Module 2-2

Intro to Ordering, Grouping, and Database Functions

Objectives

- Ordering
- Limiting Results
- String operation functions
- Aggregate functions
- Grouping Results
- Subqueries

Additional SELECT options

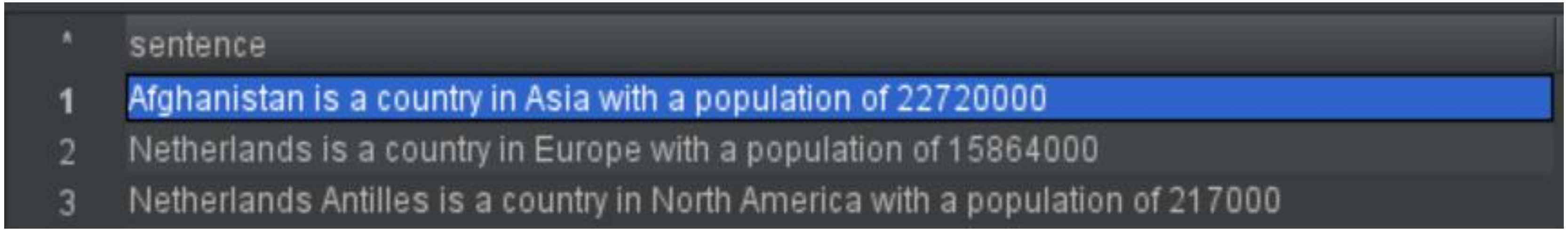
Data Concatenation

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
```

Sorting

- In SQL, sorting is achieved through the ORDER BY statement, with the following format being followed:

  ORDER BY [name of column] [direction]

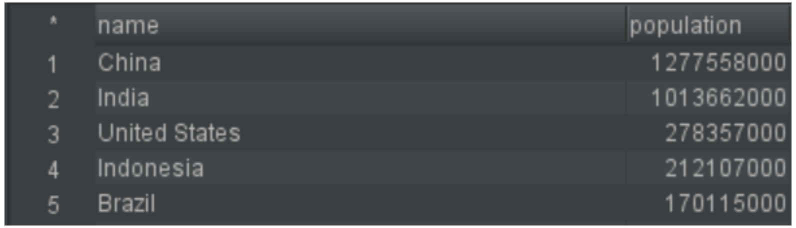
- The ORDER BY section goes after the WHERE statement.
- You need to specify which column you want to sort by.
- You can optionally specify the direction of the sort:
  - **ASC** for ascending (default)
  - **DESC** for descending.

Sorting Example

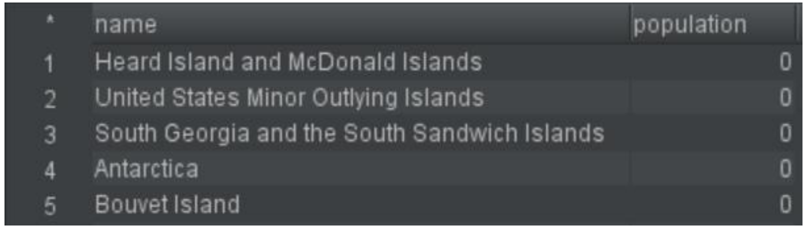
Consider the following example:

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

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



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

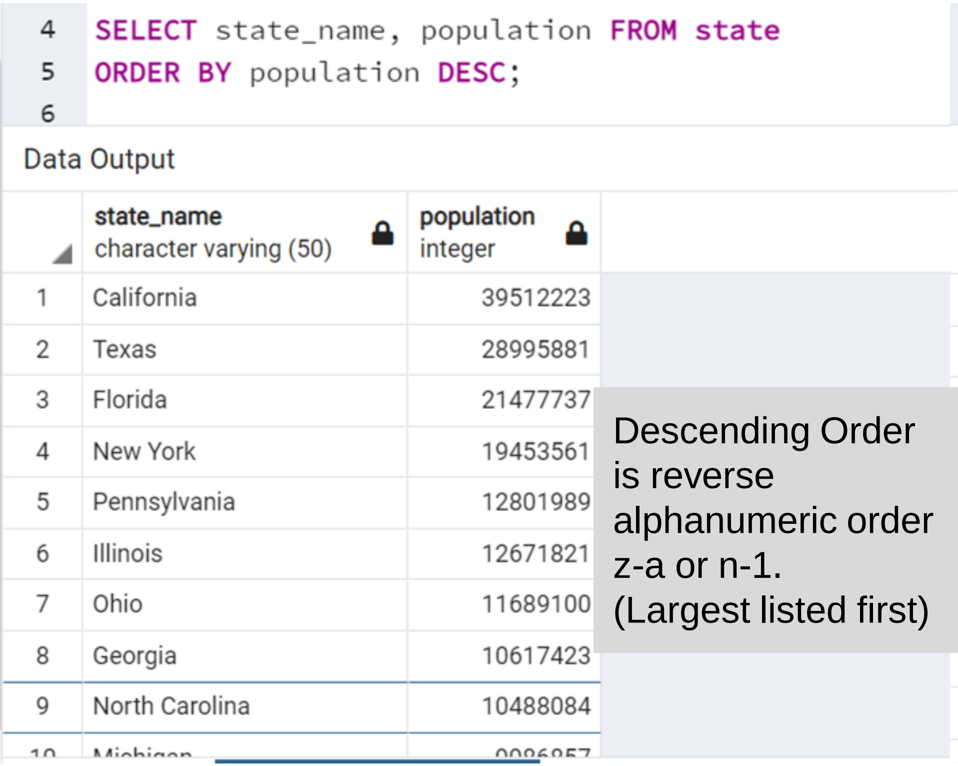


*	name	population
1	Heard Island and McDonald Islands	0
2	United States Minor Outlying Islands	0
3	South Georgia and the South Sandwich Islands	0
4	Antarctica	0
5	Bouvet Island	0

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

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

ORDER BY example :

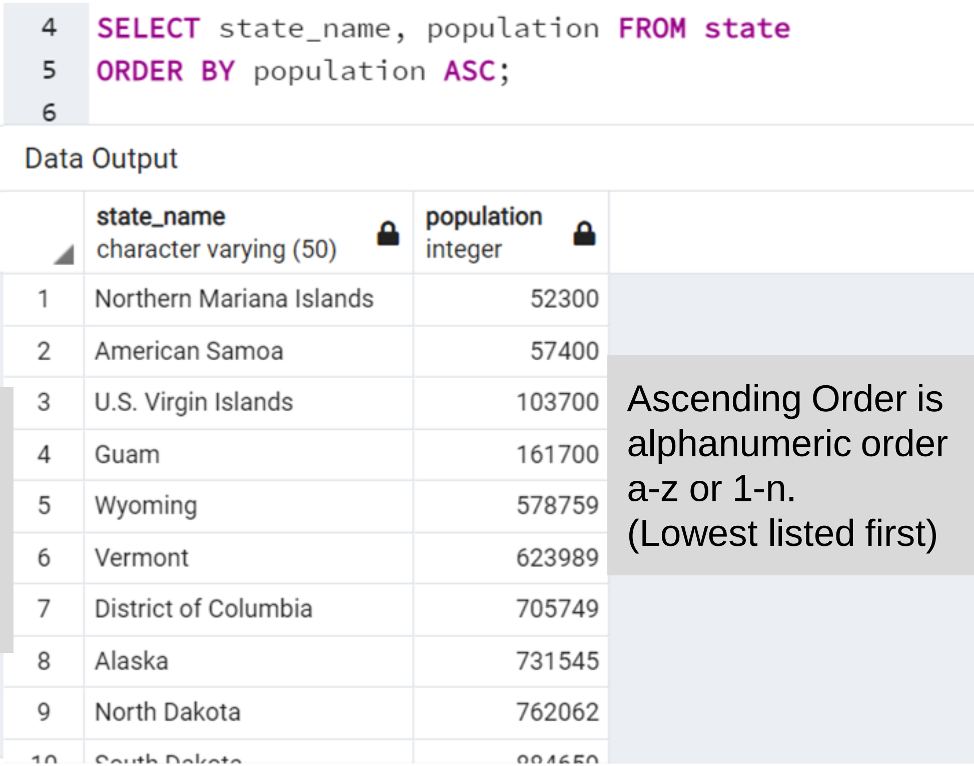


