Data Analytics

Combining Data With JOIN and UNIONs

GENERAL ASSEMBLY

Learning Goals

- Combine data from multiple sources using inner and left JOINs.
- Combine data using UNION and UNION ALL.
- Compare use cases for JOINs and UNIONs.

Celebrating Table Togetherness

One 2019 study found that most companies with 1,000 employees or more are pulling from 400+ data sources for business intelligence.

In fact, more than 20% of the organizations reported drawing from a whopping 1,000 or more data sources.

So, let's get comfortable bringing that data together!

Image source: The A Team Meme Generator

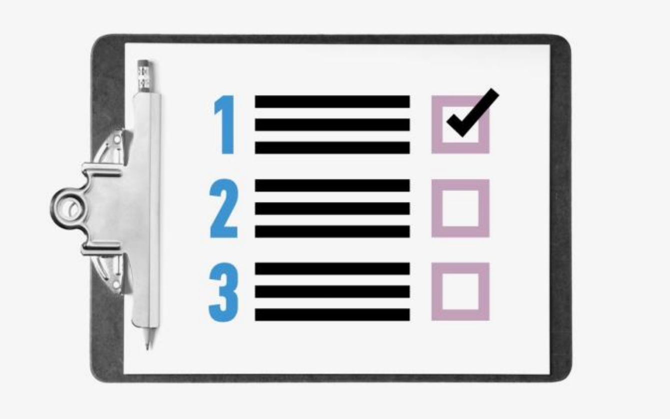
Discussion:

What Could Possibly Go Wrong?

You’ve handled a data set or two before.

Let’s make a list that addresses the following:

- What could go wrong when combining two or more data sets?
- What might you want to have control over?



Combining Data With JOINs and UNIONs

Combining Data in SQL

JOINs and UNIONs

In SQL, there are two primary methods for bringing data together:

A **JOIN** combines **columns** from tables using common unique identifiers (keys).

A **UNION** combines **rows** of *similar* data

JOINs

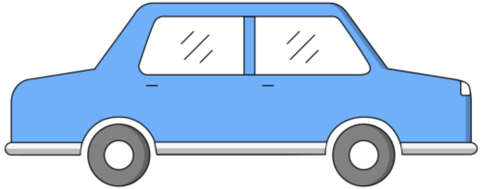
A **JOIN** combines columns from multiple tables using a common unique identifier or “key.”

drivers		
id	name	vehicle_id
1	Janet	c
2	Emily	d
3	Yoko	d
4	Ali	e

vehicles	
id	vehicle_name
a	Explorer
b	Civic
c	Corolla
d	Impala

→

id	name	vehicle_id	vehicle_name
1	Janet	c	Corolla
2	Emily	d	Impala
3	Yoko	d	Impala



UNIONs

A **UNION** combines rows from multiple tables with similar data to create a new set. Using "UNION" removes duplicates when combining the two tables.

carpoolers

id	name	vehicle_id
1	Janet	c
2	Emily	d
3	Yoko	d

monthly_parkers

id	name	vehicle_id
2	Emily	d
4	Ali	e
5	Ray	a

→

id	name	vehicle_id
1	Janet	c
2	Emily	d
3	Yoko	d
4	Ali	e
5	Ray	a

Where JOINs Live in a Query

SELECT picks the columns.

FROM points to the table.

WHERE puts filters on rows.

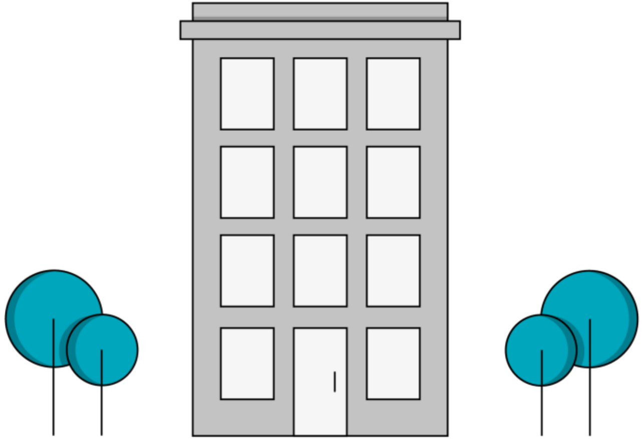
GROUP BY aggregates across values of a variable.

HAVING filters aggregated values ***after*** they have been grouped.

ORDER BY sorts the results.

LIMIT limits results to the first **n** rows.

The Query Building



Combining Data With JOINs and UNIONs

JOINs

Partner Exercise:

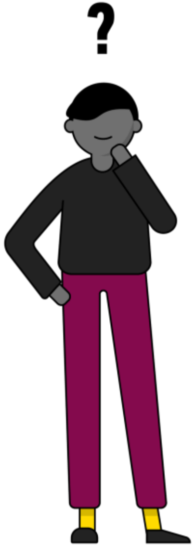
JOINs vs. UNIONs

5 minutes

With your partner, Google “database normalization” and discuss:

1. The concept of normalization.
2. Why JOINs are needed for normalized data stores.

Be prepared to share your answers with the class.



