Lesson 3A: UNIONS, EXTRACT, DATE, MATH

Welcome to Lesson 3A! In this journey, we'll dive deep into the more advanced SQL functionalities that play a crucial role in making the most out of relational databases. This lesson is designed to give you a robust understanding of JOINS, UNIONS, and essential math functions, empowering you to perform sophisticated data manipulation and analysis.

Objectives:

1. Master JOINS: Learn how to seamlessly combine data from two or more tables based on related columns.
2. Harness the Power of UNIONS: Discover how to merge results from multiple SELECT queries into one comprehensive dataset.
3. Explore Date and Time with EXTRACT: Get hands-on with extracting specific components from dates and times for detailed temporal analysis.
4. Perform Calculations with SQL: Dive into basic and advanced mathematical operations to compute directly within your queries.

Lesson 3A:

1. **Understanding JOINS**
   Let's better understand JOIN clauses with the following example:

INNER JOIN

```
SELECT film.title, language.name
FROM film
INNER JOIN language ON film.language_id = language.language_id;
```

In this example, the query retrieves the title of each film along with the name of the language the film is in, linking the `film` and `language` tables on their common `language_id` field.

LEFT JOIN

```
SELECT actor.first_name, actor.last_name, film.title
FROM actor
LEFT JOIN film_actor ON actor.actor_id = film_actor.actor_id
LEFT JOIN film ON film_actor.film_id = film.film_id;
```

In this example, the query query lists all actors whether they have been in a film or not. For actors who have appeared in films, it also shows the film titles.

RIGHT JOIN

Note: `RIGHT JOIN` is conceptually opposite to `LEFT JOIN`, but because `RIGHT JOIN` is less commonly used and some databases (like MySQL) fully support it while others might not, here's an example based on `LEFT JOIN` **logic reversed**. Remember, the `RIGHT JOIN` can always be restructured as a `LEFT JOIN` for compatibility.

```
SELECT film.title, actor.first_name, actor.last_name
```

```
FROM film
LEFT JOIN film_actor ON film.film_id = film_actor.film_id
LEFT JOIN actor ON film_actor.actor_id = actor.actor_id;
```

OR

```
SELECT film.title, actor.first_name, actor.last_name
FROM actor
RIGHT JOIN film_actor ON actor.actor_id = film_actor.actor_id
RIGHT JOIN film ON film_actor.film_id = film.film_id;
```

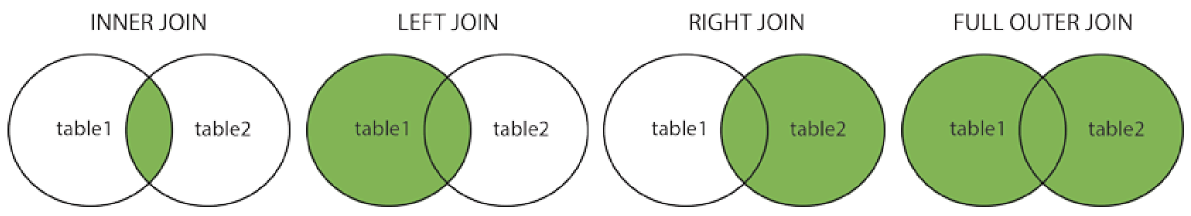
To restructure the example to follow a `RIGHT JOIN` logic while maintaining the same tables and relationships, we'd approach it from the perspective of starting with the `actor` table and then joining to `film_actor` and `film`. However, remember that in practice, a `RIGHT JOIN` is just a `LEFT JOIN` written in reverse order, and it's more common to use `LEFT JOIN` for clarity and compatibility. Both of these queries aim to achieve the same outcome, which is to list all films along with the first and last names of the actors who appeared in those films. They approach the task from different directions, but achieve more or less the same output.

FULL JOIN

```
SELECT actor.first_name, actor.last_name, category.name
FROM actor
FULL JOIN film_actor ON actor.actor_id = film_actor.actor_id
FULL JOIN film ON film_actor.film_id = film.film_id
FULL JOIN film_category ON film.film_id = film_category.film_id
FULL JOIN category ON film_category.category_id = category.category_id;
```

In this example, the query attempts a comprehensive listing, including all actors and all categories, linking them through films where possible. In scenarios where `FULL JOIN` is not available or not supported, achieving this result might require combining the results of multiple `LEFT JOIN` queries or using a UNION.

