UNIT1: INTRODUCTION TO DATA AND DATABASES

UNIONS, EXTRACT, DATE, MATH

LAGUARDIA COMMUNITY COLLEGE

Today's Fun Fact:

Did you know that SQL has been used in space? The Hubble Space Telescope uses an SQL database to store and manage the amount of data it collects using JSONS files.

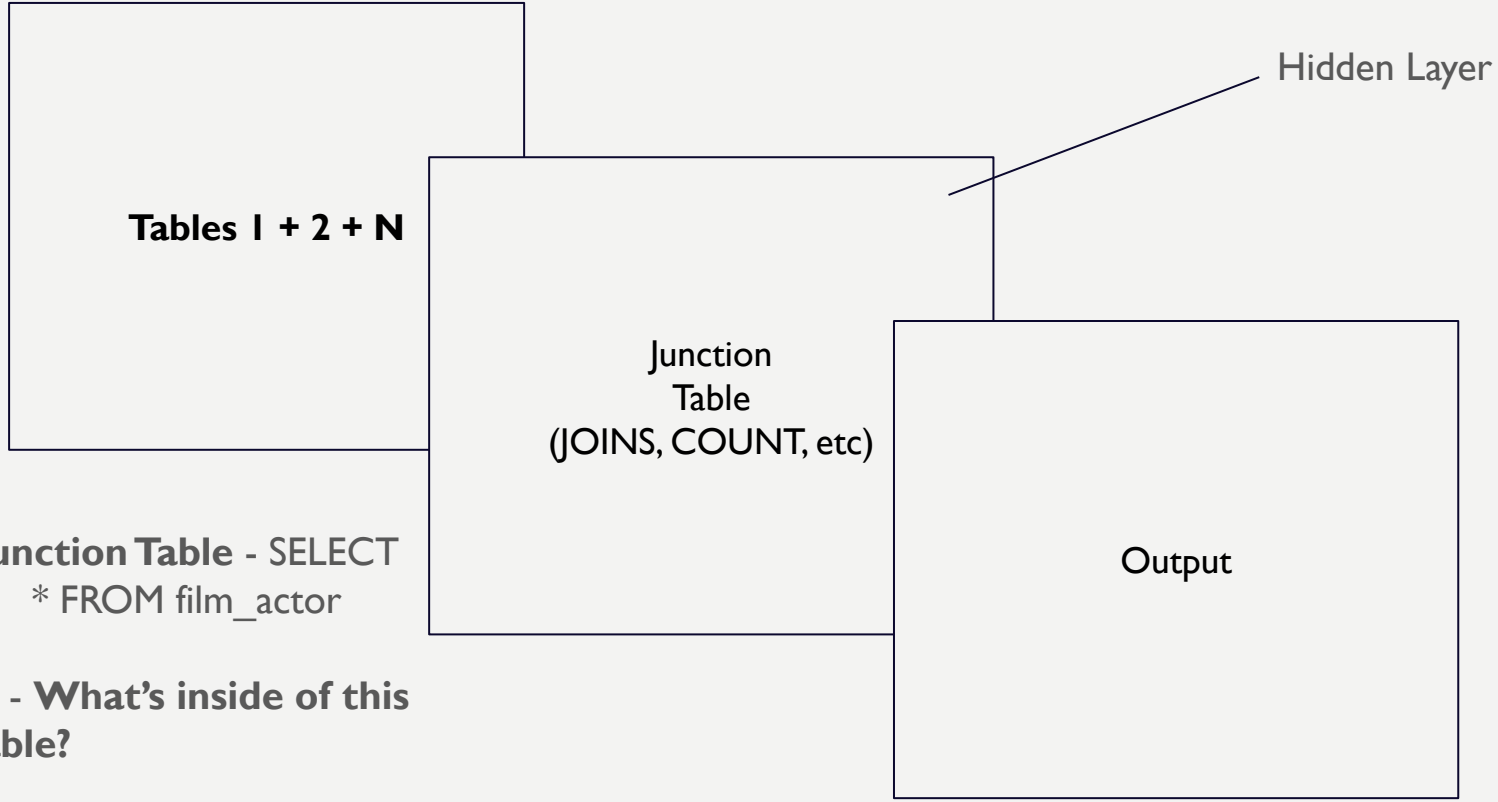
Week 3

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 1 (Posted on Wednesday)

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.