It’s Because…

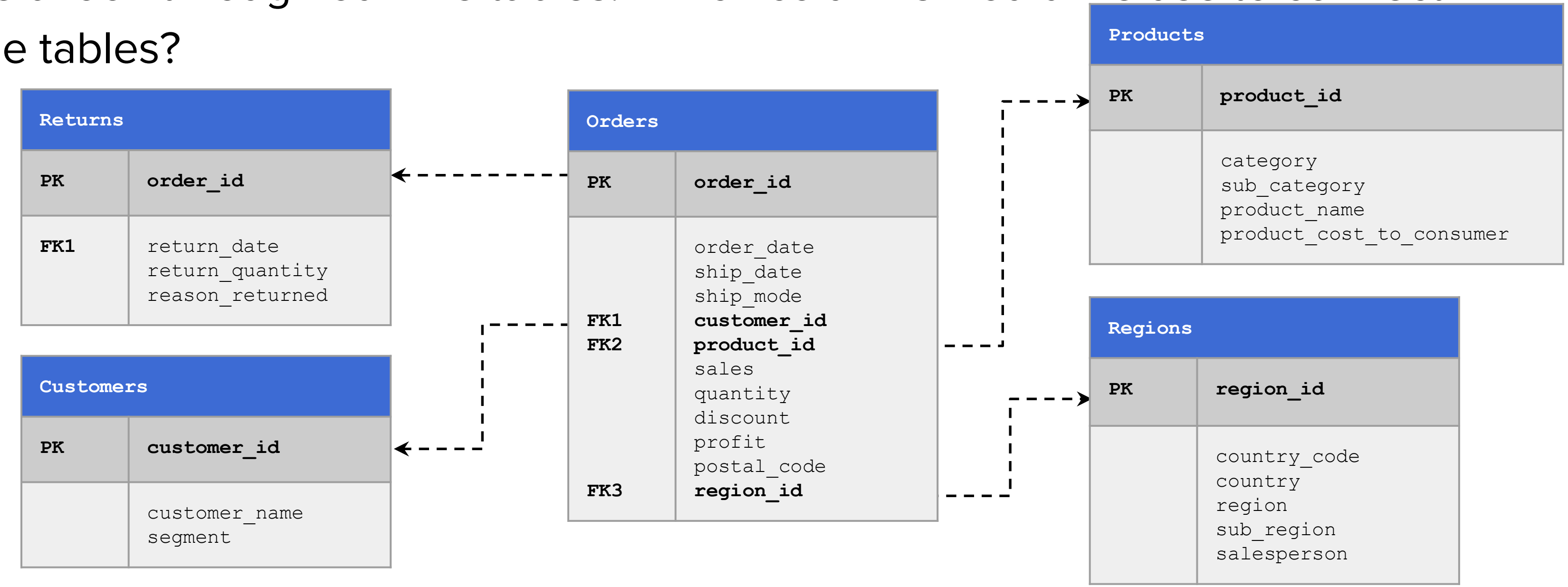
- A normalized database will seek to **separate data across multiple tables**, related to each other by keys.
- This reduces redundancy, memory footprint, and **improves speed for transactional databases.**
- These databases are typically tied to an interface where it is important for the interface application to be able to **update quickly** as data is being entered, etc.

SQL queries are most performant (memory and speed) when tables are **NARROW** (few columns) and **TALL** (many rows). This is where JOINs and UNIONs come into play!

Discussion:

Where Are Our Keys?

Take a look through our five tables. Which columns would we use to connect these tables?



Returns	
PK	**order_id**
FK1	return_date return_quantity reason_returned

Customers	
PK	**customer_id**
	customer_name segment

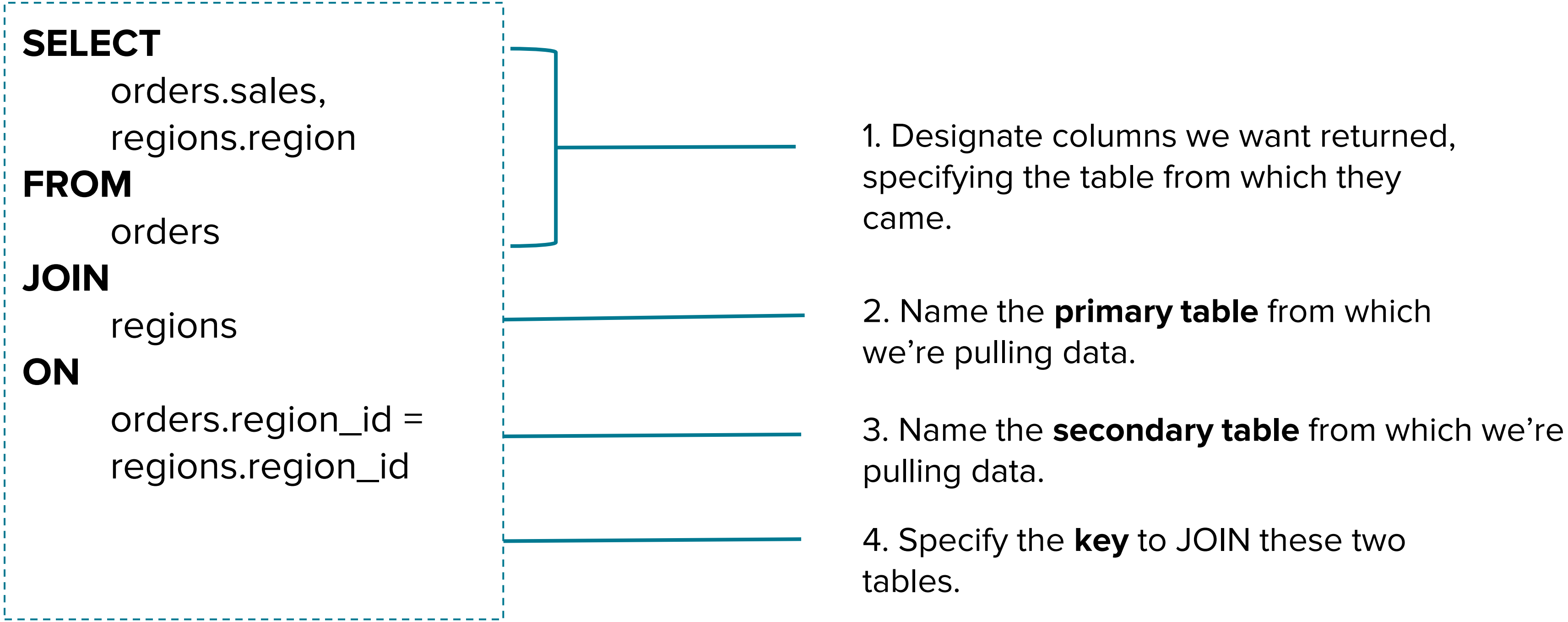
Orders	
PK	**order_id**
	order_date ship_date ship_mode
FK1	**customer_id**
FK2	**product_id**
	sales quantity discount profit postal_code
FK3	**region_id**