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



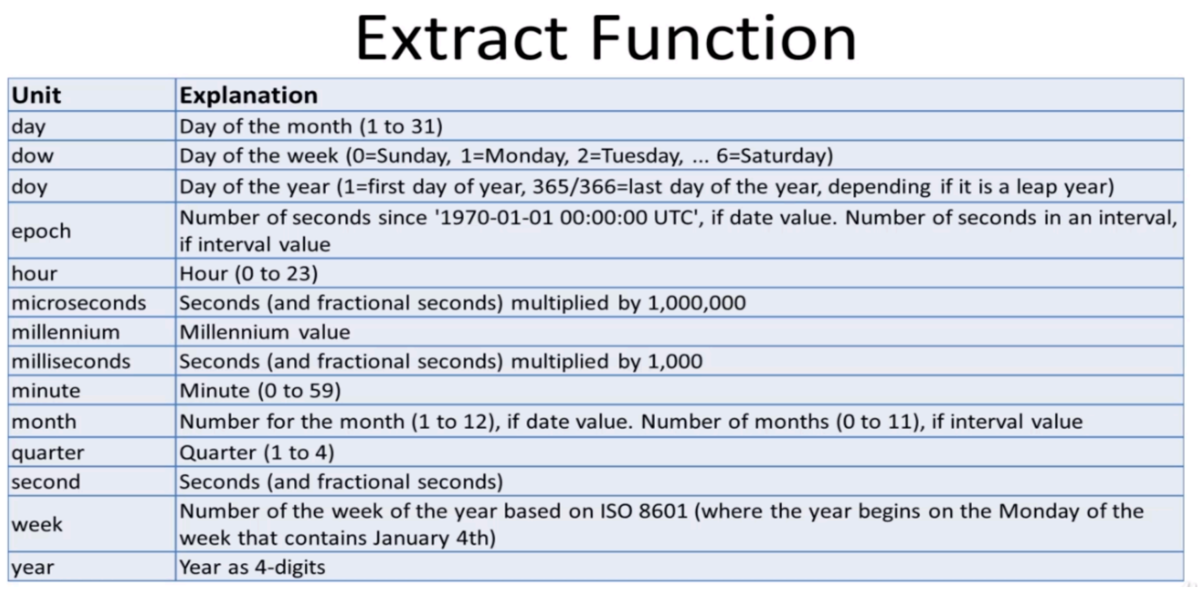
2. Understanding UNIONS

Distinguish between UNION and UNION ALL, and understand how to amalgamate results from separate queries.

```
SELECT first_name AS name FROM actor
UNION
SELECT name FROM category;
```

In this example, the query takes the first names of actors from the ‘actor’ table, and category names from the ‘category’ table, combining them into a single list with no duplicates.

3. **Understanding MATH Functions**
   Perform calculations directly in SQL queries using basic arithmetic operations (+, -, *, /) and explore advanced functions (AVG, SUM, COUNT, MIN, MAX).



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

```
SELECT EXTRACT(DOW FROM rental_date) AS day_of_week, COUNT(*) AS rentals
FROM rental
GROUP BY day_of_week
ORDER BY rentals DESC;
```

In this example, the query is extracting the number of days of the week from rental date, and counting the amount of rentals for each of those days.

```
SELECT EXTRACT(YEAR FROM rental_date) AS rental_year, COUNT(*) AS
total_rentals
FROM rental
GROUP BY rental_year
ORDER BY total_rentals DESC
LIMIT 1;
```

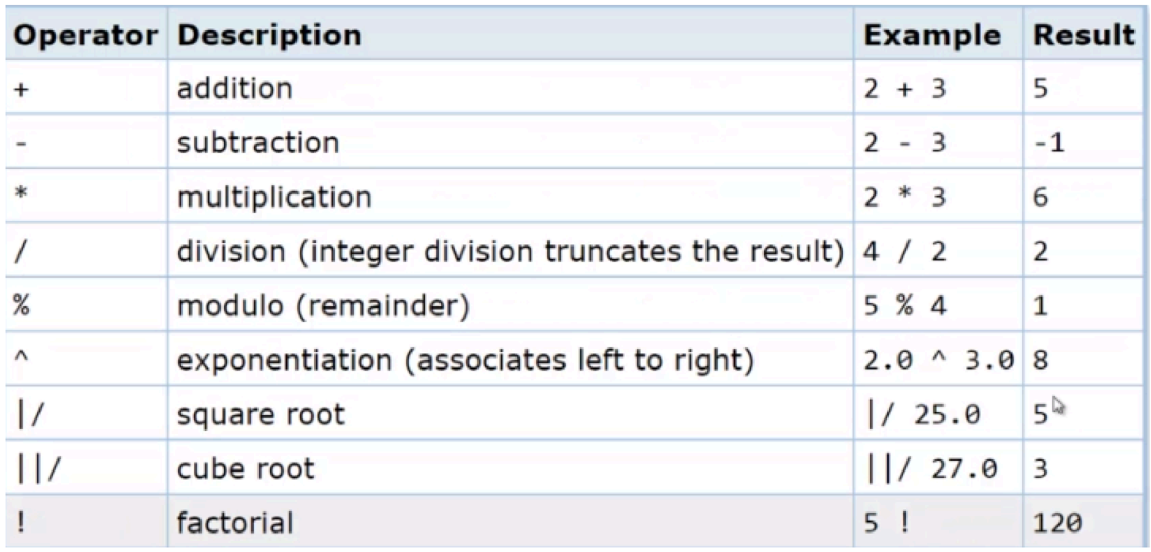
In this example, the query is finding the year with the most rentals.

```
SELECT COUNT(*) AS january_rentals
FROM rental
WHERE EXTRACT(MONTH FROM rental_date) = 1;
```

