UNIT1: INTRODUCTION TO DATA AND DATABASES

Queries & Subqueries

LAGUARDIA COMMUNITY COLLEGE

Week 3

Upcoming Assignments (Required):

- Assignment 3A - Due Friday, April 5th, 11:59PM (Topic for Final Project)
- Assignment 3B - Due Sunday, April 7th, 11:59PM (UNION, EXTRAT, DATE, MATH)
- **Challenge Question:** Using AI with Databases I

NO CLASSES MONDAY AND TUESDAY - Review Packet Instead

Note: If you would like to complete any of our challenge questions throughout the semester for practice, please let me know on Slack and I will open them for you.

Today's Fun Fact:

Many of the world's most popular online video games, including massively multiplayer online games (MMOs), rely on SQL databases to manage player data, game states, inventory items, and more. Complex SQL databases can keep track of the actions and possessions of millions of players at one time, real-time!

SQL FUNDAMENTALS-

EXTRACT, DATE, MATH

LAGUARDIA COMMUNITY COLLEGE

TIMESTAMPS

The TIMESTAMP() function returns a datetime value based on a date or datetime value.

In PostgreSQL, the extract function extracts parts from a date

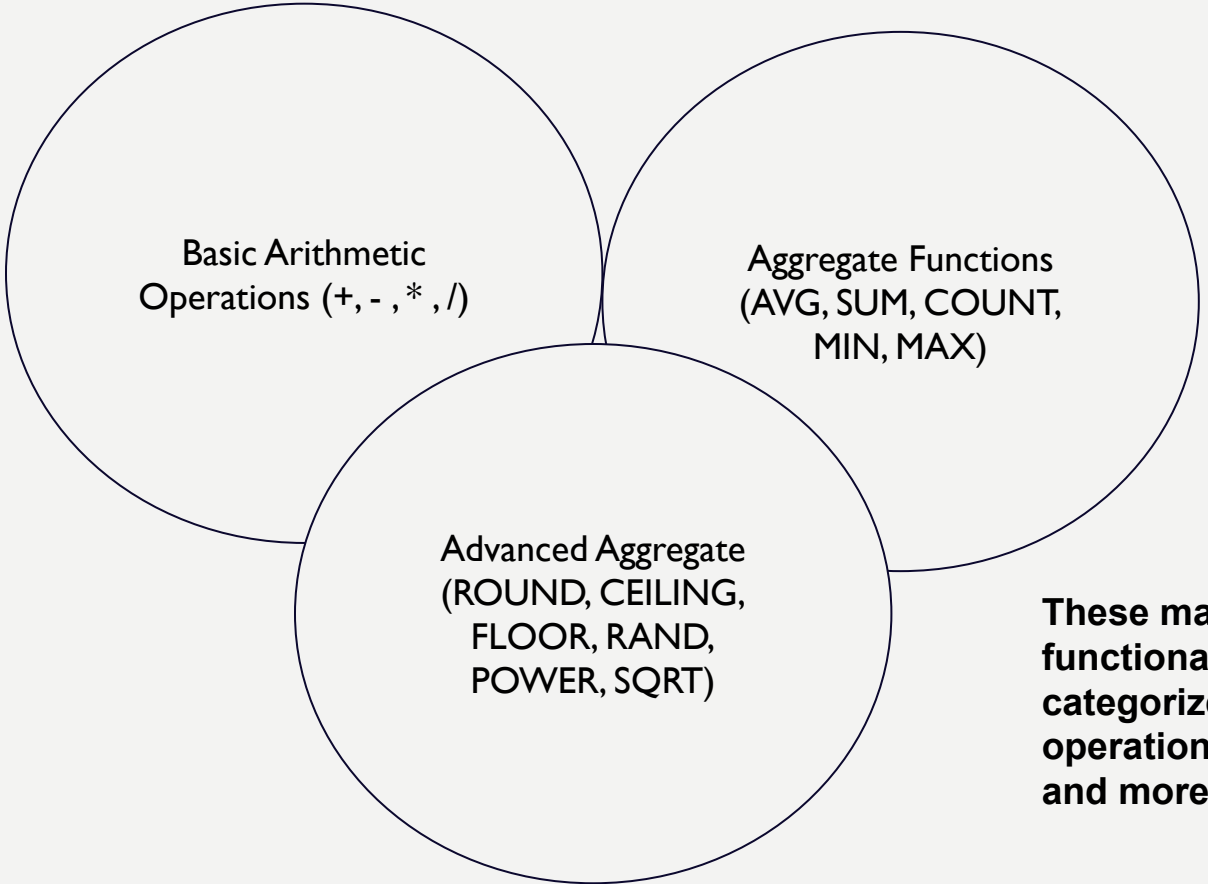
- Extract(unit from date)