Products	
PK	**product_id**
	category sub_category product_name product_cost_to_consumer

Regions	
PK	**region_id**
	country_code country region sub_region salesperson

** In Orders, order_id is used to relate to other tables but is not a true primary key. It's the common link between orders and returns. To find a unique row in Orders, use a combination of order_id and product_id.

JOIN Syntax



```
SELECT
        orders.sales,
        regions.region
FROM
        orders
JOIN
        regions
ON
        orders.region_id =
        regions.region_id
```

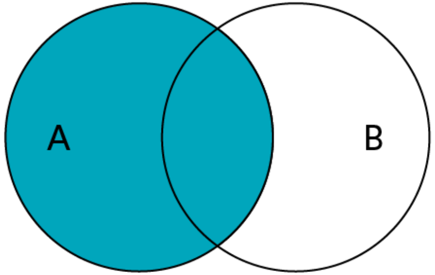
1. Designate columns we want returned, specifying the table from which they came.

2. Name the **primary table** from which we're pulling data.

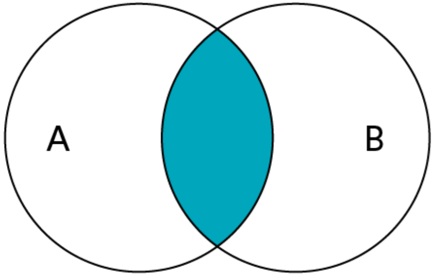
3. Name the **secondary table** from which we're pulling data.

4. Specify the **key** to JOIN these two tables.

Types of JOINs

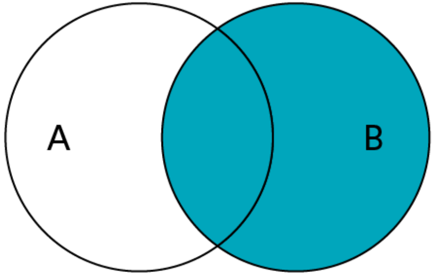


Left Join

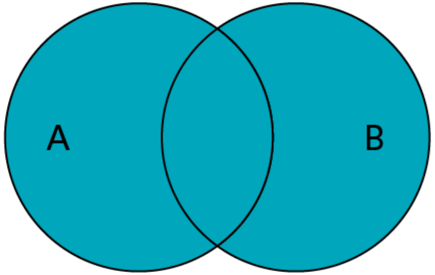


Inner Join

INNER JOIN is the same thing as JOIN



Right Join



Full Outer Join

Image source: GA Instruction

Guided Walk-Through:

Let’s Create a JOIN!

With the global expansion of Superstore, your sources of reliable data are also growing. That’s good news, right? Yes, for the most part, but…

The high volume of data can also make referencing tricky and error-prone. You got a request from your *super* boss asking you to **compare order and return dates for each order**. This requires you to pull and combine data from these two tables:

Orders		
order_id	**order_date**	**ship_date**
AE-2016-1308551	2016-09-28	2016-10-02
AE-2016-1522857	2016-09-04	2016-09-09

Returns		
order_id	**return_date**	**reason_returned**
AE-2019-1711936	2019-12-14	Not Given
AE-2019-2092798	2019-11-29	Not Given

Guided Walk-Through:

And So, a JOIN Is Born

```
SELECT DISTINCT
  orders.order_id
  , orders.order_date
  , returns.reason_returned
FROM orders
JOIN
returns ON orders.order_id =
returns.order_id
LIMIT 2;
```

Orders		
order_id	**order_date**	**ship_date**
AE-2016-1308551	2016-09-28	2016-10-02
AE-2016-1522857	2016-09-04	2016-09-09

Returns	
order_id	**return_date**
AE-2019-1711936	2019-12-14
AE-2019-2092798	2019-11-29

JOIN Result		
order_id	**order_date**	**return_date**
AE-2016-1308551	2016-09-28	2019-12-14
AE-2016-1522857	2016-09-04	2019-11-29

Working With Long Table Names

What if you're frequently referring tables with names like this one in your query?

