UNIT1: INTRODUCTION TO DATA AND DATABASES

JOINS

LAGUARDIA COMMUNITY COLLEGE

Today’s Fun Fact - SQL was Invented In 1970, and It was originally called SEQUEL, but later shortened

to SQL.

Housekeeping Items

Upcoming Assignments (Required):

- Assignment 2A - Due Sunday, August 25th, 11:59PM
- Assignment 2B - Due Sunday, August 25th, 11:59PM

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.

Important Notes:

- Class Recordings will be posted within 12-24 Hours after class.

Assignment 2B - Due Sunday

Relational Databases in the Real World

What is an Alias?

An alias is like a nickname for your tables or columns in SQL. When you're working with queries, especially complex ones with joins or multiple tables, names can get long or confusing. Aliases help you simplify these names and make your queries easier to read and write.

1. Simplicity: They shorten your SQL syntax, making it cleaner.
2. Clarity: They help clarify the purpose of a column or table, especially if the original name is cryptic.
3. Necessity: In some cases, especially with subqueries or when joining tables that have columns with the same name, aliases are needed to distinguish between them.

SELECT first_name AS fname, last_name AS lname FROM actor;

Column Alias

SELECT a.actor_id, a.first_name, a.last_name FROM actor AS a; - Table Alias

What is an Alias?

1) Column Aliases: Let's say you have a table named `employee_info` with a column called `employee_first_name`. Writing this out every time can be tedious, especially in complex queries. So, you use an alias.

Without Alias - SELECT employee_first_name FROM employee_info;

With Alias - SELECT employee_first_name AS first_name FROM employee_info;

Here, `AS first_name` creates an alias for `employee_first_name`, making it shorter and easier to reference.

What is an Alias?

2) Table Aliases: Table aliases are especially useful in joins where you reference tables multiple times.

Without Alias - SELECT employee_info.employee_first_name, department_info.department_name

FROM employee_info

JOIN department_info ON employee_info.department_id = department_info.department_id;

With Alias - SELECT ei.employee_first_name, di.department_name

FROM employee_info AS ei

JOIN department_info AS di ON ei.department_id = di.department_id;

In this example, `employee_info` is aliased as `ei` and `department_info` as `di`. This makes the `JOIN` clause and the `SELECT` clause much easier to read and write.

SQL JOINS

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Why do you think this concept might be useful?

SQL JOINS

- Suppose you'd like to query data from two tables within your database. In order to join these two tables together, you need to join on a field that both tables share.

For example:

users

user_id	email	name
10	sadio@example.com	Sadio
11	mo@example.com	Mohamed
12	rinsola@example.com	Rinsola
13	amalie@example.com	Amalie

orders

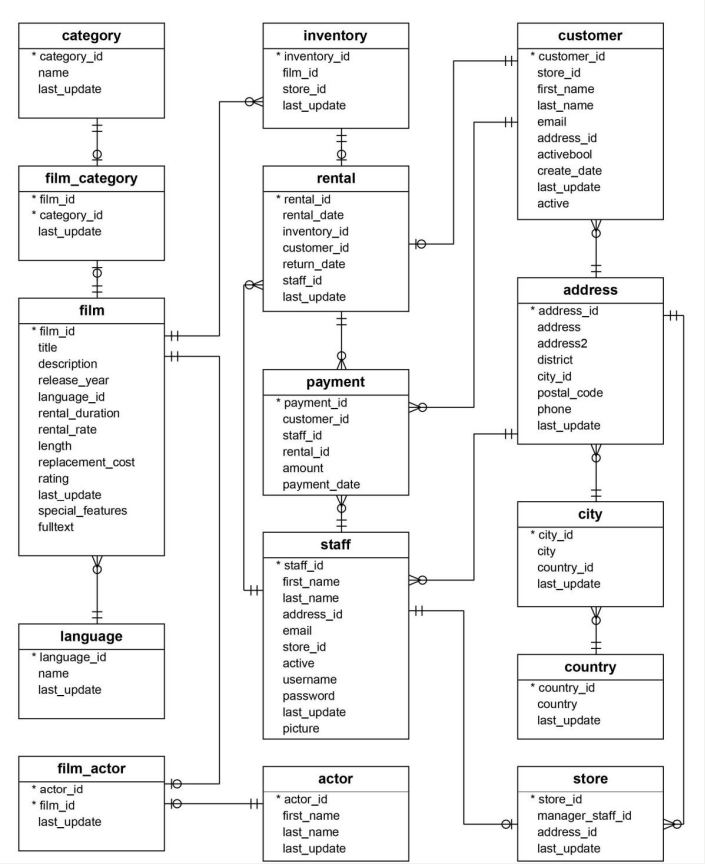
order_no	user_id	product_sku
93	11	123
94	11	789
95	13	789
96	10	101