EXTRACT FUNCTION

Extract Function

Unit	Explanation
day	Day of the month (1 to 31)
dow	Day of the week (0=Sunday, 1=Monday, 2=Tuesday, ... 6=Saturday)
doy	Day of the year (1=first day of year, 365/366=last day of the year, depending if it is a leap year)
epoch	Number of seconds since '1970-01-01 00:00:00 UTC', if date value. Number of seconds in an interval, if interval value
hour	Hour (0 to 23)
microseconds	Seconds (and fractional seconds) multiplied by 1,000,000
millennium	Millennium value
milliseconds	Seconds (and fractional seconds) multiplied by 1,000
minute	Minute (0 to 59)
month	Number for the month (1 to 12), if date value. Number of months (0 to 11), if interval value
quarter	Quarter (1 to 4)
second	Seconds (and fractional seconds)
week	Number of the week of the year based on ISO 8601 (where the year begins on the Monday of the week that contains January 4th)
year	Year as 4-digits

WHY IS MATHEMATICAL FUNCTIONS USEFUL? (MATH)



These mathematical functionalities can be broadly categorized into basic arithmetic operations, aggregate functions, and more advanced

WHY IS DATE EXTRACTION USEFUL? (EXTRACT)

```
SELECT EXTRACT(MONTH FROM rental_date) AS
rental_month, COUNT(*) AS total_rentals
FROM rental
WHERE EXTRACT(YEAR FROM rental_date) = 2005
GROUP BY rental_month
ORDER BY rental_month;
```

The `EXTRACT` clause in SQL allows us to retrieve specific parts of a date or time value, such as the year, month, day, etc. This can be particularly useful for reporting and analysis tasks, where you might need to group or filter data based on a date component.

Example (EXTRACT)

```
1  select * from payment;
```

Data Output | Explain | Messages | Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp without time zone
1	17503	341	2	1520	7.99	2007-02-15 22:25:46.996577
2	17504	341	1	1778	1.99	2007-02-16 17:23:14.996577
3	17505	341	1	1849	7.99	2007-02-16 22:41:45.996577

Payment Date Extraction

```
SELECT
  EXTRACT(YEAR FROM payment_date) AS payment_year,
  EXTRACT(MONTH FROM payment_date) AS payment_month,
  EXTRACT(DAY FROM payment_date) AS payment_day
FROM payment;
```

WHY IS MATHEMATICAL FUNCTIONS USEFUL? (MATH)

```
SELECT
  AVG(amount) AS average_payment,
  SUM(amount) AS total_revenue
FROM payment
WHERE EXTRACT(YEAR FROM payment_date) = 2007;
```

SQL supports a variety of mathematical functions and operators that allow you to perform arithmetic and complex mathematical calculations directly within your queries.

MATHEMATICAL FUNCTIONS

- Some examples of math functions:

Operator	Description	Example	Result
+	addition	2 + 3	5
-	subtraction	2 - 3	-1
*	multiplication	2 * 3	6
/	division (integer division truncates the result)	4 / 2	2
%	modulo (remainder)	5 % 4	1
^	exponentiation (associates left to right)	2.0 ^ 3.0	8
|/	square root	|/ 25.0	5
||/	cube root	||/ 27.0	3
!	factorial	5 !	120

WHY IS MATHEMATICAL FUNCTIONS USEFUL? (MATH)

```
SELECT
  AVG(amount) AS average_payment,
  SUM(amount) AS total_revenue
FROM payment
WHERE EXTRACT(YEAR FROM payment_date) = 2007;

SELECT
  ROUND(AVG(amount), 2) AS average_payment,
  ROUND(SUM(amount), 2) AS total_revenue
FROM payment
WHERE EXTRACT(YEAR FROM payment_date) = 2007;
```

SQL supports a variety of mathematical functions and operators that allow you to perform arithmetic and complex mathematical calculations directly within your queries.