Sales_With_Discount_Transaction_History

Imagine adding that to a column name twice as long! The solution?

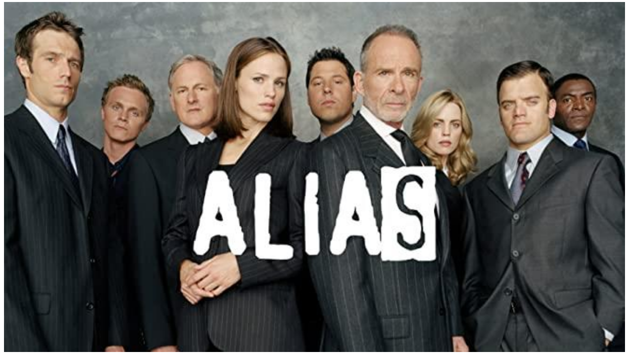


Image source: Imgflip

Shortcuts | Using an Alias

An **alias** is a shorthand name given to tables (or columns in a table) that you intend to reference repeatedly.

When creating a JOIN, each table or column can have an alias. Each column is then connected to the table by the alias.

table1 a → table1 uses the alias a.

a.column4 → column 4 is connected to table1 by the alias a.

Alias Syntax

Aliases are user-defined and designated in the FROM statement immediately following the table or column name.

Take a look at the syntaxes below. Notice that AS is in brackets because it is optional — you don't need it to designate an alias.

Alias for tables:

```
table_name [AS] alias_name
```

Alias for columns:

```
column_name [AS] alias_name
```

Guided Walk-Through:

Aliases in a Query

```
SELECT
        orders.order_id,
        orders.order_date,
        returns.return_date

FROM
        orders

INNER JOIN returns

ON orders.order_id =
returns.order_id;
```

Let’s use an alias in this query from earlier. First, designate the aliases in **FROM**.

- Orders table will be a.
- Returns table will be b.

Next, specify the connection, by column name, on which you want to link tables:

- **ON** a.column_name = b.column_name with alias for source table.
- **USING** (column_name) only if the columns have same name in each table.

Guided Walk-Through:

Aliases in a Query | **Solution**

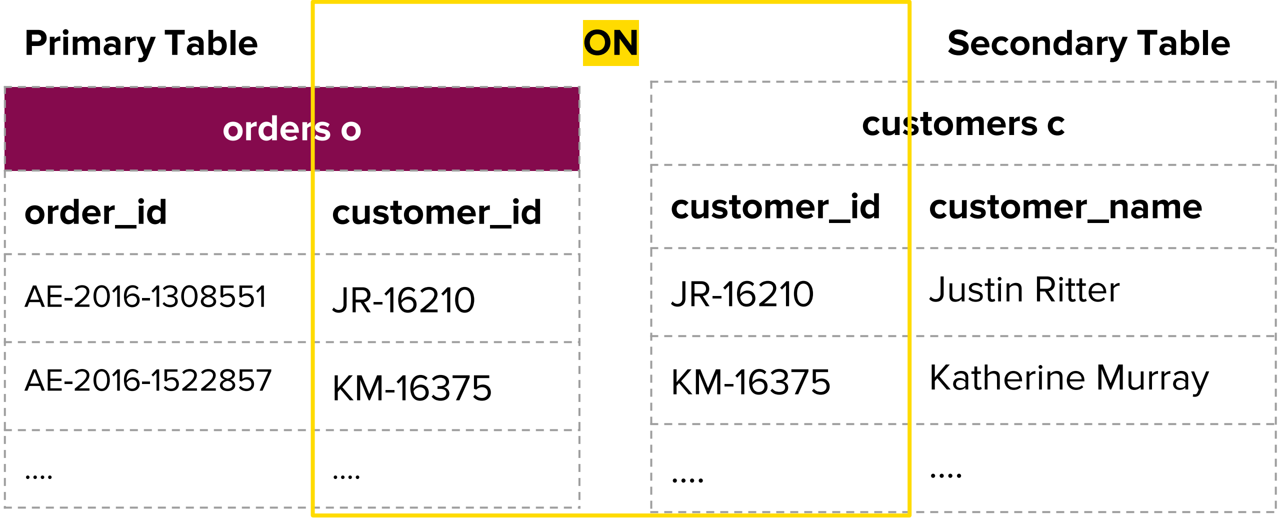
```
SELECT
        a.order_id,
        a.order_date,
        b.return_date

FROM
        orders a

INNER JOIN returns b

ON a.order_id = b.order_id;
```

This is what your query should look like with an alias for each table. Keep in mind that:

- The renaming is only temporary, and that table name does not change in the original database.
- Aliases work well when there are multiple tables in a query.

Wireframing JOINs | Single Tables

You may find drawing out tables (like below) can help you conceptualize how you plan to JOIN them. Remember, wireframes do not have to be super detailed.



orders o	
order_id	**customer_id**
AE-2016-1308551	JR-16210
AE-2016-1522857	KM-16375
….	….

customers c	
customer_id	**customer_name**
JR-16210	Justin Ritter
KM-16375	Katherine Murray
….	….

Partner Exercise:

JOINing Single Tables

5 minutes

Working with your partner, use Orders as the primary table and JOIN the Customers table. Your query should:

1. Include **order_id** from the Orders table, and **customer_name** from the Customers table.
2. Use aliases for the tables.
3. Limit the results to 100 rows.