```sql
SELECT state_name, population FROM state
ORDER BY population DESC;
```

Data Output

	state_name character varying (50)	population integer
1	California	39512223
2	Texas	28995881
3	Florida	21477737
4	New York	19453561
5	Pennsylvania	12801989
6	Illinois	12671821
7	Ohio	11689100
8	Georgia	10617423
9	North Carolina	10488084
10	Michigan	9986857

Descending Order is reverse alphanumeric order z-a or n-1. (Largest listed first)

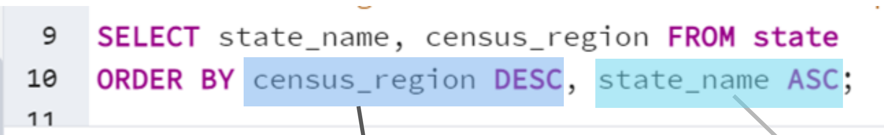


```sql
SELECT state_name, population FROM state
ORDER BY population ASC;
```

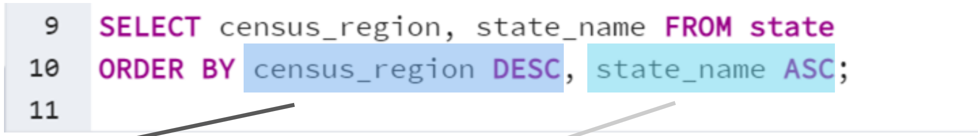
Data Output

	state_name character varying (50)	population integer
1	Northern Mariana Islands	52300
2	American Samoa	57400
3	U.S. Virgin Islands	103700
4	Guam	161700
5	Wyoming	578759
6	Vermont	623989
7	District of Columbia	705749
8	Alaska	731545
9	North Dakota	762062
10	South Dakota	884659

Ascending Order is alphanumeric order a-z or 1-n. (Lowest listed first)



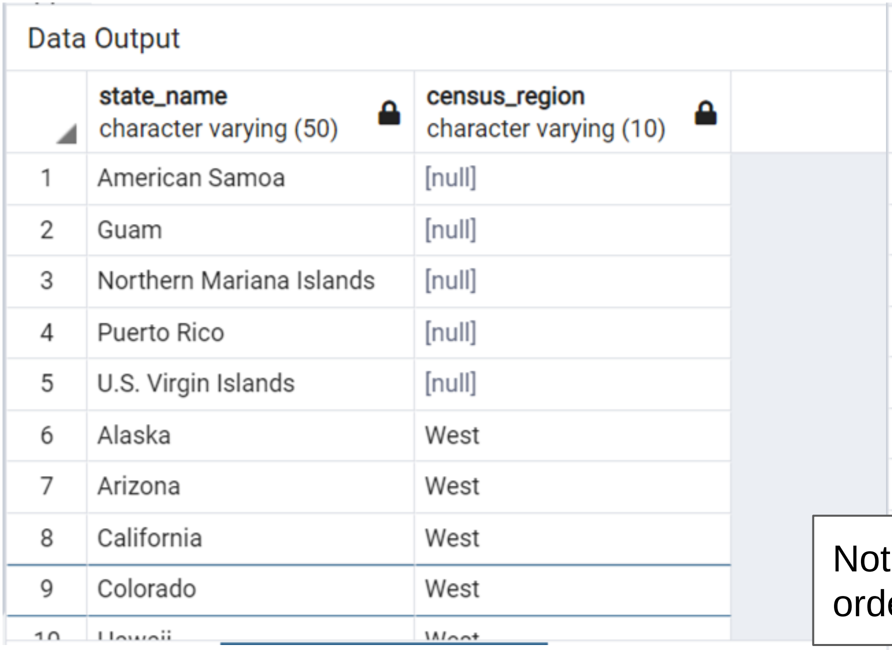
```sql
SELECT state_name, census_region FROM state
ORDER BY census_region DESC, state_name ASC;
```



