cities
- The state with the most population

Aggregate Functions

Here are commonly used aggregate functions:

- **COUNT**: Provides the number of rows that meet a given criteria
- **MAX / MIN**: The maximum or minimum value of a column
- **AVG**: The average value of a column
- **SUM**: The sum of a column

Aggregate Functions: Count Example

The following are two examples for COUNT:

```
SELECT COUNT(*)
 FROM city;
```

Returns the total row count for city.

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

Returns the total number of values for census_region (note that there are null values, so this count will be less than the total row count).

```
SELECT COUNT(*)
FROM state
WHERE census_region = ‘South’
```

Returns the row count for all rows having a census_region value of “South”

Aggregate Functions: MAX/MIN example

Here we have examples of the MAX and MIN function:

```
SELECT MAX(area)
FROM city;
```

Returns the maximum surface area encountered in the whole table.

```
SELECT MIN(area)
FROM city;
```

Return the minimum surface area encountered in the whole table.

Aggregate Functions: AVG example

The following is an example of AVG:

```
SELECT AVG(population)
FROM city;
```

Returns the average population of all cities on the city table.

Aggregate Functions: SUM example

The following is an example of SUM:

```
SELECT SUM(population)
from state;
```

This is the total population of the United States

Aggregate Functions: Group By

The previous examples illustrated how to apply the aggregate functions to the entire table, but what if we wanted to apply the aggregate functions only to subsets of the data?

- In order to do this, we introduce the concept of aggregating (or grouping) which is achieved through the SQL command **GROUP BY**.

  GROUP BY [name of column]

- The GROUP BY section goes **<u>before</u>** the ORDER BY section.

Aggregate Functions: Group By Example

Suppose you wanted to find out the sum of the population for each census region. Logically, if you did this manually you might have broken this process up into two steps:

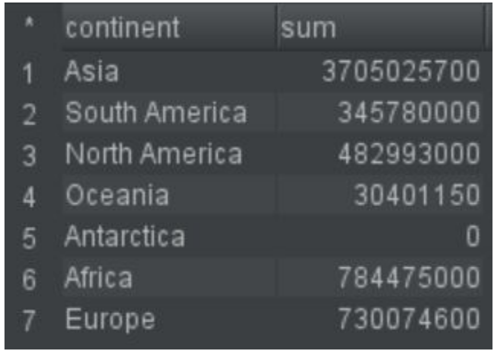
1. Group all the rows into 5 groups, one for each census region.
2. For each group, sum up the population

You end up with 5 numbers, the population count for each of the five census regions.

Aggregate Functions: Group By Example

Just like how you would break up this process in two steps if done manually, SQL requires two elements to successfully aggregate this data:

```
SELECT state_name,
SUM(population)
FROM state
GROUP BY census_region
```



	continent	sum
1	Asia	3705025700
2	South America	345780000
3	North America	482993000
4	Oceania	30401150
5	Antarctica	0
6	Africa	784475000
7	Europe	730074600

Treat all rows with the same census_region value as part of the same “bucket” of data.

Now add all the population values for each bucket.

Let’s write some SQL!