books

product_sku	title	price
123	Aurora	15
456	Blind Lake	10
789	Invisible Planets	25
101	The Sparrow	15

Primary keys serve as unique identifiers for each row in a database table. Foreign keys link data in one table to the data in another table.

Entity Relationship (ER) Diagram



```
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;
```

BASICS FOR SQL JOINS

1. Specify the column in both tables you want to select data from in the SELECT clause
2. Specify the main table (for instance table A) in the FROM clause
3. Specify the table that the main table will join with (in this case table B) in the JOIN clause. You also add an ON Keyword

Example:

```
SELECT table1.column1, table1.column2, table2.column1, table2.column2
FROM table1
JOIN table2 ON table1.column1 = table2.column1;
```

SQL JOINS

Often times two tables will have columns that share the same name.

In order to avoid ambiguity, we need to specify as: table_name.column_name, or for example A.first_name and B.first_name.

QUICK PRACTICE:

Rewrite the following SQL query to join users and orders table:

```
WITH new_table1 AS (
        SELECT users.*, orders.*
        FROM users
        INNER JOIN orders on users.user_id = orders.user_id),
```

users

user_id	email	name
10	sadio@example.com	Sadio
11	mo@example.com	Mohamed
12	rinsola@example.com	Rinsola
13	amalie@example.com	Amalie

orders

order_no	user_id	product_sku
93	11	123
94	11	789
95	13	789
96	10	101

books

product_sku	title	price
123	Aurora	15
456	Blind Lake	10
789	Invisible Planets	25
101	The Sparrow	15

SQL JOINS – 4 TYPES

There are 4 types of SQL joins

- **Inner**
- **Left OUTER**
- Right OUTER
- FULL Outer

Each has their own slightly different syntax

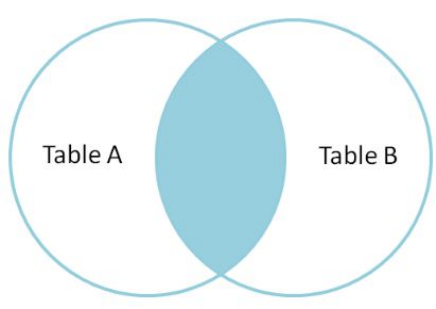
SQL JOINS – INNER JOIN

ONLY Returning the rows where there is a match in both tables. Purpose is to **ONLY** return these matches or pairs between the two tables.

- Assume we have the following two tables. **Table A** on the left and **Table B** on the right:
- Both tables have 4 records each.

id	name	id	name
1	Pirate	1	Rutabaga
2	Monkey	2	Pirate
3	Ninja	3	Darth Vader
4	Spaghetti	4	Ninja

```
SELECT * FROM TableA
INNER JOIN TableB
ON TableA.name = TableB.name
```



- What do you think will be returned from this query?

INNER JOIN EXAMPLE

ONLY Returning the rows where there is a match in both tables. Purpose is to **ONLY** return these matches or pairs between the two tables.

