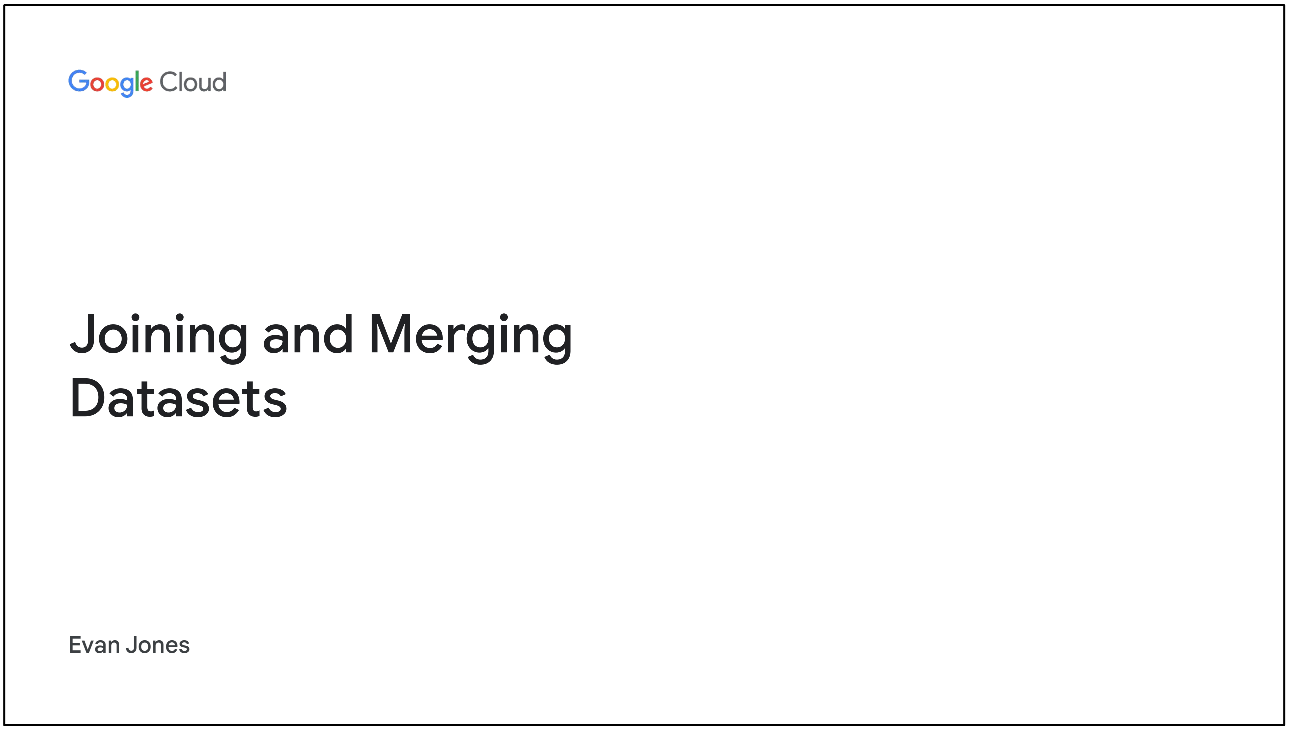
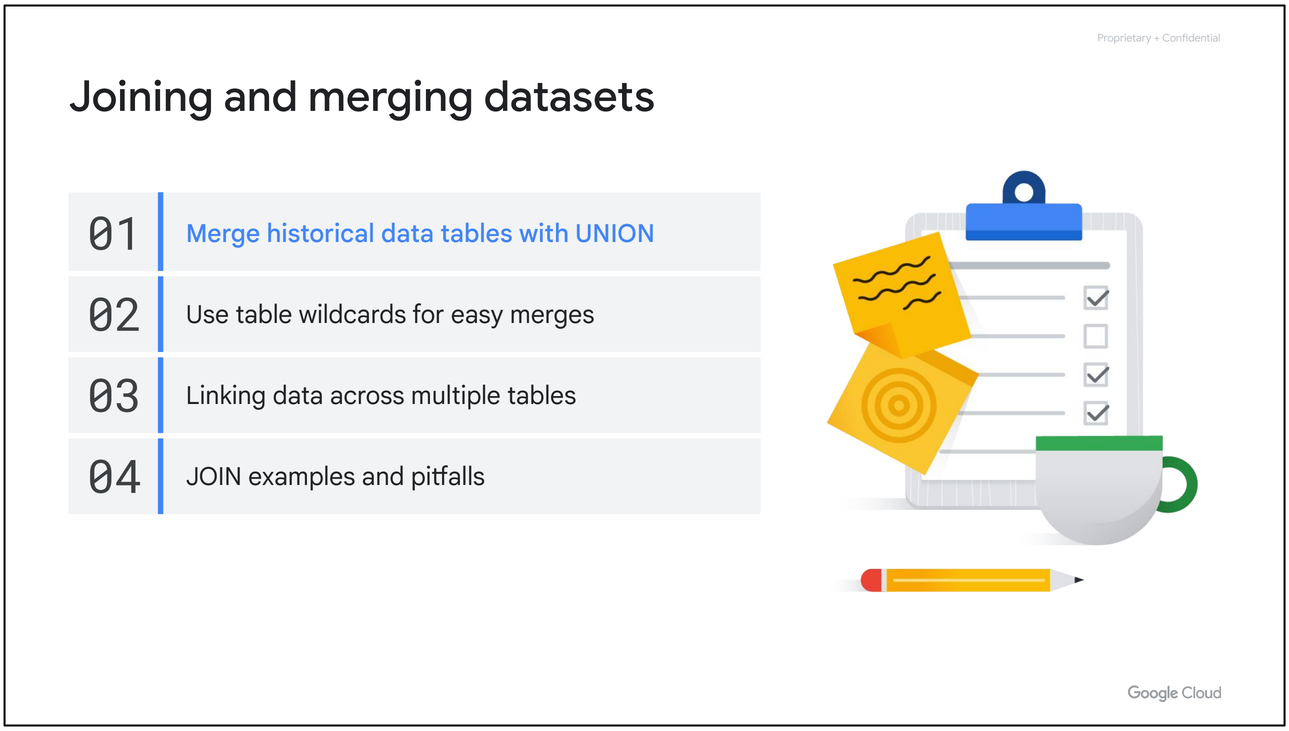