```sql
SELECT census_region, state_name FROM state
ORDER BY census_region DESC, state_name ASC;
```

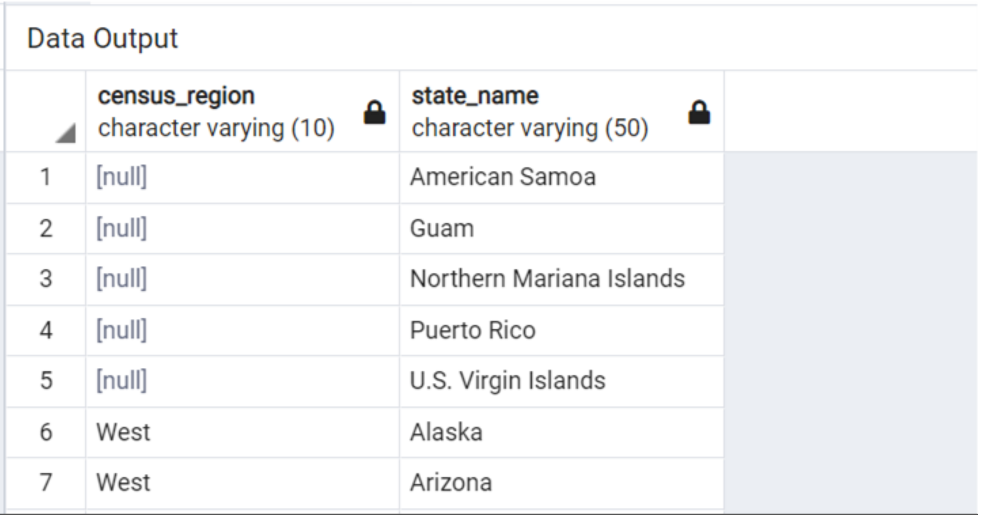
Major sort

Minor sort



Data Output

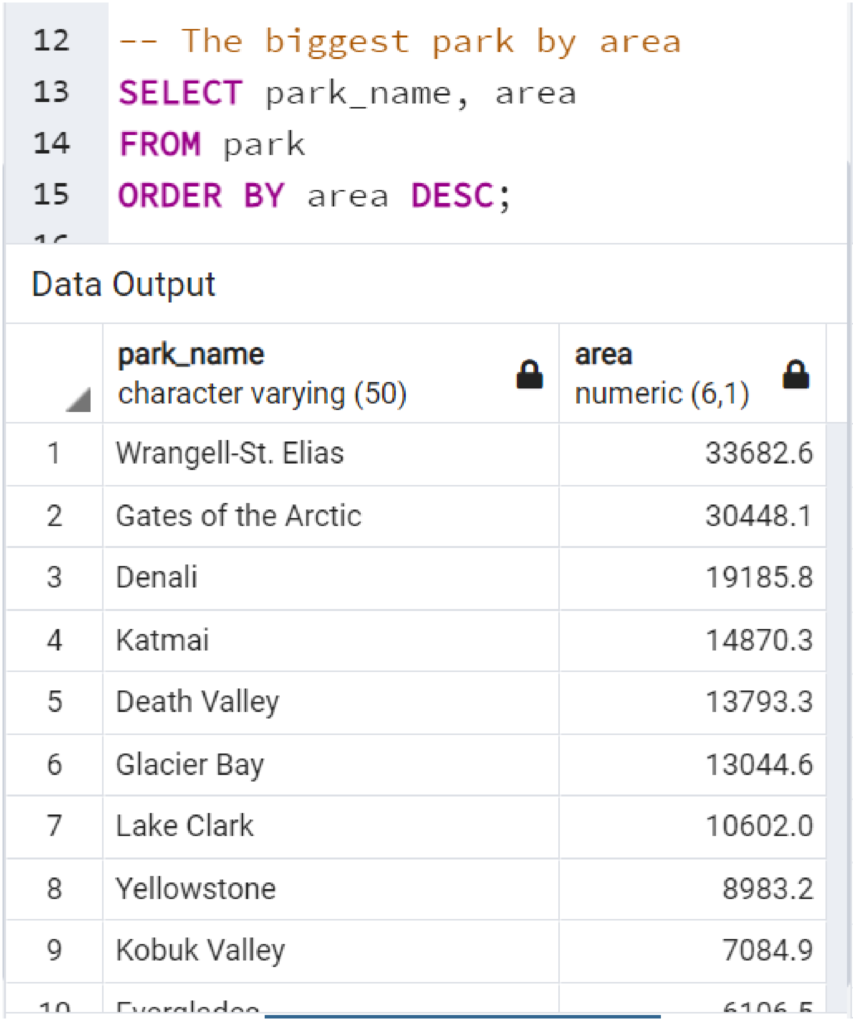
	state_name character varying (50)	census_region character varying (10)
1	American Samoa	[null]
2	Guam	[null]
3	Northern Mariana Islands	[null]
4	Puerto Rico	[null]
5	U.S. Virgin Islands	[null]
6	Alaska	West
7	Arizona	West