Before going into SQL, practice wireframing your JOINs on a piece of paper.

Partner Exercise:

JOINing Single Tables | **Solution**

Solution Query

```
SELECT
  o.order_id
  , c.customer_name
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
LIMIT 100
```

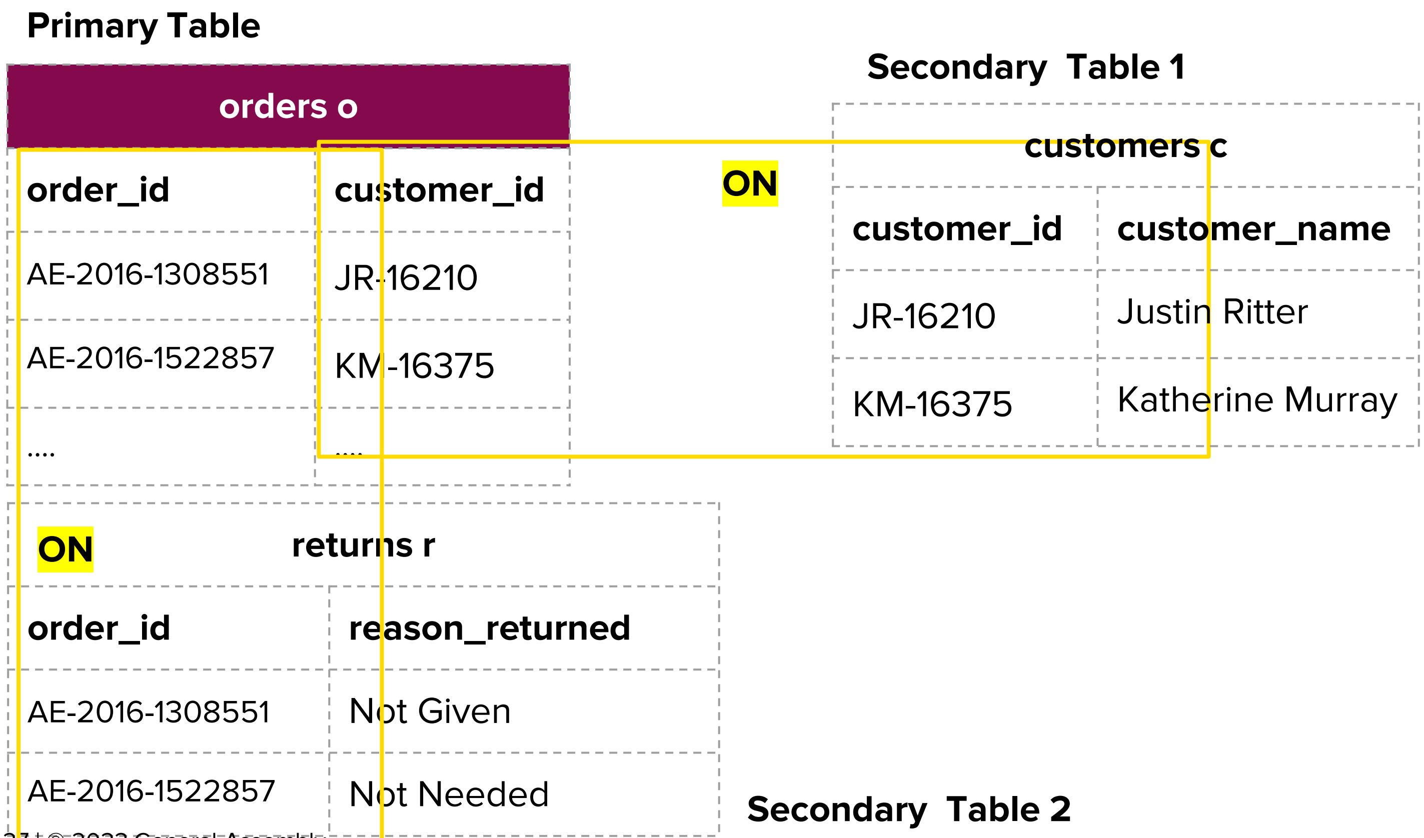
JOINing Multiple Tables

You can also JOIN multiple tables together. Here is an example — notice that we have *two* JOIN statements.

Syntax: JOIN syntax restarts when you add on a new table:

```
SELECT a.field3, a.field4, b.field1, c.field4
FROM table1 a
    JOIN table2 b ON a.field1 = b.field1
    JOIN table3 c ON a.field2 = c.field1
ORDER BY b.field1
```

Wireframing JOINs | Multiple Tables



Primary Table

orders o	
order_id	customer_id
AE-2016-1308551	JR-16210
AE-2016-1522857	KM-16375
….	….

ON

Secondary Table 1

customers c	
customer_id	customer_name
JR-16210	Justin Ritter
KM-16375	Katherine Murray

ON

returns r	
order_id	reason_returned
AE-2016-1308551	Not Given
AE-2016-1522857	Not Needed

Secondary Table 2

Partner Exercise:

JOINing Multiple Tables

5 minutes

Using **Orders** as our primary table, JOIN *both* the **Returns** *and* the **Customers** tables.

Before going into SQL, practice wireframing your JOINs on a piece of paper.