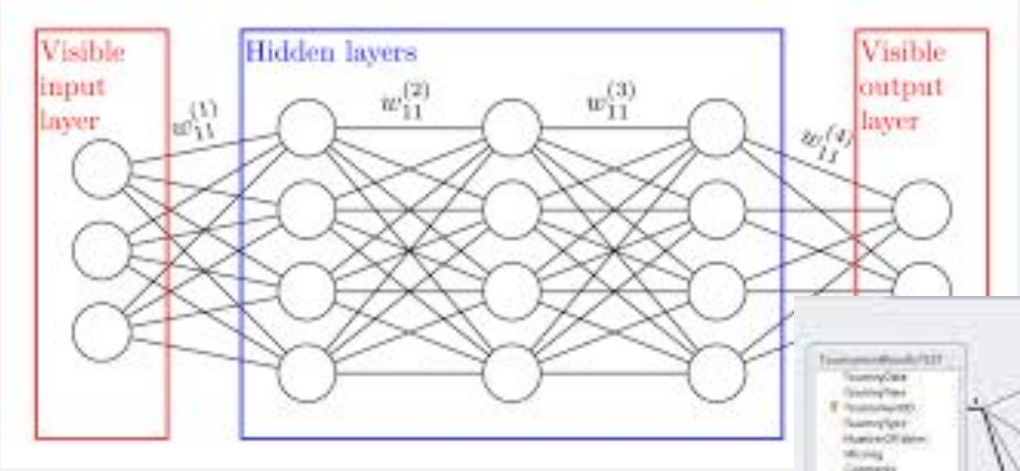
JOINS - What is a Junction Table?



Junction Table - SELECT * FROM film_actor

Q - What's inside of this table?

