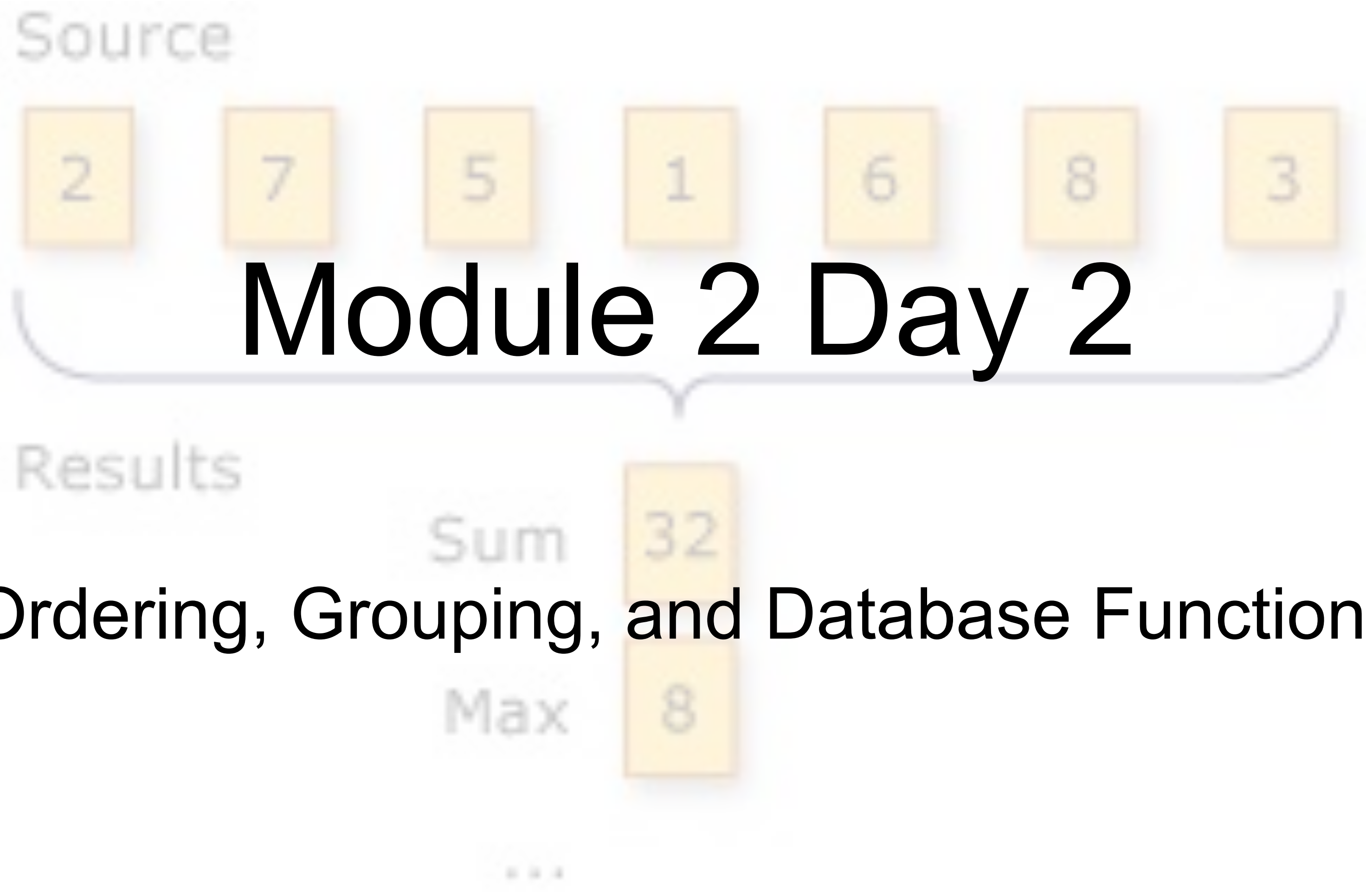


Module 2 Day 2

Ordering, Grouping, and Database Functions

Module 2 Day 2
Can you construct statements that … ?