Your query should:

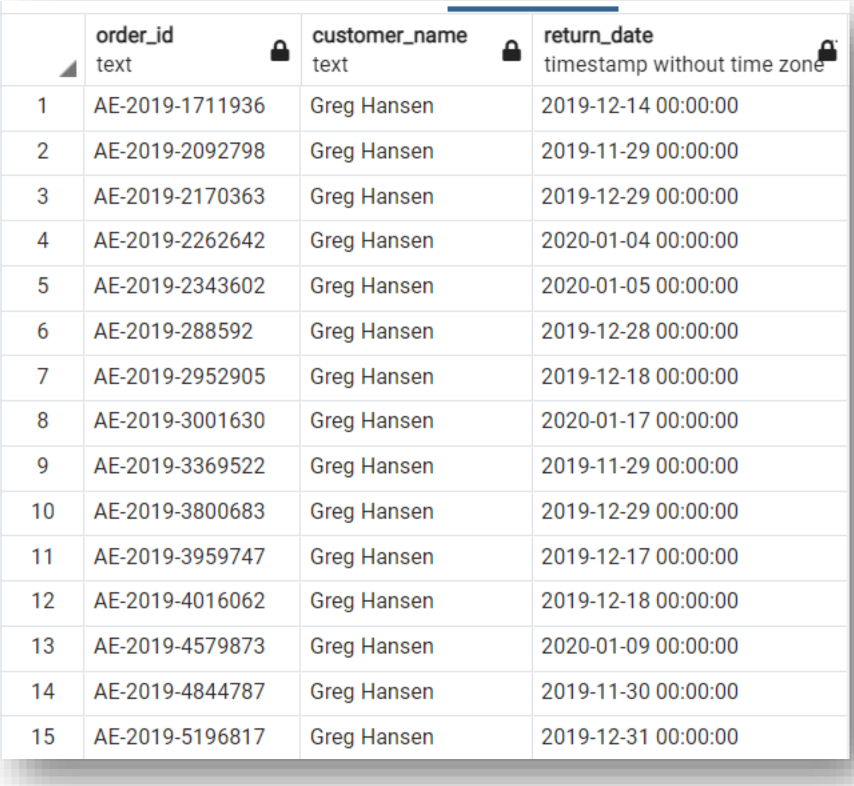
1. Include **order_id** from the orders table, **customer_name** from the Customers table, and **reason_returned** from the Returns table.
2. Limit results to 100 rows.

Partner Exercise:

JOINing Multiple Tables

5 minutes

Desired Data Output



	order_id text	customer_name text	return_date timestamp without time zone
1	AE-2019-1711936	Greg Hansen	2019-12-14 00:00:00
2	AE-2019-2092798	Greg Hansen	2019-11-29 00:00:00
3	AE-2019-2170363	Greg Hansen	2019-12-29 00:00:00
4	AE-2019-2262642	Greg Hansen	2020-01-04 00:00:00
5	AE-2019-2343602	Greg Hansen	2020-01-05 00:00:00
6	AE-2019-288592	Greg Hansen	2019-12-28 00:00:00
7	AE-2019-2952905	Greg Hansen	2019-12-18 00:00:00
8	AE-2019-3001630	Greg Hansen	2020-01-17 00:00:00
9	AE-2019-3369522	Greg Hansen	2019-11-29 00:00:00
10	AE-2019-3800683	Greg Hansen	2019-12-29 00:00:00
11	AE-2019-3959747	Greg Hansen	2019-12-17 00:00:00
12	AE-2019-4016062	Greg Hansen	2019-12-18 00:00:00
13	AE-2019-4579873	Greg Hansen	2020-01-09 00:00:00
14	AE-2019-4844787	Greg Hansen	2019-11-30 00:00:00
15	AE-2019-5196817	Greg Hansen	2019-12-31 00:00:00

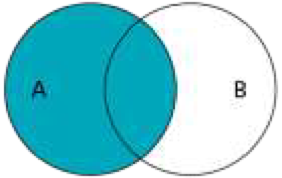
Partner Exercise:

JOINing Multiple Tables | **Solution**

Solution Query

```
SELECT
  o.order_id
  ,c.customer_name
  ,r.return_date
FROM orders o
        JOIN customers c ON o.customer_id = c.customer_id
        JOIN returns r ON o.order_id = r.order_id
LIMIT 100;
```

Left JOINs



LEFT JOIN loads all entries that appear in the first table, with NULLs where there is no match.

people

id	name	vehicle_id
1	Janet	c
2	Emily	d
3	Yoko	e

vehicles

id	vehicle_name
a	Explorer
b	Civic
c	Corolla
d	Impala

id	name	vehicle_id	vehicle_name
1	Janet	c	Corolla
2	Emily	d	Impala
3	Yoko	e	NULL

Guided Walk-Through:

Let’s Create a LEFT JOIN!

Let’s revisit the query we wrote earlier that JOINs the Orders and Returns tables. We want to find all orders and **return information if it exists**. How should we JOIN these two tables?