Google Cloud

Joining and Merging Datasets

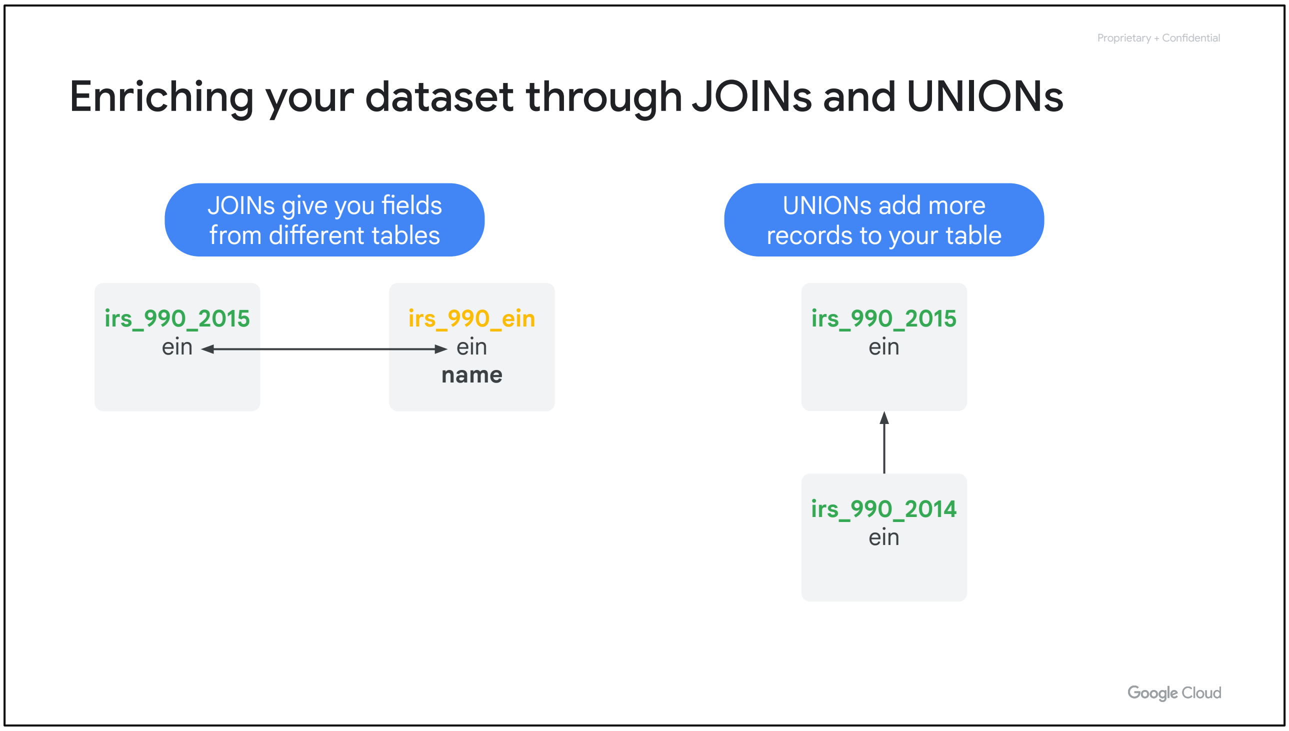
Evan Jones

Now one of the most popular topics in SQL has had a mash-up multiple data sources together in a single query to answer more complex insights.



