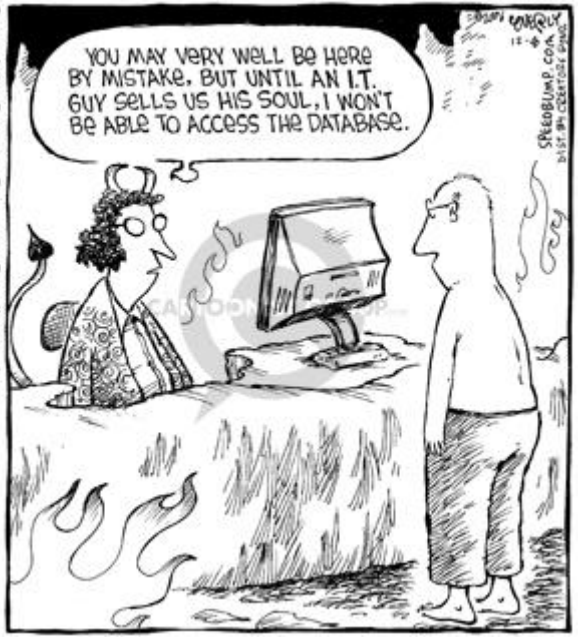

Module 2-2

Intro to Ordering, Grouping, and Database Functions

Objectives

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

String Operations

```
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.

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.

```
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)

```
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)

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

```
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:

Query Editor | Query History

```
SELECT state_name, population/area AS density
    FROM state
    ORDER BY density DESC;
```

Data Output | Explain | Messages | Notifications

	state_name character varying (50)	density integer
1	District of Columbia	4009
2	New Jersey	393
3	Rhode Island	264
4	Massachusetts	252
5	Connecticut	248
6	Puerto Rico	231
7	Maryland	188
8	Delaware	151

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

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 UnitedStates database, these are a few possible examples:

- The total population for a particular census region.
- The largest state.
- The average sales tax for the US.
- The least populated state.

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 state;
```

Returns the total row count for state.

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

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

```
SELECT COUNT(*)
   FROM state
   WHERE census_region = 'West';
```

Returns the row count for all rows having a census region of West.

Aggregate Functions: MAX/MIN example

```
SELECT MAX(population) as Largest_Population
   FROM state;
```

Returns the maximum population encountered in the whole table.

```
SELECT MIN(sales_tax)
   FROM state;
```

Return the minimum sales_tax encountered in the whole table.

Aggregate Functions: AVG example

The following is an example of AVG:

```
SELECT AVG(sales_tax)
   FROM state;
```

Returns the average sales tax of all the states in the state table.

Aggregate Functions: SUM example

The following is an example of SUM:

```
SELECT SUM(area)
   FROM state;
```

This is the total US area.

Let’s code some more!

GROUP BY

GROUP BY groups records into summary rows and returns one record for each group.

Used in conjunction with Aggregate Functions to tell SQL how to group non-aggregate values. All non-aggregate columns in the SELECT must be in the GROUP BY clause.

```
SELECT min(population), max(population), region, name FROM country
GROUP BY region, name
ORDER BY region, name
```

Groups are applied in the order listed. So first the data is grouped by region and then by name within each region, and then the min() and max() aggregate function is applied to each group.

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

first_name	last_name	age		AVG(age)
Jane	Smith	32	>	39.6
Joe	Smith	15		
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.

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

The **LIMIT #** clause can be used to limit the number of rows returned. The LIMIT clause is added at the end of the query.

Note: Limiting the number of rows returned has nothing to do with ordering (or sorting the data).

```
SELECT city_name, population FROM city
ORDER BY population DESC
LIMIT 10;
```

Data Output

	city_name character varying (50)	population integer
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
9	Dallas	1343573
10	San Jose	1021795

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 state rather than the state abbreviation:

```
SELECT COUNT (city_name) AS cities,
    (SELECT state_name
        FROM state
        WHERE state_abbreviation = c.state_abbreviation
    )
    FROM city c
    GROUP BY c.state_abbreviation
    ORDER BY cities DESC;
```

	cities bigint	state_name character varying (50)
1	75	California
2	41	Texas
3	22	Florida
4	12	Colorado
5	10	Washington
6	10	Arizona
7	9	North Carolina
8	8	Illinois
9	7	Michigan
10	7	Virginia
11	7	New Jersey
12	7	Georgia
13	6	Oregon
14	6	Tennessee
15	6	Ohio

Rounding

```
SELECT AVG(sales_tax)
   FROM state;
```

	avg numeric
1	4.9851071428571429

```
SELECT round(AVG(sales_tax), 2)
   FROM state;
```

	round numeric
1	4.99

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

film
* film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
rating
last_update
special_features
fulltext

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;
```