TEST IT OUT: SELECT c.first_name, c.last_name, r.rental_date

FROM customer AS c

INNER JOIN rental AS r ON c.customer_id = r.customer_id;

SELECT c.first_name, c.last_name, r.rental_date # Pulling Information from Table1 and Table2

FROM customer AS c # Uses an alias for customer table (c)

When the table name comes right after the FROM clause, it is Table1

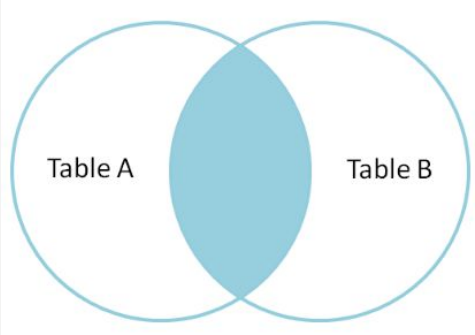
INNER JOIN rental AS r ON c.customer_id = r.customer_id; # Use INNER JOIN and shared column to form temporary relationship between the two tables

Uses an alias for rental table (r)

SQL JOINS – INNER JOIN

```
id name       id  name
-- ----       --  ----
1  Pirate     1   Rutabaga
2  Monkey     2   Pirate
3  Ninja      3   Darth Vader
4  Spaghetti  4   Ninja
```

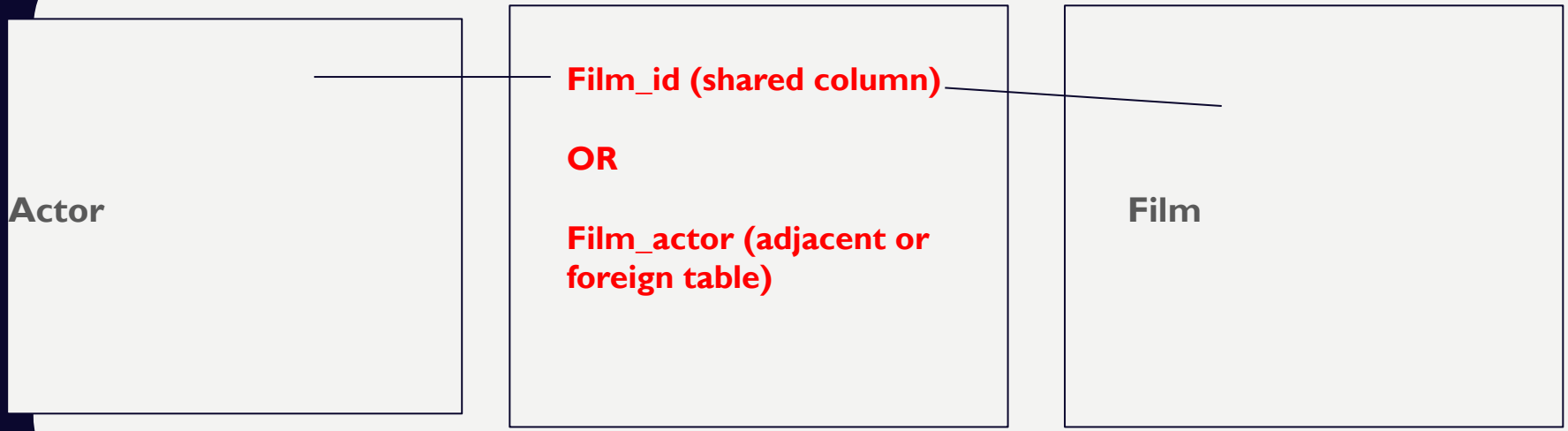
```
SELECT * FROM TableA
INNER JOIN TableB
ON TableA.name = TableB.name
```



1	Pirate	Pirate
2	Ninja	Ninja

****Inner join** produces only the set of records that match in both Table A and Table B.**

SELECT a.first_name, a.last_name, fa.film_id FROM actor a



Outputs in query environment whatever you wanted to pull (SELECT) from the query.

INNER JOIN – LET'S EXPLORE

- Take a look at our customer and payment tables.

  SELECT * FROM customer;

  SELECT * FROM payment;

  – What columns do they have in common?

INNER JOIN – LET'S EXPLORE

- Take a look at our customer and payment tables.