JOINS - Hidden Functions

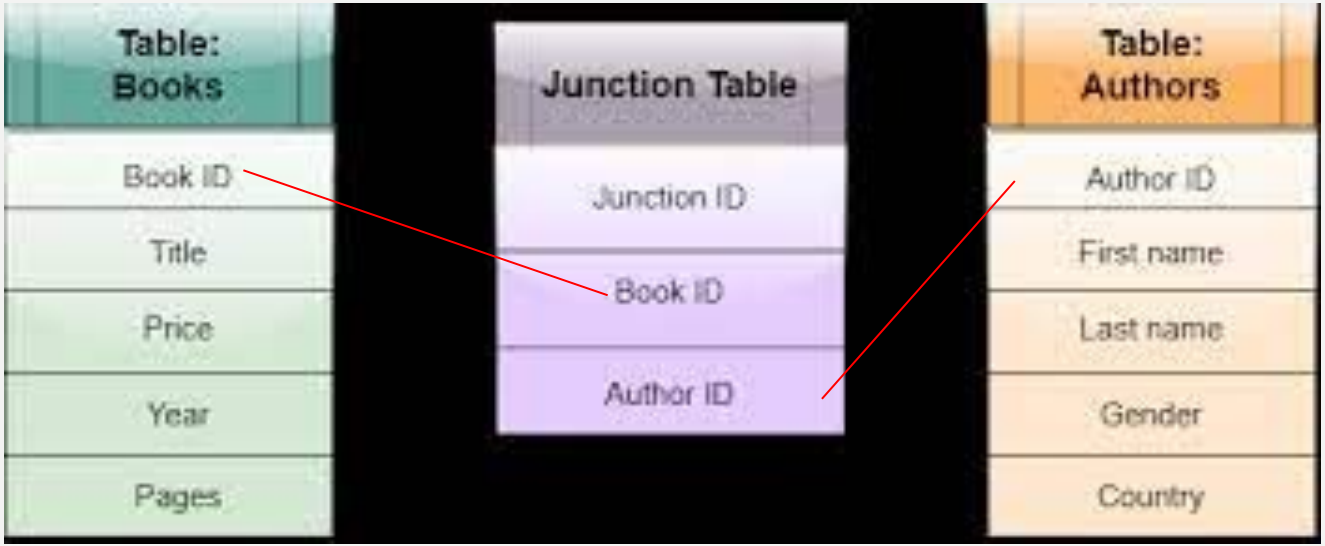


Artificial Intelligence Computing

SQL Computing

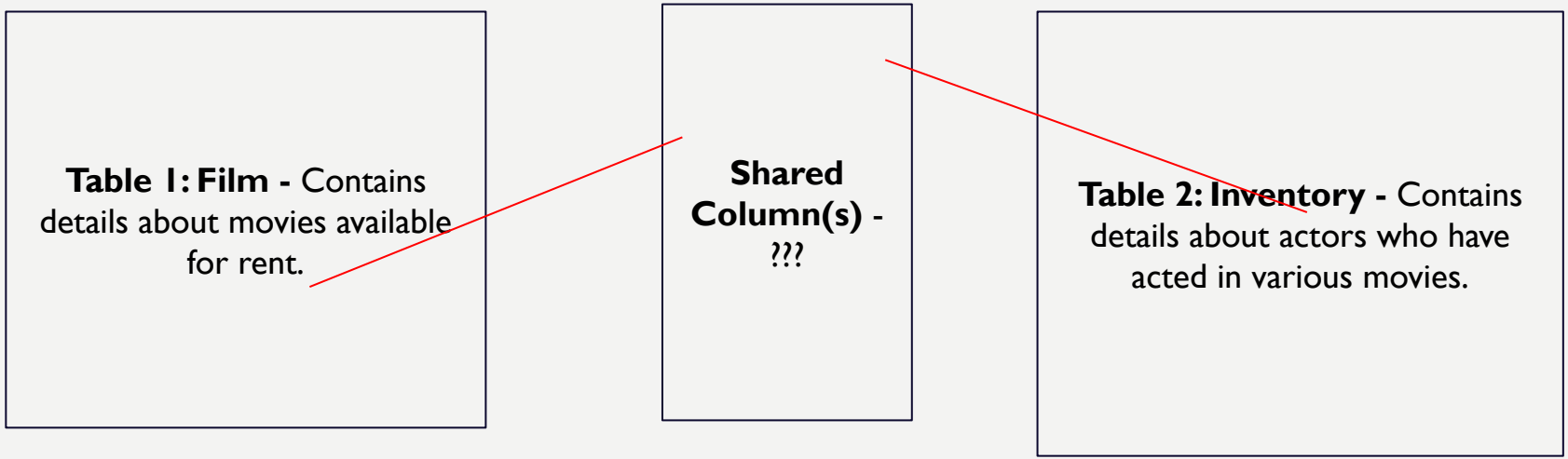
JOINS - What is a Junction Table?

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.



JOINS - What is 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.



JOINS - What is 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.

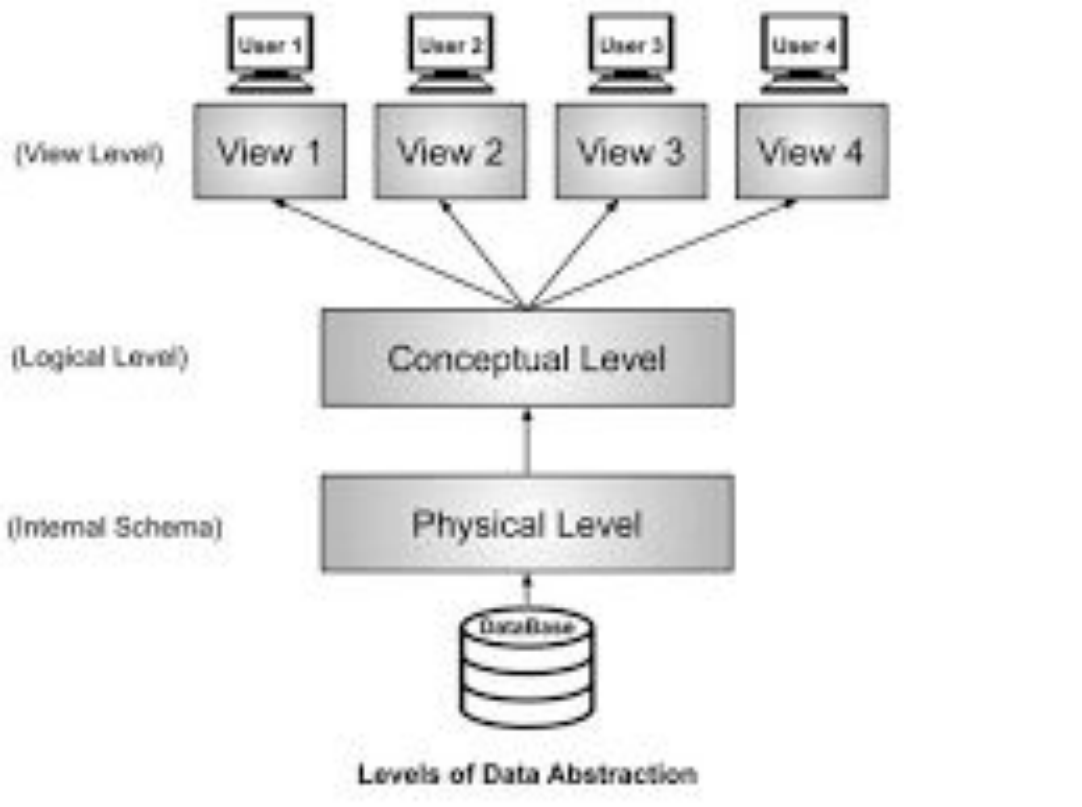
Table 1: Film - Contains details about movies available for rent.