Orders		
order_id	**order_date**	**ship_date**
AE-2016-1308551	2016-09-28	2016-10-02
AE-2016-1522857	2016-09-04	2016-09-09

+

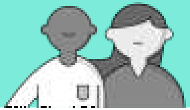
Returns	
order_id	**return_date**
AE-2019-1711936	2019-12-14
AE-2019-2092798	2019-11-29

Guided Walk-Through:

Creating a LEFT JOIN

Knowing that we want to keep all entries that appear in the Orders table, we'll add a LEFT JOIN that designates the Orders table as the first table. Here is our query:

```
SELECT
  o.order_id
  ,r.return_date
FROM orders o
    LEFT JOIN returns r ON o.order_id = r.order_id
LIMIT 100;
```



Partner Exercise:

How Do We JOIN This? |Challenge

10 minutes

Superstore is developing a training program to help salespeople reduce the likelihood of returns. To do so, Superstore wants to interview salespeople (each salesperson has a region) who've processed higher volumes of returns in the past. You're generating a list of salespeople and return reasons (including NULL returns!). With your partner, discuss what type of JOIN(s) will you use. Be ready to explain why.

Orders			Returns		Region	
order_id	**order_date**	**ship_date**	**order_id**	**return_date**	**country**	**region**
AE-2016-1308551	2016-09-28	2016-10-02	AE-2019-1711936	2019-12-14	Benin	EMEA
AE-2016-1522857	2016-09-04	2016-09-09	AE-2019-2092798	2019-11-29	Morocco	EMEA

© 2023 General Assembly

Partner Exercise:

How Do We JOIN This? |Solution

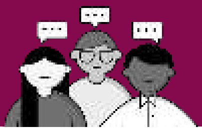
```
SELECT
  rg.salesperson
  ,r.reason_returned
  ,COUNT(o.order_id) AS count_of_returns
FROM orders o
JOIN regions rg ON o.region_id = rg.region_id
LEFT JOIN returns r ON o.order_id = r.order_id
GROUP BY 1, 2
ORDER BY 3 DESC
LIMIT 100;
```

This aggregate is run *after* the JOIN on the Returns and Regions tables is complete.

Recommended Practice for Faster Queries

- SELECT specific fields instead of using SELECT *.
- When testing JOINs, use LIMIT to control query sizes.
- Use IS NULL or IS NOT NULL to test for NULLs in a column.

UNIONs



Discussion:

Why UNIONs?

As we learned earlier, UNIONs combine rows from multiple tables with the same columns. In what scenarios will we use a UNION instead of a JOIN?

carpoolers		
id	name	vehicle_id
1	Janet	c
2	Emily	d
3	Yoko	d

monthly_parkers		
id	name	vehicle_id
2	Emily	d
4	Ali	e
5	Ray	a

→

id	name	vehicle_id
1	Janet	c
2	Emily	d
3	Yoko	d
4	Ali	e
5	Ray	a

UNION Syntax

Let’s look at some simple mock syntax for a **UNION**:

```
SELECT field1
    FROM table1
UNION
SELECT field1
    FROM table2
```

Guided Walk-Through:

Exploring Examples of UNIONs

A UNION takes a single column or collection of columns and “stacks” them on top of each other. A common use case is if we have similar data between two tables and want to UNION those two tables together.

For illustration purposes, we’ll be using the following sample HR tables:

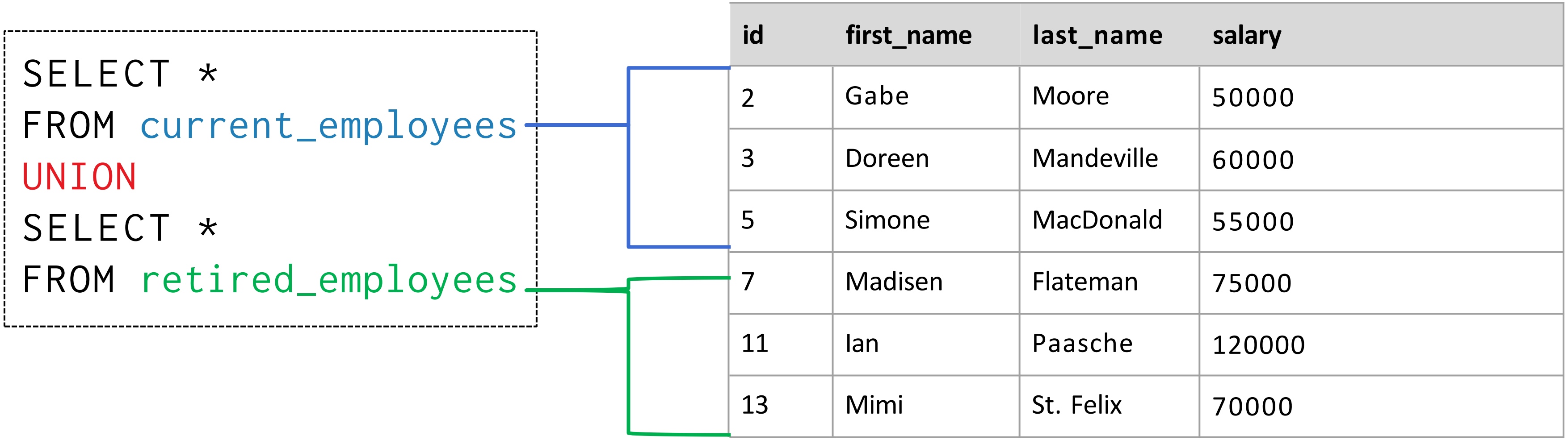
current_employees			
id	**first_name**	**last_name**	**salary**
2	Gabe	Moore	50000
3	Doreen	Mandeville	60000
5	Simone	MacDonald	55000

retired_employees			
id	**first_name**	**last_name**	**salary**
7	Madisen	Flateman	75000
11	Ian	Paasche	120000
13	Mimi	St. Felix	70000