8	California	West
9	Colorado	West
10	Hawaii	West



Data Output

	census_region character varying (10)	state_name character varying (50)
1	[null]	American Samoa
2	[null]	Guam
3	[null]	Northern Mariana Islands
4	[null]	Puerto Rico
5	[null]	U.S. Virgin Islands
6	West	Alaska
7	West	Arizona

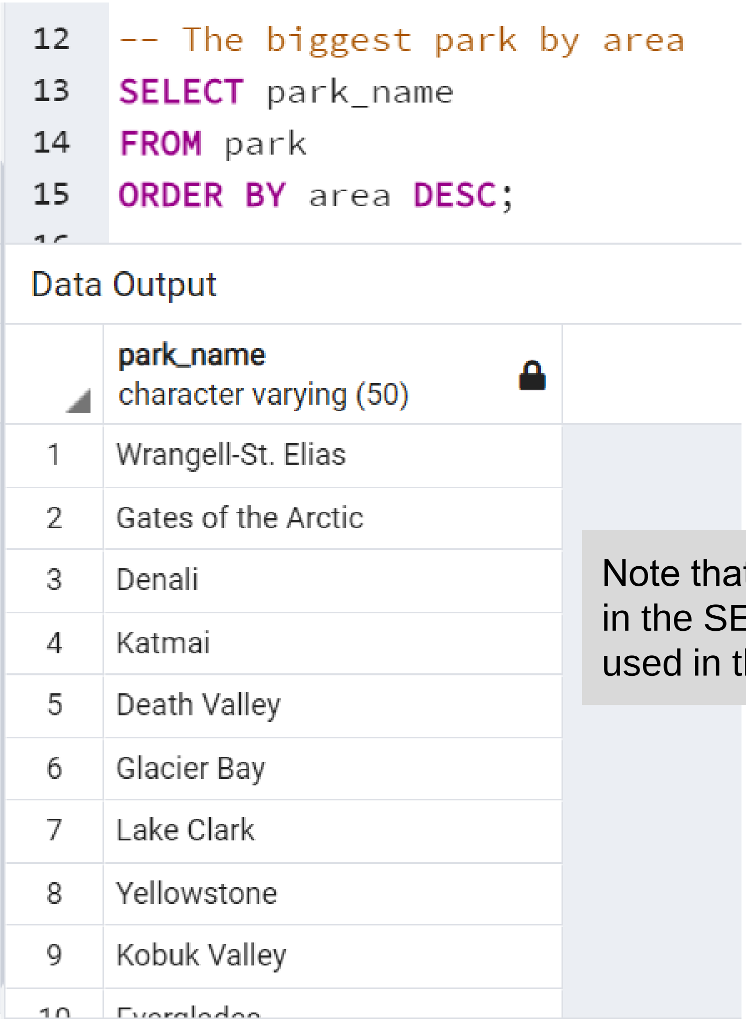
Note the order of columns in the SELECT only controls the order of the columns returned/displayed, not the order.