Shared Column(s) - film_id, last_update

Table 2: Inventory - Contains details about actors who have acted in various movies.

JOINS (ABSTRACTION)

Abstraction in SQL is the process of hiding the details of how data is stored and retrieved from the user. This allows users to focus on what they want to do with the data, rather than how it is stored.



JOINS - Creating a Junction Table (Class 3B)

SQL JOINS – 8 TYPES

There are 4 types of SQL joins

- Inner
- Left
- Right
- Outer

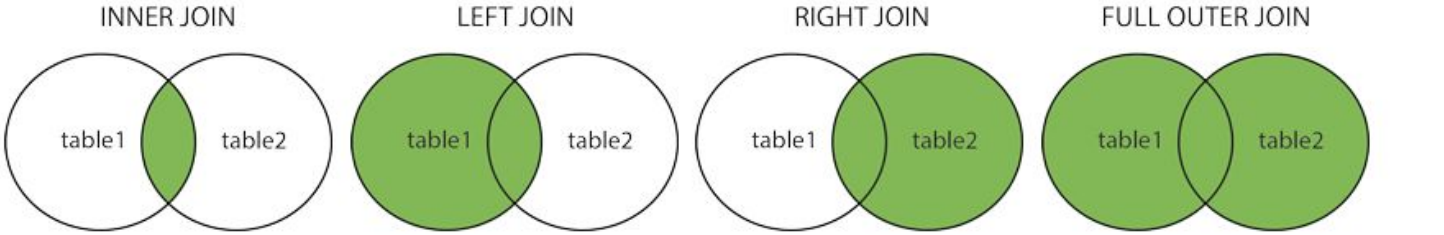
Other JOIN clauses

- ANTI
- Cross
- Self (NEXT CLASS)
- Natural
- Equi Join

JOINS - INNER

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

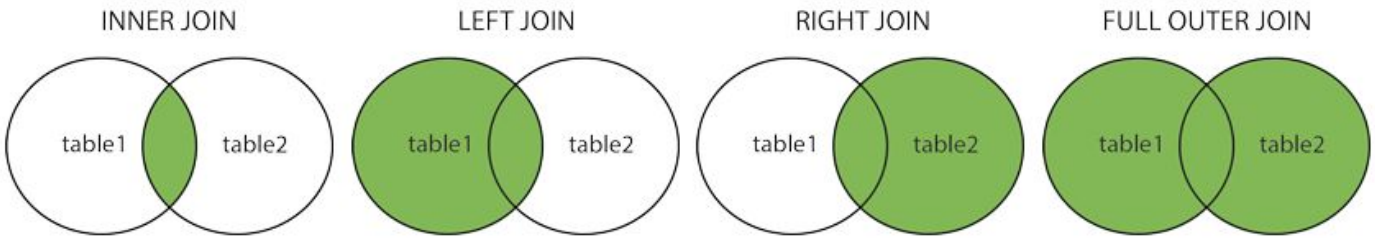
- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
-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



JOINS - LEFT

```
SELECT film.title, language.name
FROM film
LEFT JOIN language ON film.language_id = language.language_id
ORDER BY film.title;
```