Joining and merging datasets

Now in this module, we'll tackle how to append additional historical datasets together through UNIONs, as well as how to JOIN together different datasets horizontally through SQL JOINs. Let's walk through the basics, know how to highlight some key pitfalls along the way.



Proprietary + Confidential

Enriching your dataset through JOINs and UNIONs

JOINs give you fields from different tables
irs_990_2015
ein
irs_990_ein
ein
name
UNIONs add more records to your table
irs_990_2015
ein
irs_990_2014
ein
Google Cloud

All right, so now let's move into one of my most favorite topics to talk about which is enriching your dataset through the use of JOINs and UNIONs. And if you are a SQL guru, there's a couple of different tricks and tips that you can use inside of BigQuery that I think you'll like. But let's go over some of the basics.

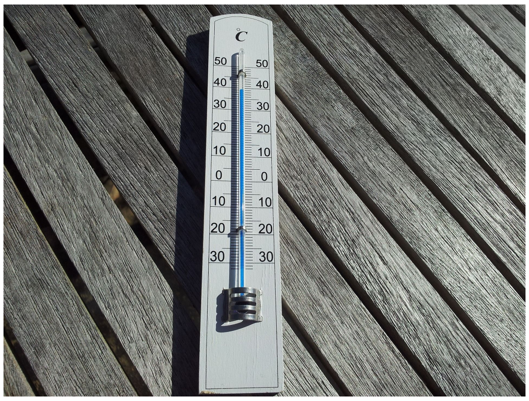
A JOIN for example, if you have the tables for the IRS, the annual tax history for 2015, but you want to get the name of the charity, now that's in two different tables. You actually have to link those two tables together, in SQL we call that a JOIN. And you do that on a common identifying field that's shared between the two tables. And there's a lot of caveats when it comes to whether or not those fields are duplicative or which type of JOIN that you're going to use, I want to get into all that nuance as part of this module.

Now a little bit simpler than that is a UNION, which on the right if you have historical data that shares the same schema, you can actually append records or mash them together vertically, I like to say, through what's called the SQL UNION.

So combining both of these two concepts together, you could actually bring together the annual tax filing history for 2012 or how long the dataset goes for, 2012-2013, 2014-2015, and so on, and get an enriching information about the organizational details that's not present in the individual filings by using a JOIN. So let's explore these concepts a little bit more in depth with a walkthrough example.

Walkthrough example

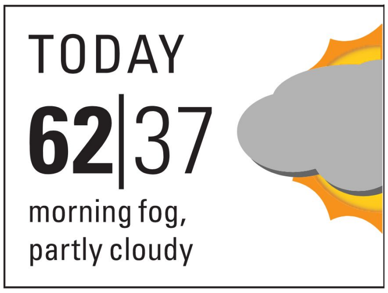
Joining and merging *temperature* and *weather station* data



So the example that we're going to use to illustrate JOINs in merging of data is going to be temperature and weather station data.

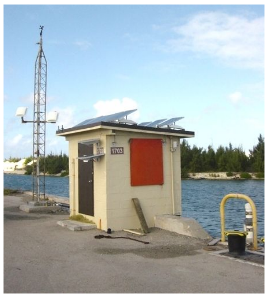
Two types of tables in the NOAA weather dataset