- … Order Results
- … Limit Results
- … Perform String operations/functions
- … Perform Aggregate functions
- … Group Results
- … Use Subqueries

Additional SELECT options

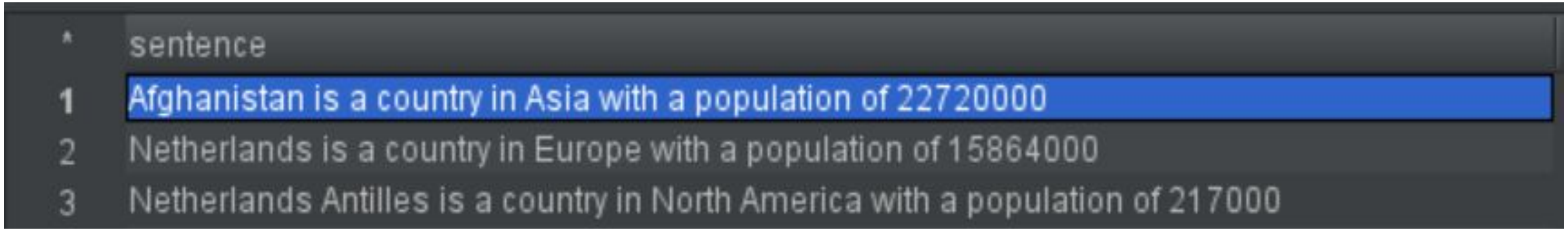
Data Concatenation

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

- Using the country data to build a sentence:

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

- The first three rows of output would be:

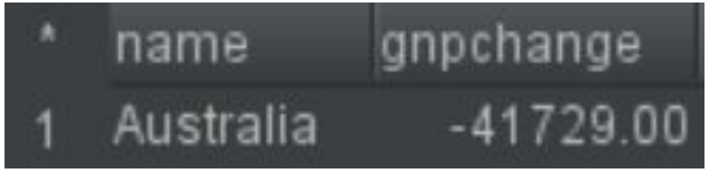


```
    sentence
1   Afghanistan is a country in Asia with a population of 22720000
2   Netherlands is a country in Europe with a population of 15864000
3   Netherlands Antilles is a country in North America with a population of 217000
```

Functions: Absolute Value

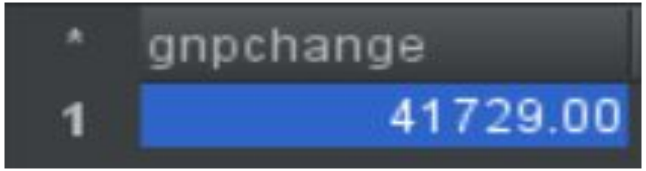
An absolute value can be calculated by using the ABS(...) function by passing in the column for which the absolute value is to be calculated.

```
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

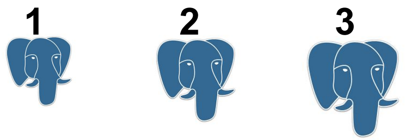
With ABS, the results will never be negative and represent the value of change.

Sorting

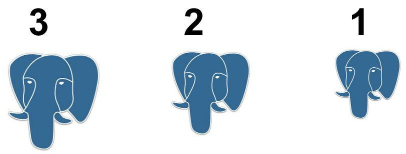
- In SQL, sorting is done using the ORDER BY statement:

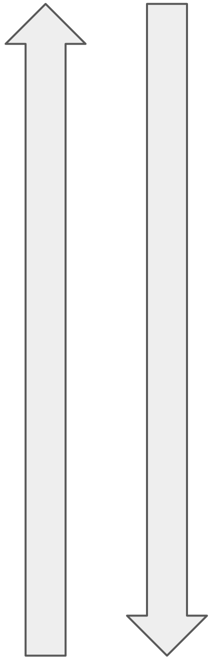
  ORDER BY [name of column] [direction]

- The ORDER BY clause goes after the WHERE clause.
- You need to specify which column(s) you want to sort by.
- You may specify the direction of the sort:
  - **ASC** for ascending

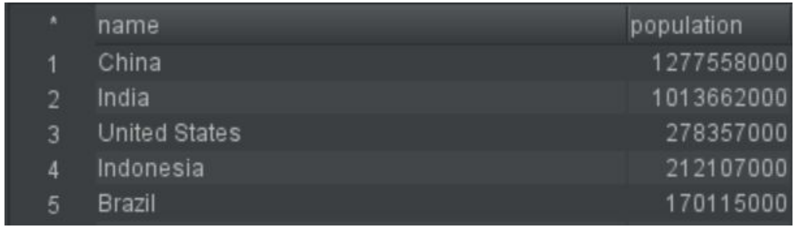
  - **DESC** for descending.



Sorting Example

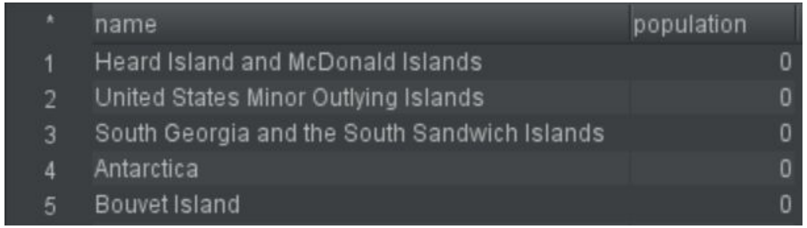
```
SELECT name, population
FROM country
ORDER BY population DESC;
```



*	name	population
1	China	1277558000
2	India	1013662000
3	United States	278357000
4	Indonesia	212107000
5	Brazil	170115000

The records are now sorted in descending order, with the largest population countries appearing first.