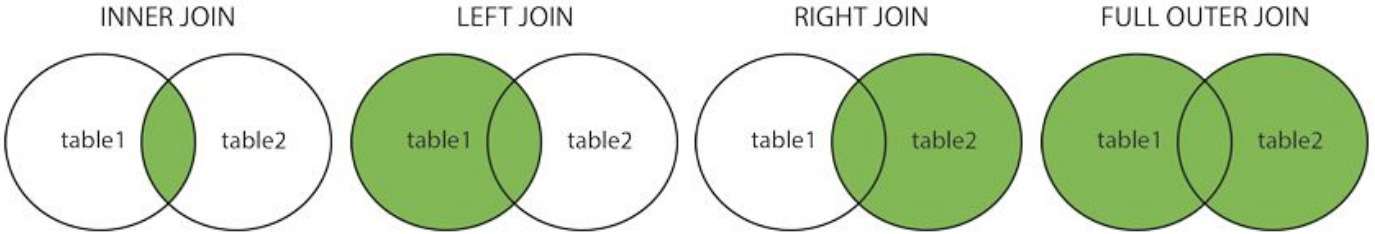
- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
-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



JOINS - RIGHT

```
SELECT language.name, film.title
FROM language
RIGHT JOIN film ON language.language_id = film.language_id
ORDER BY language.name;
```

- **(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

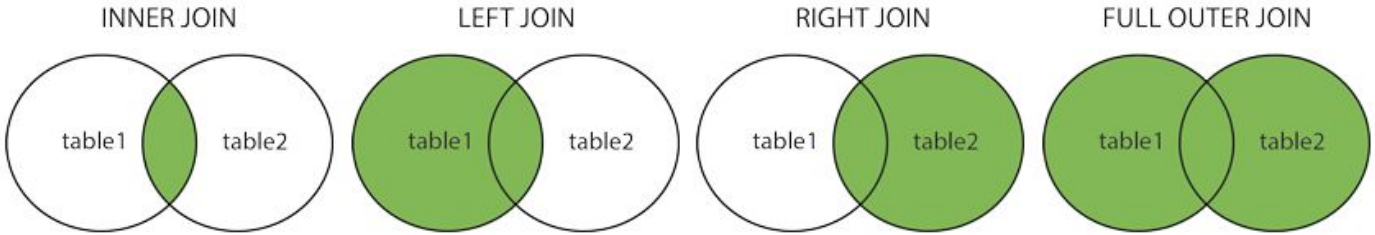


JOINS - FULL

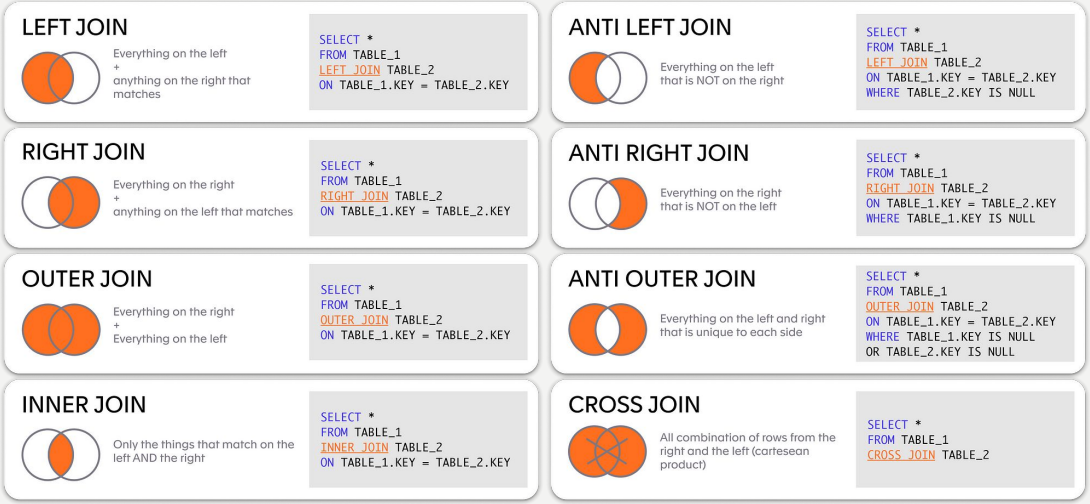
```
SELECT actor.first_name, actor.last_name, film.title
FROM actor
FULL JOIN film_actor ON actor.actor_id =
film_actor.actor_id
FULL JOIN film ON film_actor.film_id = film.film_id
ORDER BY actor.last_name, film.title;
```

Actor Info (first and last name) = specific film title
ACTOR = SOMETHING = FILM
ACTOR ID = ACTOR ID in <film_actor> = film_id in <film_actor> = film_id in film