```sql
-- The biggest park by area
SELECT park_name, area
FROM park
ORDER BY area DESC;
```

Data Output

	park_name character varying (50)	area numeric (6,1)
1	Wrangell-St. Elias	33682.6
2	Gates of the Arctic	30448.1
3	Denali	19185.8
4	Katmai	14870.3
5	Death Valley	13793.3
6	Glacier Bay	13044.6
7	Lake Clark	10602.0
8	Yellowstone	8983.2
9	Kobuk Valley	7084.9
10	Everglades	6106.5



```sql
-- The biggest park by area
SELECT park_name
FROM park
ORDER BY area DESC;
```

Data Output

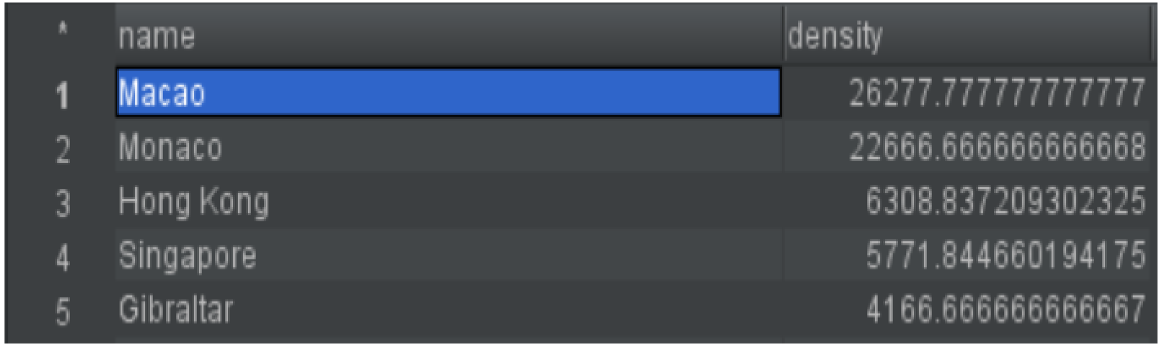
	park_name character varying (50)
1	Wrangell-St. Elias
2	Gates of the Arctic
3	Denali
4	Katmai
5	Death Valley
6	Glacier Bay
7	Lake Clark
8	Yellowstone
9	Kobuk Valley
10	Everglades

Note that the area isn’t in the SELECT, but is used in the ORDER BY

Sorting Example with Derived Fields

You can also sort by any derived fields that were created. Consider the following example:

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



*	name	density
1	Macao	26277.7777777777
2	Monaco	22666.6666666668
3	Hong Kong	6308.837209302325
4	Singapore	5771.844660194175
5	Gibraltar	4166.666666666667

Aggregate Functions

Aggregate data can be created by combining the value of one or more rows in a table. Using the world database, these are a few possible examples:

- 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

We will concern ourselves with the following aggregate functions:

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

Aggregate Functions: Count Example

The following are two examples for COUNT.

`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: MAX/MIN example

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

Returns the maximum surface area encountered in the whole table.

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

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 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.

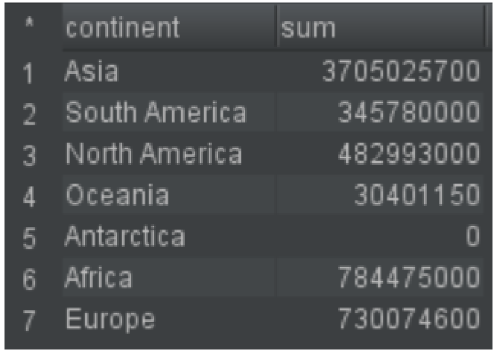
Aggregate Functions: Group By Example

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

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

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