In this example, the query is counting the amount of rentals made in January across all years.

```
SELECT film_id
FROM film
ORDER BY RANDOM()
LIMIT 1;
```

In this example, the query uses the random function to select a random film ID that represents a specific instance of a film.



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

```
SELECT SQRT(AVG(length)) AS sqrt_avg_length
FROM film;
```

In this example, the query is calculating the square root of the average length of each film.

```
SELECT MAX(rental_rate) * AVG(rental_duration) AS result
FROM film;
```

In this example, the query is calculating the highest rental rate, multiplied by the average rental duration.

```
SELECT ROUND(AVG(replacement_cost), 2) AS rounded_avg_cost
FROM film;
```

In this example, the query is rounding the average replacement cost of films to 2 decimal places.

```
SELECT MAX(rental_date) - MIN(rental_date) AS date_difference
FROM rental;
```

In this example, the query identifies the difference in days between the earliest and latest rental.

```
SELECT SUM(film.length) AS total_length
FROM film
JOIN film_category ON film.film_id = film_category.film_id
JOIN category ON film_category.category_id = category.category_id
WHERE category.name = 'Comedy';
```

In this example, the query is finding the total length of all movies in a specific category (in this case, comedy).

```
SELECT AVG(rental_duration) AS average_rental_duration
FROM film;
```

In this example, the query is calculating the average rental duration for all films.

Key Terms:

- **JOIN:** A SQL clause used to combine rows from two or more tables based on a related column.
- **UNION:** A SQL operation that merges the result sets of two or more SELECT statements into a single result set.
- **Aggregate Functions:** SQL functions that perform a calculation on a set of values and return a single value (e.g., AVG, SUM, COUNT).
- **Mathematical Operations:** Operations such as addition (+), subtraction (-), multiplication (*), and division (/) that can be used directly in SQL queries for data analysis.

Practice Questions (Multiple Choice):
Question 1 - What does the `AVG` function calculate in the context of SQL queries?

A) The total sum of a numeric column
B) The average value of a specified numeric column
C) The maximum value found in a specified column
D) The total count of rows that match a criterion

Answer: B) The average value of a specified numeric column

Question 2 - Which SQL function is used to round a number to a specified number of decimal places?

A) `ROUND`
B) `TRUNC`

C) `CUT`
D) `FIX`

Answer: A) `ROUND`

Question 3 - How can you extract the month from a `rental_date` column in the `rental` table?

A) `SELECT DATE_PART('month', rental_date) FROM rental;`
B) `SELECT EXTRACT(MONTH FROM rental_date) FROM rental;`
C) `SELECT MONTH(rental_date) FROM rental;`
D) `SELECT GET_MONTH(rental_date) FROM rental;`

Answer: B) `SELECT EXTRACT(MONTH FROM rental_date) FROM rental;`

Question 4 - If you want to find the total length of films for a specific category, which of the following SQL clauses is essential to include in your query?

A) `GROUP BY`
B) `ORDER BY`
C) `JOIN`
D) `LIMIT`

Answer: C) `JOIN`

Question 5 - What is the result of the following SQL statement: `SELECT SQRT(AVG(length)) FROM film;`?

A) It calculates the square root of the length of each film.
B) It finds the average length of all films and then calculates the square root of this average.
C) It adds all film lengths together and calculates the square root of the total.
D) It calculates the average square root length of all films.

Answer: B) It finds the average length of all films and then calculates the square root of this average.