```
SELECT name, population
FROM country
ORDER BY population ASC;
```



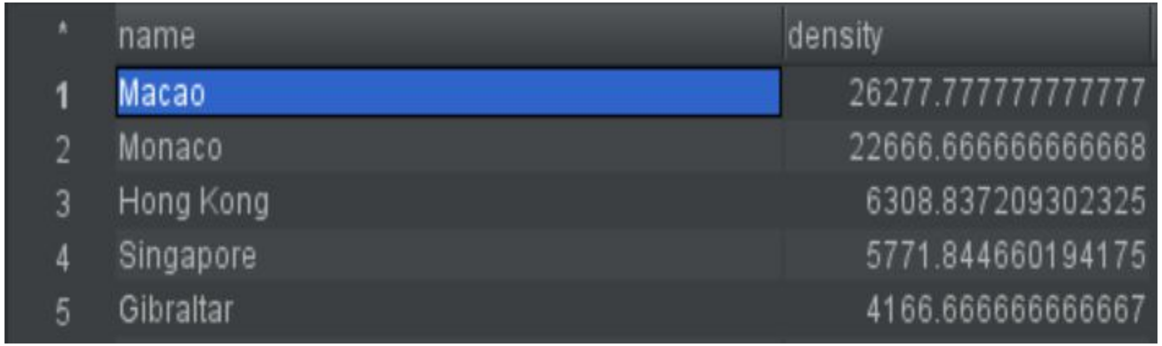
*	name	population
1	Heard Island and McDonald Islands	0
2	United States Minor Outlying Islands	0
3	South Georgia and the South Sandwich Islands	0
4	Antarctica	0
5	Bouvet Island	0

Now, the records are sorted in ascending order, countries with the smallest population appearing first.

Sorting Example Using Derived Fields

You can also sort by any derived fields that were created. To sort by the value of a derived column, you ***must*** refer to the derived column's alias.

```
SELECT name, population/surfacearea AS density
FROM country
ORDER BY density DESC;
```



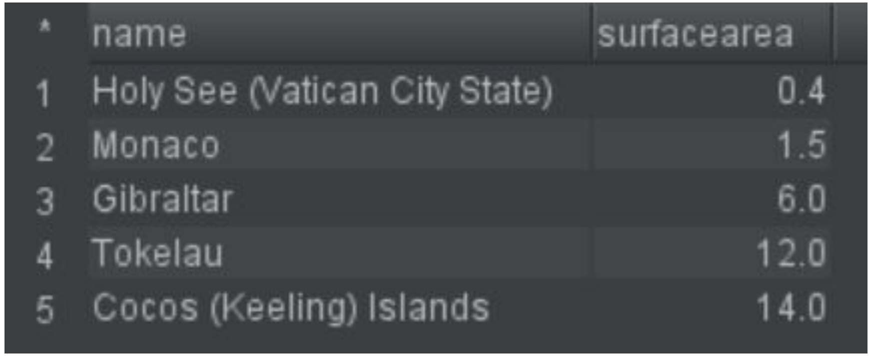
	name	density
1	Macao	26277.777777777777
2	Monaco	22666.666666666668
3	Hong Kong	6308.837209302325
4	Singapore	5771.844660194175
5	Gibraltar	4166.666666666667

Limiting Results

You can limit the number of rows returned from your query using the **LIMIT [n]** clause.

A common use of the LIMIT clause is to combine it with and ORDER BY clause to build limited lists like “top 10”, “3 Best”, or 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

Aggregate Functions

Aggregate Functions

Aggregate data can be created by summarizing the data from one or more columns across multiple rows in a table. Here are some examples using the world database:

- The total population for North America.
- The total GNP for the whole world.
- The average surface area for all countries in Europe.
- The least populated country in Africa.

Aggregate Functions

The five basic aggregate functions:

- **COUNT**: Provides the number of rows in a *subset*.
- **MIN:** The minimum value of a column in a *subset*.
- **MAX**: The maximum value of a column in a *subset*.
- **AVG**: The average value of a column in a *subset*.
- **SUM**: The sum of a column in a *subset*.

Aggregate Functions: COUNT Example

Three examples for COUNT using the country table:

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

Returns the total row count for country.

```
SELECT COUNT(indepyear)
FROM COUNTRY;
```

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

```
SELECT COUNT(*) FROM
COUNTRY
WHERE continent = 'Europe';
```

Returns the row count for all rows having a continent value of Europe.

Aggregate Functions: MIN/MAX example

Two examples for MIN and MAX using the country table:

```
SELECT MAX(surfacearea)
FROM COUNTRY;
```

$\rightarrow$ Returns the maximum surface area encountered in the whole table.