  SELECT * FROM customer;

  SELECT * FROM payment;

  – they both have a customer_id field

INNER JOIN – ON YOUR OWN 5-10 MINS

- Try to join the customer and payment table on the customer_id field

```
SELECT * FROM TableA
INNER JOIN TableB
ON TableA.name = TableB.name
```

INNER JOIN – SOLUTION

- Try to join the customer and payment table on the customer_id field

```
SELECT *
FROM customer
INNER JOIN payment ON payment.customer_id = customer.customer_id
```

INNER JOIN – ADD ONS: 5 MINS

- Try to add a where clause to your JOIN statement

```
SELECT *
FROM customer
INNER JOIN payment ON payment.customer_id =
customer.customer_id
WHERE ……..
```

INNER JOIN – WHERE

- Previous statements still work with joins!

```
SELECT *
FROM customer
INNER JOIN payment ON payment.customer_id =
customer.customer_id
WHERE first_name LIKE 'A%'
```

SQL OUTER JOINS

- Outer joins make up the other types of JOINs. There are three types of outer joins
  - FULL
  - LEFT
  - RIGHT

FULL OUTER JOIN

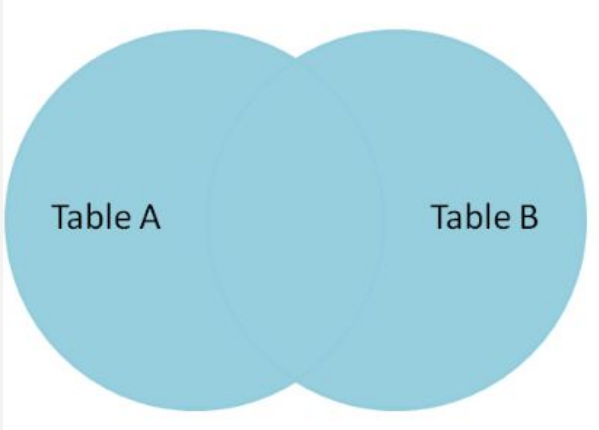
Returning all of the rows where there is a match in either the left table OR the right table. If there is no match for that specific row, it returns a NULL value for columns requested from the table that don't have a match for a specific row.

- **Full outer join** produces the set of all records in Table A and Table B, with matching records from both sides where available. If there is no match, the missing side will contain null.

id	name	id	name
1	Pirate	1	Rutabaga
2	Monkey	2	Pirate
3	Ninja	3	Darth Vader
4	Spaghetti	4	Ninja

```
SELECT * FROM TableA
FULL OUTER JOIN TableB
ON TableA.name = TableB.name
```

1	Pirate	2	Pirate
2	Monkey	null	null
3	Ninja	4	Ninja
4	Spaghetti	null	null
null	null	1	Rutabaga
null	null	3	Darth Vader



FULL OUTER JOIN

Returning all of the rows where there is a match in either the left table OR the right table. If there is no match for that specific row, it returns a NULL value for columns requested from the table that don't have a match for a specific row.

TEST IT OUT: SELECT c.first_name, c.last_name, r.rental_date
FROM customer AS c
FULL JOIN rental AS r ON c.customer_id = r.customer_id;

SELECT c.first_name, c.last_name, r.rental_date # Pulling Information from Table1 and Table2

FROM customer AS c # Uses an alias for customer table (c)

When the table name comes right after the FROM clause, it is Table1

FULL JOIN rental AS r ON c.customer_id = r.customer_id; # Use FULL JOIN and shared column to form temporary relationship between the two tables. Returning all of the ROWS with information, but putting NULL for results that don't have a specific MATCH.

Uses an alias for rental table (r)