Guided Walk-Through:

Creating a UNION for Two Tables

When you want to combine the two tables, and both tables have the same columns, you can use a UNION with a SELECT *:



```
SELECT *
FROM current_employees
UNION
SELECT *
FROM retired_employees
```

id	first_name	last_name	salary
2	Gabe	Moore	50000
3	Doreen	Mandeville	60000
5	Simone	MacDonald	55000
7	Madisen	Flateman	75000
11	Ian	Paasche	120000
13	Mimi	St. Felix	70000

Guided Walk-Through:

Creating a UNION for Two Tables (Cont.)

You can also UNION tables on only columns. These columns must match data types but do not have to represent the same data. What happened in the table below? And where do the resulting headers come from?

```
SELECT first_name,
last_name
FROM current_employees
UNION
SELECT last_name,
first_name
FROM retired_employees
```

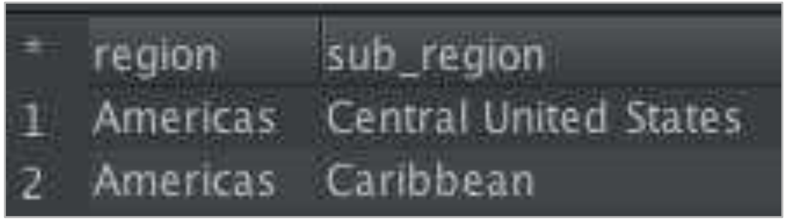
first_name	last_name
Gabe	Moore
Doreen	Mandeville
Simone	MacDonald
Flateman	Madisen
Paasche	Ian
St. Felix	Mimi

Guided Walk-Through:

Creating a UNION

UNIONs can help organize tables into logical groups, making your SQL code more reusable and easier to debug. Let’s see how this works by applying UNION to the regions table to combine region and sub-regions.

```
SELECT region, sub_region
FROM regions
WHERE sub_region =
'Central United States'
UNION
SELECT region,
sub_region FROM
regions
WHERE sub_region = 'Caribbean'
```



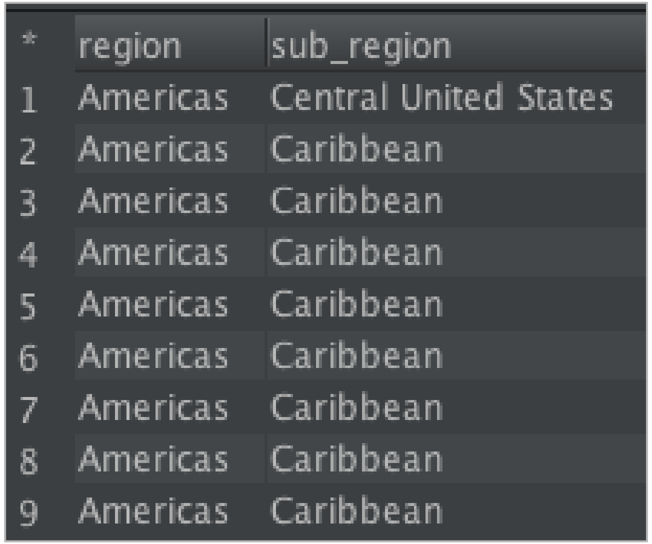
	region	sub_region
1	Americas	Central United States
2	Americas	Caribbean

Guided Walk-Through:

Creating a UNION ALL

Let’s rework the same example with a UNION ALL. What changed?

```
SELECT region, sub_region
FROM regions
WHERE sub_region =
'Central United States'
UNION ALL
SELECT region,
sub_region FROM
regions
WHERE sub_region = 'Caribbean'
```



*	region	sub_region
1	Americas	Central United States
2	Americas	Caribbean
3	Americas	Caribbean
4	Americas	Caribbean
5	Americas	Caribbean
6	Americas	Caribbean
7	Americas	Caribbean
8	Americas	Caribbean
9	Americas	Caribbean

Discussion:

UNION ALL

We know that UNIONs remove duplicates, whereas UNION ALL allows duplicates. Looking at the UNION ALL syntax for Superstore, what are some of the reasons why we’d want to keep duplicate values?

```
SELECT region, sub_region
FROM regions
WHERE sub_region =
'Central United States'
UNION ALL
SELECT region,
sub_region FROM
regions
WHERE sub_region = 'Caribbean'
```

Rules for Using UNIONs

Remember these rules when using UNIONs:

- You *must* match the number of columns, and they *must* be of compatible data types.
- You can only have one **ORDER BY** at the bottom of your full SELECT statement.
- UNION removes *composite* duplicates.
- UNION ALL allows duplicates.

Combining Data With JOINs and UNIONs

Wrapping Up

Recap

Today, we worked on...

- Combining data from multiple sources using JOINs and UNIONs.

Looking Ahead

Up Next:

Subqueries

Additional Resources

- Microsoft reference material on UNIONs: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-union-transact-sql
- INNER JOIN tutorial: http://www.sqltutorial.org/sql-inner-join/
- Common table expressions (where UNIONS are used frequently) — this is a more advanced concept, out of the scope of this course.
- Differences between Normalization and Denormalization