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



*	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

GROUP BY example: **GROUP BY last_name**

Table: Patients

first_name	last_name	age
Jane	Smith	32
Joe	Smith	15
Dave	Jones	25
Sam	Davies	42
Bill	Smith	72
Jill	Jones	54
Fred	Hart	38

SELECT last_name, AVG(age) FROM patients GROUP BY last_name

first_name	last_name	age
Jane	Smith	32
Joe	Smith	15
Dave	Jones	25
Sam	Davies	42
Bill	Smith	72
Jill	Jones	54
Fred	Hart	38

first_name	last_name	age
Jane	Smith	32
Joe	Smith	15
Bill	Smith	72
Dave	Jones	25
Jill	Jones	54
Sam	Davies	42
Fred	Hart	38

First the rows are grouped by unique values in the column in the GROUP BY.

For this table and data it creates 4 groups by last_name: Smith, Jones, Davies, Hart

GROUP BY example: **GROUP BY last_name**

first_name	**last_name**	**age**		**AVG(age)**
Jane	Smith	**32**		
Joe	Smith	**15**	>	**39.6**
Bill	Smith	**72**		
Dave	Jones	**25**	>	**39.5**
Jill	Jones	**54**		
Sam	Davies	**42**	>	**42**
Fred	Hart	**38**	>	**38**

The Aggregate Function, in this case AVG(), is applied to the values in each GROUP.

RETURNED RESULT

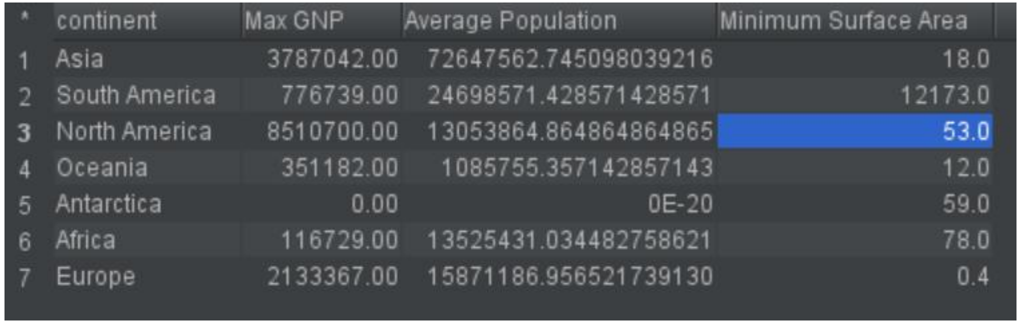
last_name	**AVG(age)**
Smith	39.6
Jones	39.5
Davies	42
Hart	38

The return is 1 row for each group with the aggregate (AVG) performed for the data in each group, in this case the age. Since the items are grouped by last_name, then there will be 1 row returned for each unique last_name in the data set, with the average done for the set of ages associated with the last name.

Aggregate Functions: A more complex example

You can combine multiple 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 'Minimum 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

Limiting Results

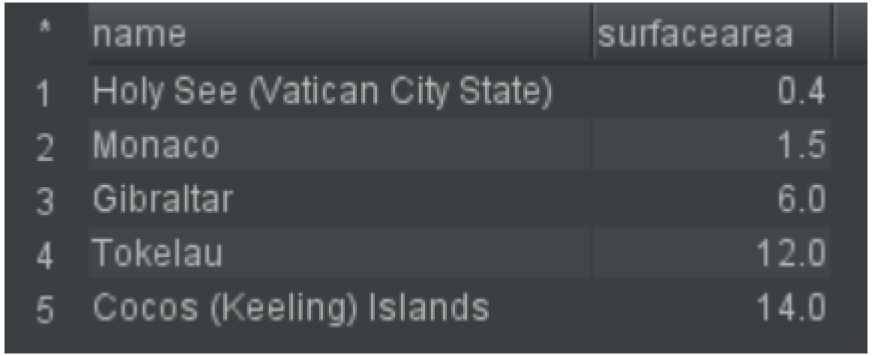
You can limit the number of rows from your query with **LIMIT [n]** . You would specify the number of rows you want to limit the result set by.

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

Limiting Results Example

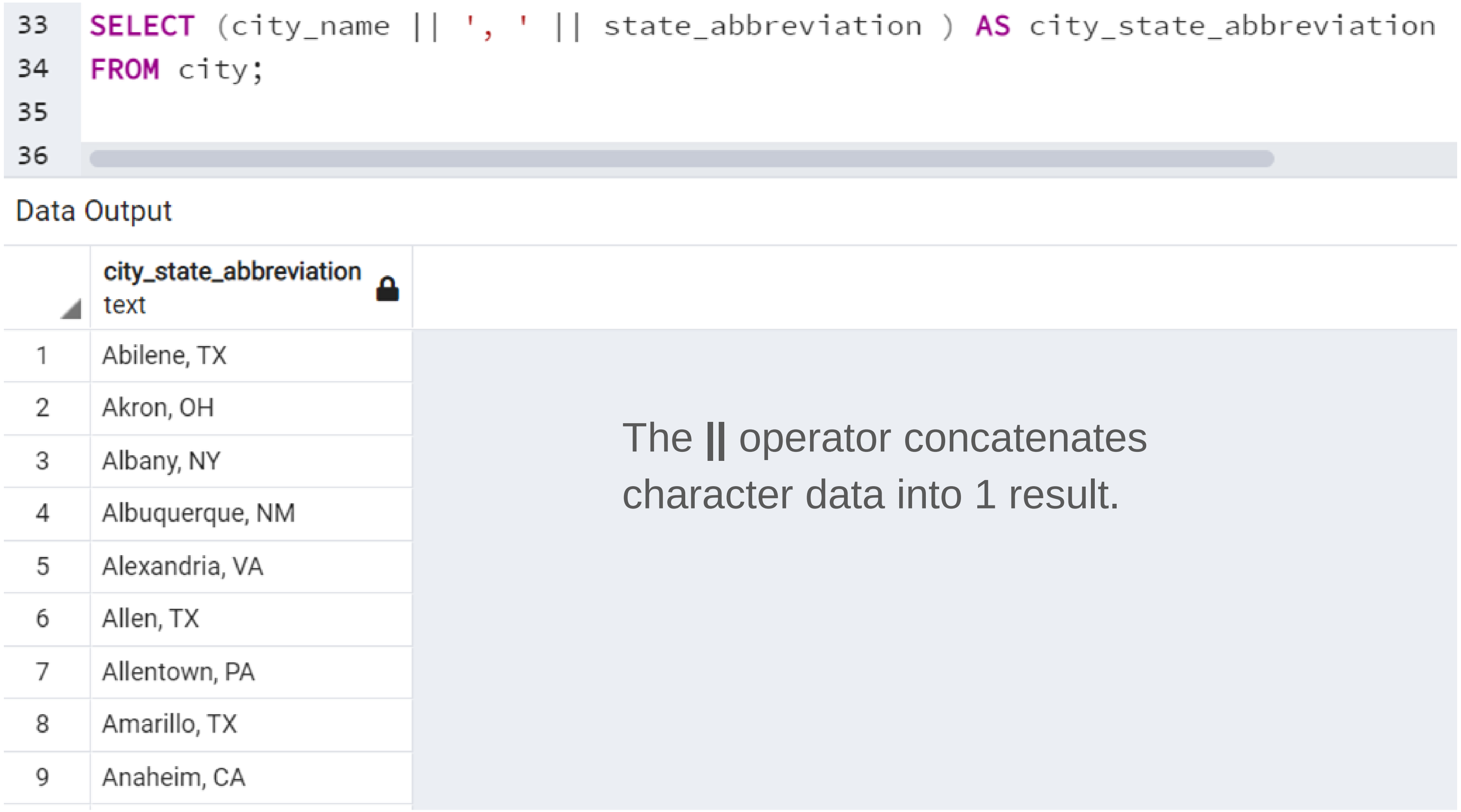
The following query gives you the “top 5” smallest countries by surface area:

```
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

String Operations



```sql
SELECT (city_name || ', ' || state_abbreviation ) AS city_state_abbreviation
FROM city;
```

Data Output

```
   city_state_abbreviation
   text
1  Abilene, TX
2  Akron, OH
3  Albany, NY
4  Albuquerque, NM
5  Alexandria, VA
6  Allen, TX
7  Allentown, PA
8  Amarillo, TX
9  Anaheim, CA
```

The **||** operator concatenates character data into 1 result.

Numeric Operations

round(value, scale) rounds a floating point number to a set scale.

```
select area/3 from park;
```

result : 347.4666666666666

```
select round(area/3, 4) from park;
```

result : 347.4667

```
select round(area/3, 2) from park;
```

result : 347.47

Subqueries

A **SubQuery** is an inner query that can provide results as input to its parent query.
A subquery can only return 1 column of data.

```
SELECT * FROM country WHERE continent = 'Europe' AND gnp > 1000000
```

Returns: `'GBR', 'ITA', 'FRA', 'DEU'`

Without SubQuery: `SELECT * FROM city WHERE countrycode IN ('GBR', 'ITA', 'FRA', 'DEU');`

Subquery provides same list
for use in the in.

With SubQuery: `SELECT * FROM city WHERE countrycode IN (SELECT code FROM COUNTRY WHERE continent = 'Europe' AND gnp > 1000000);`