- **(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

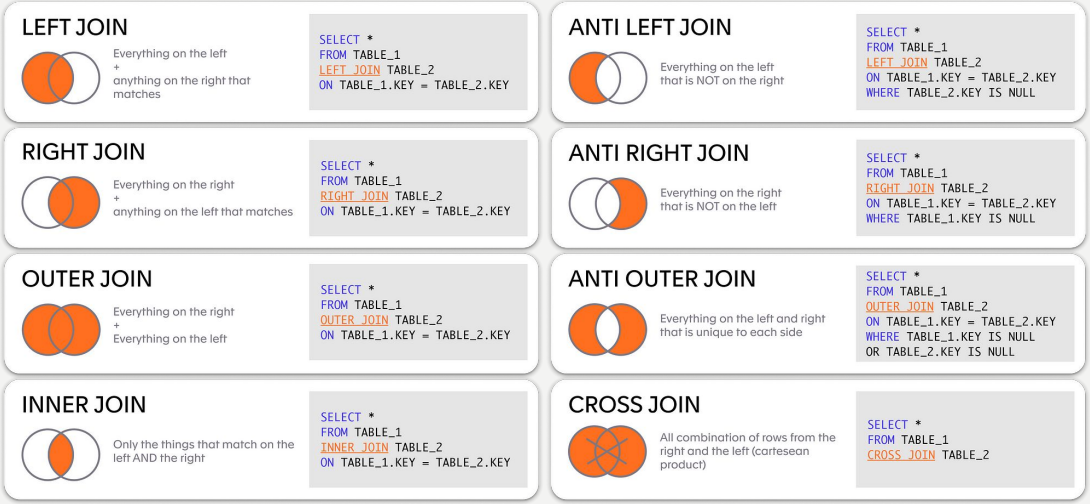


JOINS - CROSS




```
SELECT actor.first_name, actor.last_name, category.name
AS category_name
FROM actor
CROSS JOIN category
ORDER BY actor.last_name, category.name;
```

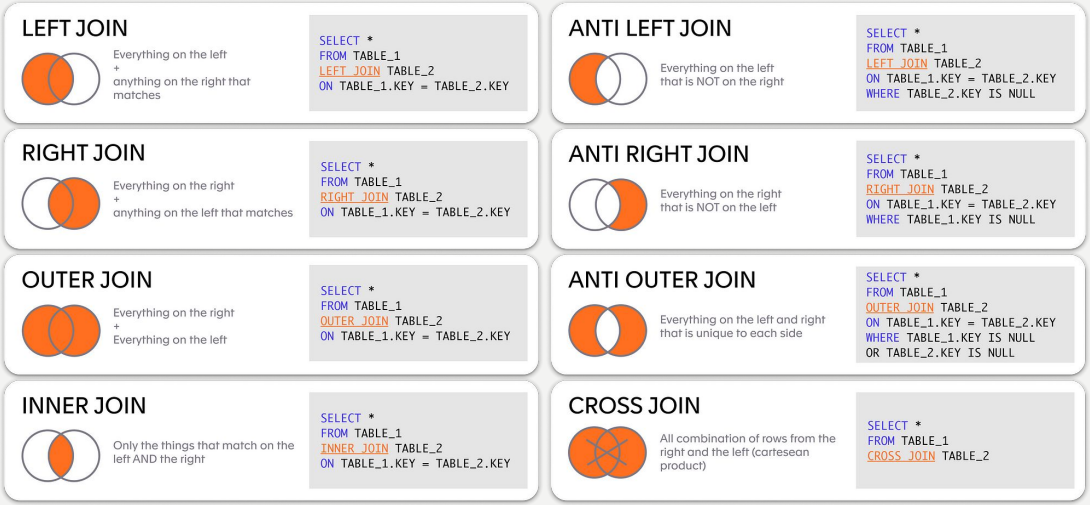
JOINS - ANTI




```
SELECT actor.actor_id,
actor.first_name, actor.last_name
FROM actor
LEFT JOIN film_actor ON
actor.actor_id =
film_actor.actor_id
WHERE film_actor.actor_id IS
NULL;
```

```
SELECT actor.actor_id,
actor.first_name, actor.last_name
FROM actor
WHERE NOT EXISTS (
   SELECT 1
   FROM film_actor
   WHERE film_actor.actor_id =
actor.actor_id
);
```