Did someone say BREAK?!

Shake out, and stay hydrated!

Return at: 7:15

JOINS - Creating a Junction Table

In a simple world, if each movie had only one actor, and each actor acted in only one movie, we could directly link these tables with a foreign key. However, movies usually feature multiple actors, and actors typically act in multiple movies, creating a **many-to-many relationship**.
Junction Tables can be premade (ex. film_actor) or newly created.

Scenario: In this scenario, we want to allow customers to mark certain films as their favorites, creating a personalized list of films for each customer. This requires establishing a many-to-many relationship between the `customer` and `film` tables.

JOINS - Creating a Junction Table

Step 1 - Explore the current tables we need.

First, let's review the key components of the existing tables we'll be linking:

customer` Table: Contains customer information, with a unique identifier for each customer (`customer_id`).

film` Table: Contains details about films, including a unique identifier for each film (`film_id`).

Step 2 - Design the Junction Table

To link these tables in a many-to-many relationship, we'll create a junction table named `customer_favorite_films`. This table will have at least two columns: `customer_id` and `film_id`. We may also include an additional column such as `added_on` to track when a film was marked as a favorite.

(X,Y) X = Independent V ; Y = Dependent Variable

JOINS - Creating a Junction Table

Step 3 - Create the Junction Table

```
CREATE TABLE customer_favorite_films (
    customer_id INT,
    film_id INT,
    added_on DATE DEFAULT CURRENT_DATE,
    PRIMARY KEY (customer_id, film_id),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON DELETE CASCADE,
    FOREIGN KEY (film_id) REFERENCES film(film_id) ON DELETE CASCADE
);
```

Step 4 - Populate the Junction Table with Data!
To add favorite films for a customer, you would insert records into the `customer_favorite_films` table.

```
INSERT INTO customer_favorite_films (customer_id, film_id)
VALUES (1, 101), (1, 102);
```

Let's take a look - SELECT * FROM customer_favorite_films;

JOINS - Creating a Junction Table

Step 5 - Adding More Data

```
INSERT INTO customer_favorite_films (customer_id,
film_id) VALUES
(2, 103), (2, 102), (3, 104), (1, 105),
(4, 101), (5, 102), (6, 107),
(7, 108), (8, 109), (9, 110),
(10, 111), (11, 112), (12, 113),
(13, 114), (14, 115), (15, 116),
(16, 117), (17, 118), (18, 119),
(19, 120), (20, 121), (21, 122),
(22, 123), (23, 124), (24, 125),
(25, 126);
```

Step 6 - Let's use our data!

6a -

```
SELECT customer_id, film_id FROM
customer_favorite_films;
```

6b -

```
SELECT c.first_name, c.last_name,
f.title
FROM customer_favorite_films cff
JOIN customer c ON cff.customer_id =
c.customer_id
JOIN film f ON cff.film_id = f.film_id;
```

6c -

```
SELECT f.title
FROM customer_favorite_films cff
JOIN film f ON cff.film_id = f.film_id
WHERE cff.customer_id = 1;
```

6d -

```
SELECT f.title
FROM customer_favorite_films cff
JOIN film f ON cff.film_id = f.film_id
ORDER BY f.title;
```

Activity - Topic Selection for Final Project (10 Minutes)

1. Open *Assignment 3A* and SKIP the instructions in Part 1. Complete this on your own later in refining your assignment.
2. Discuss your two topics with classmates, and listen to others' topic descriptions as well.
3. **Complete Part 2a and 2B of assignment 3A.**

You will complete the other parts of this assignment at a later time on your own. Deadline is this Friday, 11:59PM

END OF CLASS