```
SELECT MIN(surfacearea)
FROM COUNTRY;
```

$\rightarrow$ Return the minimum surface area encountered in the whole table.

Aggregate Functions: AVG & SUM example

The following is an example of AVG:

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

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

...and here is an example of SUM:

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

→ This is the total world population.

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 **before** the ORDER BY section.

Aggregate Functions: Group By Example

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

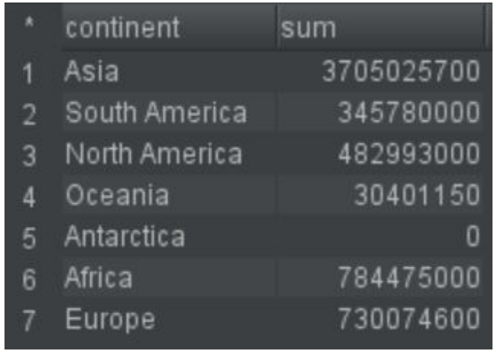
1. Group all the rows into 5 groups, one for each continent.
2. For each group, sum up the population

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

Aggregate Functions: Group By Example

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

```
SELECT continent, SUM(population)
FROM country
GROUP BY continent
```



*	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

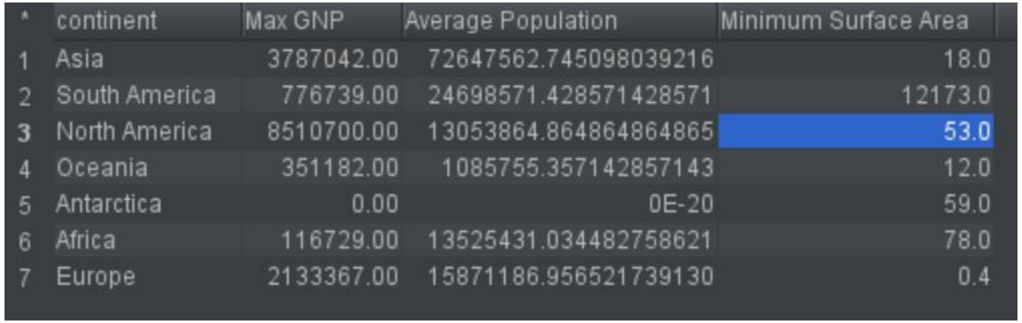
This is the SQL equivalent to step 1, treat all rows with the same continent value as part of the same “bucket” of data or subset.

This is the SQL equivalent to step 2, adding up all the population values **only for a given subset**

Aggregate Functions: A more complex example

You can combine multiple derived derived fields using different aggregate functions. Consider this example, where I want the **maximum GNP**, the **average population size**, and the **minimum surface area** of each continent:

```
SELECT continent,
MAX(gnp) AS "Max GNP",
AVG(population) AS "Average
Population",
MIN(surfacearea) AS "Smallest
Surface Area"
FROM country
GROUP BY continent
```

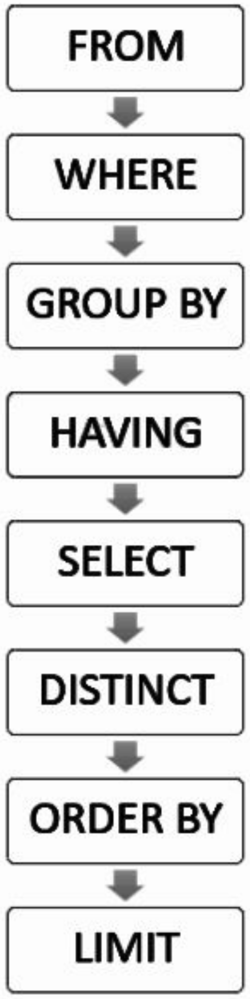


*	continent	Max GNP	Average Population	Minimum Surface Area
1	Asia	3787042.00	72647562.745098039216	18.0
2	South America	776739.00	24698571.428571428571	12173.0
3	North America	8510700.00	13053864.864864864865	53.0
4	Oceania	351182.00	1085755.357142857143	12.0
5	Antarctica	0.00	0E-20	59.0
6	Africa	116729.00	13525431.034482758621	78.0
7	Europe	2133367.00	15871186.956521739130	0.4

PostgreSQL Order of *Evaluation*

The order in which SQL is ***written*** is similar to the way that we might ask a question. However, SQL statements are not ***processed*** in the order in which it is written. SQL Statements are processed in a sequence …

FROM defines the sources, **WHERE** filters the source rows, **GROUP BY** aggregates the data, **HAVING** filters the aggregate rows, **SELECT** defines the returned attributes (columns) **DISTINCT** removes duplicate rows, **ORDER BY** sorts the result rows, and **LIMIT** restricts the number of rows returned


FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
LIMIT