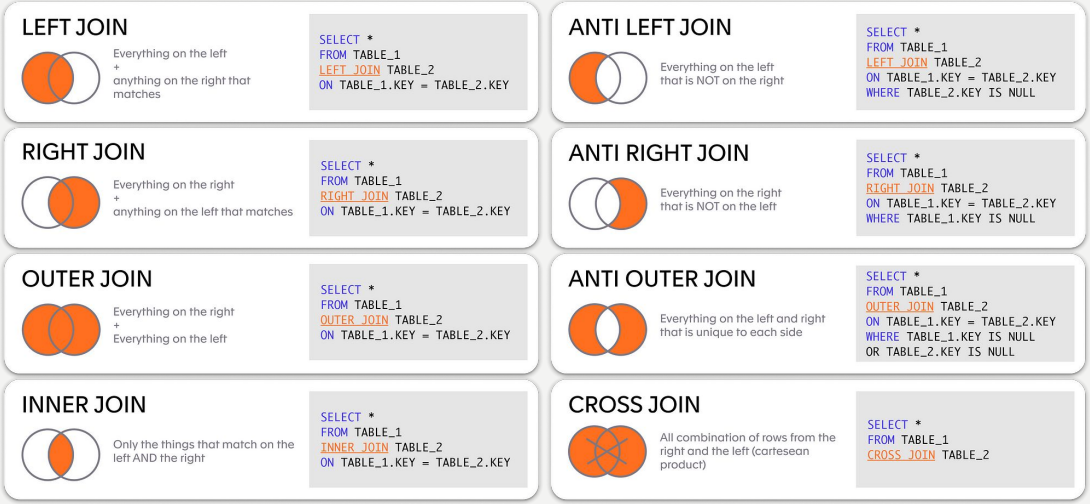
JOINS - EQUI




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

JOINS - NATURAL

SELECT film.title, language.name
FROM film
NATURAL JOIN language;



JOINS REVIEW: ACTIVITY

1. Identify movies that are currently rented out. In the `dvdrental` database, the `rental` table has `return_date` to indicate when a movie was returned. If `return_date` is NULL, it means the movie hasn't been returned yet.

2. Imagine you have a DVD rental store. You want to list all customers, including those who have not rented any DVDs yet, along with any DVDs they have rented.

3. In the same DVD rental store, you're interested in seeing all DVDs, regardless of whether they've been rented out, and details about any rentals.

4. For a promotional event, you want to create a list of all possible pairings of customers with DVDs for a special offer mail-out.

5. Your DVD rental store is planning a multicultural film night and wants to highlight movies in various languages to cater to a diverse audience. The task is to compile a list of all unique film languages available in your inventory to help with the selection process.

JOINS REVIEW: ACTIVITY

Answer 1 to beat the ‘game’; and 3 to win! (1

	title character varying (255)	first_name character varying (45)	last_name character varying (45)	rental_ timest
1	Academy Dinosaur	Dwayne	Olvera	2005-(
2	Ace Goldfinger	Brandon	Huey	2006-(
3	Affair Prejudice	Carmen	Owens	2006-(
4	African Egg	Seth	Hannon	2006-(
5	Ali Forever	Tracy	Cole	2006-(
6	Alone Trip	Marcia	Dean	2006-(
7	Amadeus Holy	Cecil	Vines	2006-(
8	American Circus	Marie	Turner	2006-(
9	Amistad Midsummer	Joe	Gilliland	2006-(

Total rows: 183 of 183 | Query complete 00:00:00.277 | Ln 7, Col 3

Format for Answering Questions:

1. What is being asked here? (i.e. objective of question)
2. What should my output look like?
3. Type of JOIN and/or other Clauses
4. Query & Output

Example: You're tasked with identifying customers who have not yet rented any movies.

