
YOU MAY VERY WELL BE HERE BY MISTAKE. BUT UNTIL AN I.T. GUY SELLS US HIS SOUL, I WON'T BE ABLE TO ACCESS THE DATABASE.
Copyright ©2006 Creators Syndicate, Inc.

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

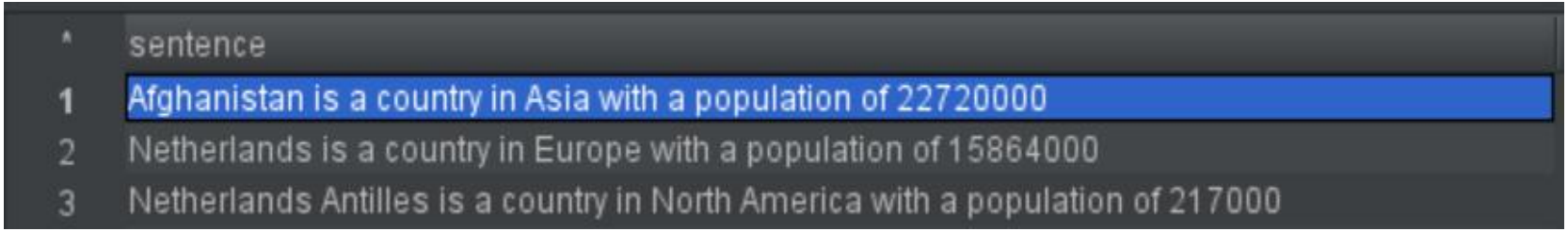
Data Concatenation

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

Absolute Value

The absolute value can be calculated by using the ABS(...) function and including the column for which the absolute value is to be calculated.

- 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.

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
  - **DESC** for descending.

Sorting Example

Consider the following 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

Note that 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

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

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	26277.777777777777
2	Monaco	22666.666666666668
3	Hong Kong	6308.837209302325
4	Singapore	5771.844660194175
5	Gibraltar	4166.666666666667

Aggregate Functions
but first let’s code!

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.

Let’s code some more!

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
```

*	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 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**

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:

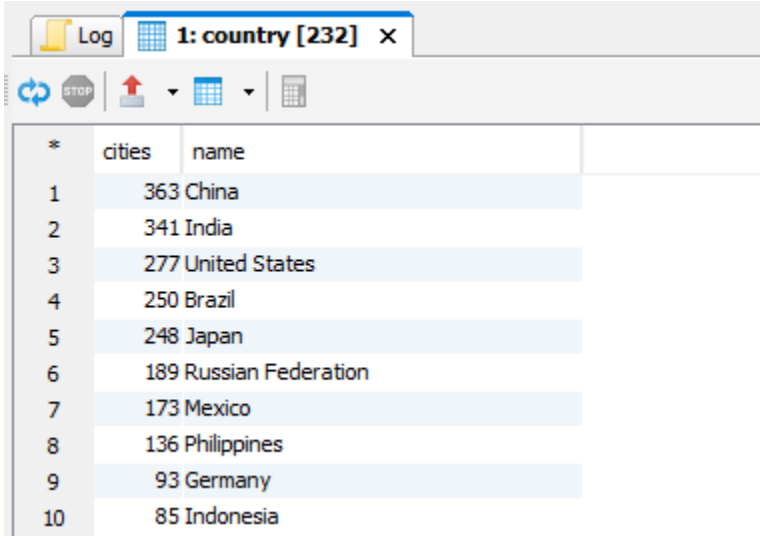
```
SELECT name, surfacearea
FROM country
ORDER BY surfacearea ASC
LIMIT 5;
```

*	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

Subqueries

Counts up all the cities and displays the count using the name of the country rather than the country code:

```
SELECT COUNT(name) AS cities,
   (
     SELECT name
       FROM country
       WHERE code = c.countrycode
   )
   FROM city as c
   GROUP BY c.countrycode
   ORDER BY cities DESC;
```



*	cities	name
1	363	China
2	341	India
3	277	United States
4	250	Brazil
5	248	Japan
6	189	Russian Federation
7	173	Mexico
8	136	Philippines
9	93	Germany
10	85	Indonesia

Objectives

- Ordering

customer
* customer_id
store_id
first_name
last_name
email
address_id
activebool
create_date
last_update
active

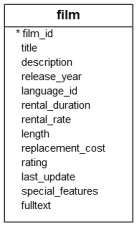
1) Using PostgreSQL `ORDER BY` clause to sort rows by one column

The following query uses the `ORDER BY` clause to sort customers by their first names in ascending order:

```
SELECT
	first_name,
	last_name
FROM
	customer
ORDER BY
	first_name ASC;
```

Objectives

- Ordering
- Limiting Results



1) Using PostgreSQL LIMIT to constrain the number of returned rows example

This example uses the `LIMIT` clause to get the first five films sorted by `film_id`:

```
SELECT
	film_id,
	title,
	release_year
FROM
	film
ORDER BY
	film_id
LIMIT 5;
```

2) Using PostgreSQL LIMIT with OFFSET example

To retrieve 4 films starting from the fourth one ordered by `film_id`, you use both `LIMIT` and clauses as follows:

```
SELECT
	film_id,
	title,
	release_year
FROM
	film
ORDER BY
	film_id
LIMIT 4 OFFSET 3;
```

Objectives

- Ordering
- Limiting Results
- String operation functions

Example	Result
'Post' || 'greSQL'	PostgreSQL
'Value: ' || 42	Value: 42
bit_length('jose')	32
char_length('jose')	4
lower('TOM')	tom
octet_length('jose')	4
overlay('Txxxxas' placing 'hom' from 2 for 4)	Thomas
position('om' in 'Thomas')	3
substring('Thomas' from 2 for 3)	hom
substring('Thomas' from '...$')	mas
substring('Thomas' from '%#"o_a#"_' for '#')	oma
trim(both 'x' from 'xTomxx')	Tom
upper('tom')	TOM

Objectives

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

Introduction to PostgreSQL aggregate functions

Aggregate functions perform a calculation on a set of rows and return a single row. PostgreSQL provides all standard SQL's aggregate functions as follows:

- `AVG()` – return the average value.
- `COUNT()` – return the number of values.
- `MAX()` – return the maximum value.
- `MIN()` – return the minimum value.
- `SUM()` – return the sum of all or distinct values.

Objectives

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

For example, to select the total amount that each customer has been paid, you use the `GROUP BY` clause to divide the rows in the `payment` table into groups grouped by customer id. For each group, you calculate the total amounts using the `SUM()` function.

The following query uses the `GROUP BY` clause to get total amount that each customer has been paid:

```
SELECT
	customer_id,
	SUM (amount)
FROM
	payment
GROUP BY
	customer_id;
```

payment
* payment_id
customer_id
staff_id
rental_id
amount
payment_date

1) Using PostgreSQL `GROUP BY` without an aggregate function example

You can use the `GROUP BY` clause without applying an aggregate function. The following query gets data from the `payment` table and groups the result by customer id.

```
SELECT
   customer_id
FROM
   payment
GROUP BY
   customer_id;
```

	customer_id smallint
1	184
2	87
3	477
4	273
5	550
6	51
7	394
8	272
9	70

Objectives

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

https://www.postgresqltutorial.com/postgresql-subquery/

Summary: in this tutorial, you will learn how to use the **PostgreSQL subquery** that allows you to construct complex queries.

Introduction to PostgreSQL subquery

Let's start with a simple example.

Suppose we want to find the films whose rental rate is higher than the average rental rate. We can do it in two steps:

- Find the average rental rate by using the SELECT statement and average function (`AVG`).
- Use the result of the first query in the second `SELECT` statement to find the films that we want.

The following query gets the average rental rate:

```
SELECT
        AVG (rental_rate)
FROM
        film;
```