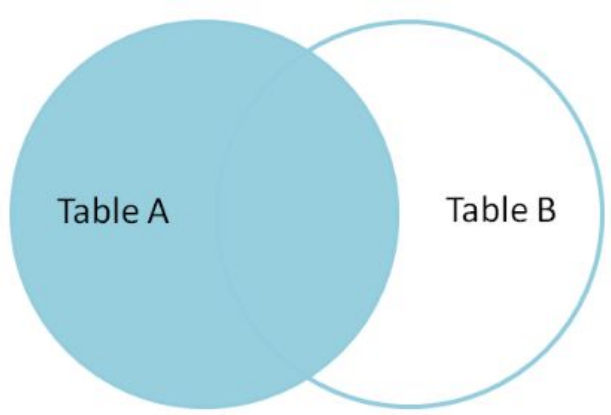
LEFT OR RIGHT OUTER JOIN

- **Left outer join** produces a complete set of records from Table A, with the matching records (where available) in Table B. If there is no match, the right side will contain null.

```
id name       id  name
-- ----       --  ----
1  Pirate     1   Rutabaga
2  Monkey     2   Pirate
3  Ninja      3   Darth Vader
4  Spaghetti  4   Ninja
```

```
SELECT * FROM TableA
LEFT OUTER JOIN TableB
ON TableA.name = TableB.name
```

```
id  name       id    name
--  ----       --    ----
1   Pirate     2     Pirate
2   Monkey     null  null
3   Ninja      4     Ninja
4   Spaghetti  null  null
```



LEFT: ONLY the rows from the LEFT TABLE (TABLE1) are being returned, and the matched rows of the RIGHT TABLE (TABLE2)

RIGHT: ONLY the rows from the RIGHT TABLE (TABLE2) are being returned, and the matched rows of the LEFT TABLE (TABLE1)

SELECT t1.c1, t2.c1, t2.c2 FROM TABLE1 # Table 1 is the main table because it comes after FROM (LEFT TABLE)
LEFT JOIN TABLE2 ON ..... # Table 2 is the secondary table because it comes after the JOIN clause

LEFT OR RIGHT OUTER JOIN

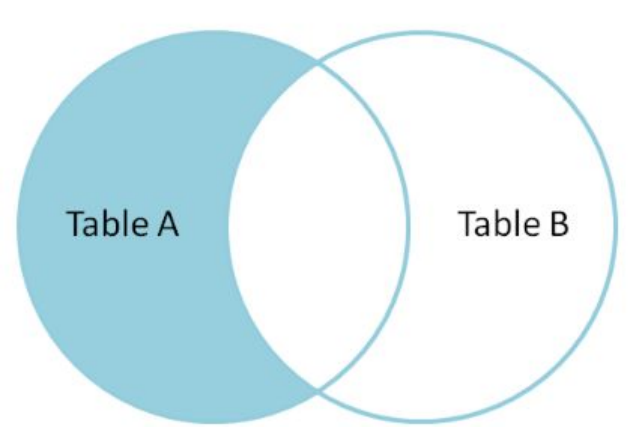
- To produce the set of records only in Table A, but not in Table B, we perform the same left outer join, then **exclude the records we don't want from the right side via a where clause**.

```
id name       id  name
-- ----       --  ----
1  Pirate     1   Rutabaga
2  Monkey     2   Pirate
3  Ninja      3   Darth Vader
4  Spaghetti  4   Ninja
```

```
SELECT * FROM TableA
LEFT OUTER JOIN TableB
ON TableA.name = TableB.name
WHERE TableB.id IS null
```

```
2   Monkey     null   null

4   Spaghetti  null   null
```



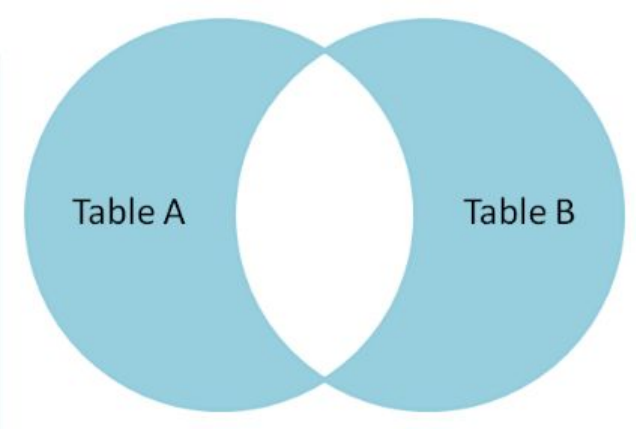
FULL OUTER JOIN

- To produce the set of records unique to Table A and Table B, we perform the same full outer join, then **exclude the records we don't want from both sides via a where clause**.