1. Create a list of all customers who have not rented any film yet.
2. The output should be a list of customers, including their `customer_id`, `first_name`, and `last_name`, who have no entries in the `rental` table associated with their `customer_id`.
3. NO JOIN CLAUSE | WHERE, FROM, SELECT, NOT EXISTS
4. Query:

```
SELECT customer_id, first_name, last_name
FROM customer
WHERE NOT EXISTS (
   SELECT 1
   FROM rental
   WHERE rental.customer_id = customer.customer_id
);
```

JOINS REVIEW: Group 1

Answer 1 to beat the ‘game’; and 3 to win! (15 Minutes)

JOINS REVIEW: Group 2

Answer 1 to beat the ‘game’; and 3 to win! (15 Minutes)

JOINS REVIEW: Group 3

Answer 1 to beat the ‘game’; and 3 to win! (15 Minutes)

JOINS REVIEW: Group 4

Answer 1 to beat the ‘game’; and 3 to win! (15 Minutes)

JOINS REVIEW: *Answers*

1. Identify Movies that are currently rented out.

```
SELECT film.title
FROM rental
JOIN inventory ON
rental.inventory_id =
inventory.inventory_id
JOIN film ON inventory.film_id =
film.film_id
WHERE rental.return_date IS
NULL;
```

2. List all customers, including those who have not rented any DVDs yet, along with any DVDs they have rented

```
SELECT customer.customer_id,
customer.first_name,
customer.last_name, film.title
FROM customer
LEFT JOIN rental ON
customer.customer_id =
rental.customer_id
LEFT JOIN inventory ON
rental.inventory_id =
inventory.inventory_id
LEFT JOIN film ON
inventory.film_id = film.film_id;
```

3. See all DVDs, regardless of whether they've been rented out, and details about any rentals

```
SELECT film.title,
rental.rental_date,
customer.first_name,
customer.last_name
FROM film
LEFT JOIN inventory ON
film.film_id = inventory.film_id
LEFT JOIN rental ON
inventory.inventory_id =
rental.inventory_id
LEFT JOIN customer ON
rental.customer_id =
customer.customer_id;
```

JOINS REVIEW: *Answers*

4. Create a list of all possible pairings of customers with DVDs for a special offer mail-out SELECT customer.first_name, customer.last_name, film.title FROM customer CROSS JOIN film;	5. Compile a list of all unique film languages available in your inventory SELECT DISTINCT language.name FROM film JOIN language ON film.language_id = language.language_id; OR SELECT DISTINCT name FROM language;

Did someone say BREAK?!

Shake out, and stay hydrated!

Return at: XXX

SQL FUNDAMENTALS-

UNIONS

LAGUARDIA COMMUNITY COLLEGE

UNIONS

Unions are often used to combine data from similar tables that aren't perfectly normalized.

1. Distinct Results: By default, `UNION` removes duplicate rows between the combined queries. Each row in the result set is unique unless `UNION ALL` is used, which retains duplicates.

2. Same Number and Type of Columns: The `SELECT` statements being combined must have the same number of columns in their result sets, and the corresponding columns must have compatible data types to be matched correctly.

3. Order of Results: Although each individual `SELECT` statement within the `UNION` can have its own `ORDER BY` clause, typically, a single `ORDER BY` clause is applied to the entire result set of the `UNION` at the end of the statement to sort the combined results.

4. Use Cases: `UNION` is particularly useful when you need to aggregate data from similar tables or datasets that are stored separately but have the same structure, or when you want to aggregate results from different queries for comparative or comprehensive analysis.

SQL UNIONS

The UNION operator is used to combine the result-set of two or more SELECT statements.

Each SELECT statement within UNION must have the same number of columns

The columns must also have similar data types

The columns in each SELECT statement must also be in the same order

UNIONS

The UNION removes all duplicate rows – unless UNION ALL is used.

The UNION may place rows in the first query before, after, or between the rows in the result-set of the second query.

To sort rows in the combined result-set by a specific column, use the ORDER BY clause.

UNION SYNTAX

SELECT *column_name(s)* FROM *table1*
UNION
SELECT *column_name(s)* FROM *table2*

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL

SELECT *column_name(s)* FROM *table1*
UNION ALL
SELECT *column_name(s)* FROM *table2*

UNION EXAMPLES

Example 1: Suppose you want to create a comprehensive list of both actors and staff members, simply by their names. This could be useful for a newsletter or a general directory.

```
SELECT first_name || ' ' || last_name AS name
FROM actor
UNION
SELECT first_name || ' ' || last_name
FROM staff;
```

Example 2: If you're interested in creating a list that combines all film titles and all category names (perhaps for a tagging or categorization exercise), you can use `UNION` to aggregate this information.

```
SELECT title AS text
FROM film
UNION
SELECT name
FROM category;
```

Example 3: For a global analysis or report, you might want to aggregate all cities and countries from the `city` and `country` tables into a single list.

```
SELECT city AS location
FROM city
UNION
SELECT country
FROM country;
```