id	name	id	name
1	Pirate	1	Rutabaga
2	Monkey	2	Pirate
3	Ninja	3	Darth Vader
4	Spaghetti	4	Ninja

```
SELECT * FROM TableA
FULL OUTER JOIN TableB
ON TableA.name = TableB.name
WHERE TableA.id IS null
OR TableB.id IS null
```

2	Monkey	null	null
4	Spaghetti	null	null
null	null	1	Rutabaga
null	null	3	Darth Vader



LETS GET SOME PRACTICE … 5-10 MINS

- Let's take a look at our database… let's look at the film and inventory table
  - SELECT * FROM film LIMIT 5;
  - SELECT * FROM inventory LIMIT 5;
  - What columns do these tables have in common?

LETS GET SOME PRACTICE … 5-10 MINS

```
SELECT f.film_id,  f.title,  inventory_id
FROM film f
LEFT OUTER JOIN inventory ON inventory.film_id = film.film_id;
```

Why doesn't this field need table.column name?

LETS ADD-ON TO OUR STATEMENT… 5 MINS

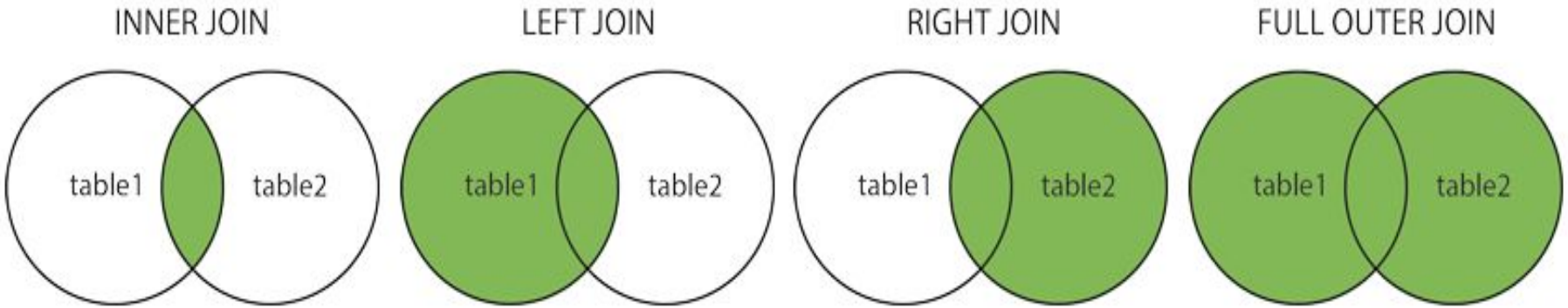
```
SELECT film.film_id,  film.title,  inventory_id
FROM film
LEFT OUTER JOIN inventory ON inventory.film_id =
film.film_id
WHERE ….
ORDER BY …
```

LETS ADD-ON TO OUR STATEMENT… EXAMPLE

```
SELECT film.film_id,  film.title,  inventory_id
FROM film
LEFT OUTER JOIN inventory ON inventory.film_id = film.film_id
WHERE inventory.film_id is NULL
ORDER BY film.film_id;
```

JOINS PRACTICE 10-15 MINS

- **(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



RECAP

1

We learned that JOINS are useful when trying to combine data from multiple tables. This allows you to expand your querying capabilities.

2

JOINS need to be joined by a common column from each table.

3

There are 4 major types of joins – INNER, LEFT, RIGHT, FULL

SQL FUNDAMENTALS-

UNIONS

LAGUARDIA COMMUNITY COLLEGE

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

UNION EXAMPLE

Customers table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Suppliers Table:

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA

The following SQL statement returns the cities (only distinct values) from both the "Customers" and the "Suppliers" table:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

City
Berlin
Mexico
London
New Orleans

UNION ALL EXAMPLE

Customers table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Suppliers Table:

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA

The following SQL statement returns the cities from both the "Customers" and the "Suppliers" table:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

City
Berlin
Mexico
Mexico
London
New Orleans

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.

UNIONS

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

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*

RECAP

- 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