Daily temperature readings



Weather recording station locations

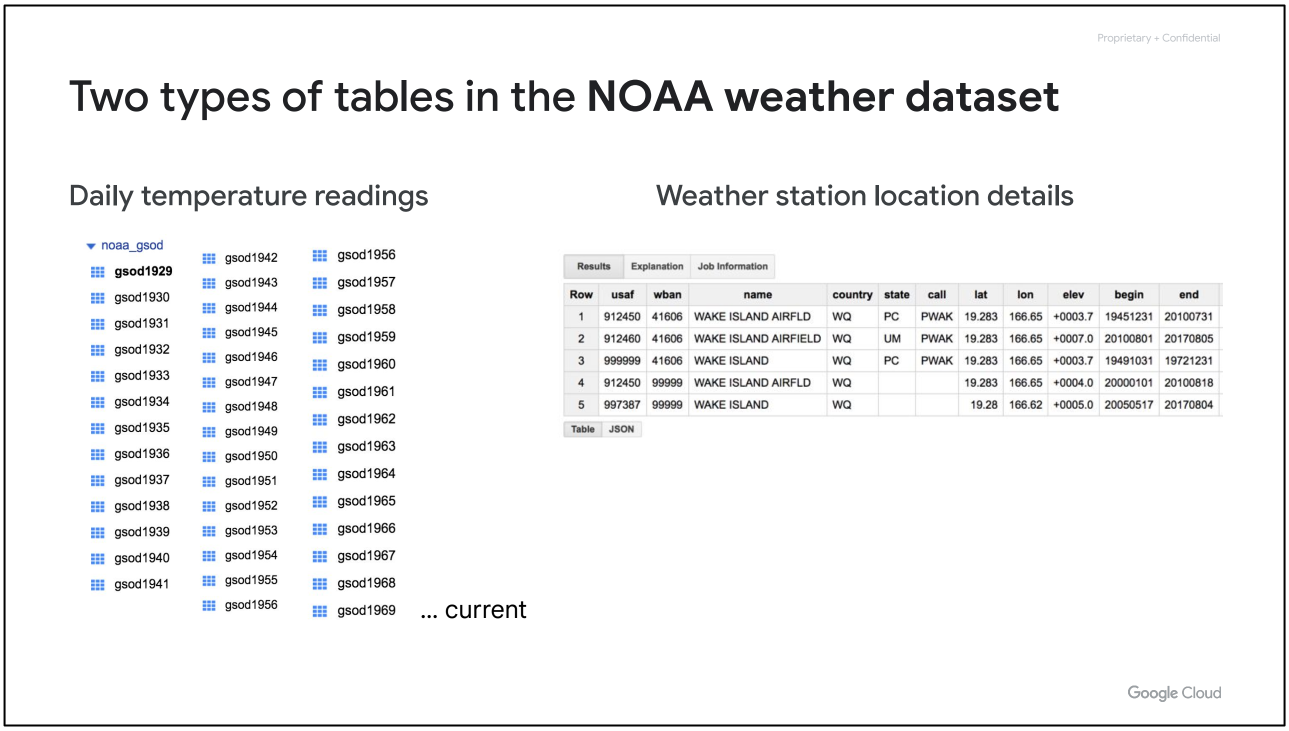
Victoria, Australia


Wake Island Harbor

So the example that we're going to be using for this walkthrough is the NOAA Weather Dataset. NOAA is a research agency in the US which tracks the patterns of weather and does a bunch of other cool things with meteorological studies, and as such they have a lot of really, really good weather data.

So we have temperature recordings not just for today, but all the way back into the 1920s, for a lot of other different weather stations all around the world.

So we have these different temperature readings, and you have the locations of the weather stations that they belong to. And as you see with that dividing line, those two pieces of data are in two separate tables that we're gonna have to JOIN together.



Two types of tables in the NOAA weather dataset

Daily temperature readings

gsod1929
gsod1930
gsod1931
gsod1932
gsod1933
gsod1934
gsod1935
gsod1936
gsod1937
gsod1938
gsod1939
gsod1940
gsod1941
gsod1942
gsod1943
gsod1944
gsod1945
gsod1946
gsod1947
gsod1948
gsod1949
gsod1950
gsod1951
gsod1952
gsod1953
gsod1954
gsod1955
gsod1956
gsod1956
gsod1957
gsod1958
gsod1959
gsod1960
gsod1961
gsod1962
gsod1963
gsod1964
gsod1965
gsod1966
gsod1967
gsod1968
gsod1969

… current

Weather station location details

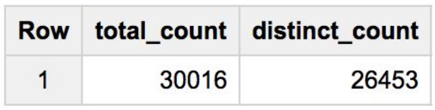
Row	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
1	912450	41606	WAKE ISLAND AIRFLD	WQ	PC	PWAK	19.283	166.65	+0003.7	19451231	20100731
2	912460	41606	WAKE ISLAND AIRFIELD	WQ	UM	PWAK	19.283	166.65	+0007.0	20100801	20170805
3	999999	41606	WAKE ISLAND	WQ	PC	PWAK	19.283	166.65	+0003.7	19491031	19721231
4	912450	99999	WAKE ISLAND AIRFLD	WQ			19.283	166.65	+0004.0	20000101	20100818
5	997387	99999	WAKE ISLAND	WQ			19.28	166.62	+0005.0	20050517	20170804

So it's many more tables than just two. So for those daily temperature readings, it actually goes back to 1929. So, there's an individual table for the annual readings from 1929, and I believe this is hourly, every hour. So you're talking millions and millions of records all the way to the current day.

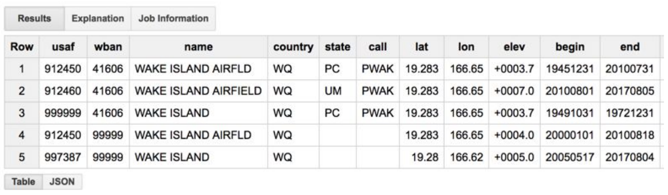
And then on the right hand side, you have a lookup table that has the information about the stations that took those readings. So this is like Wake Island airfield. The latitude and longitude and a bunch of other lookup information that we don't want to repeatedly store in those daily temperature reading tables that we're going to use something like a JOIN to enrich those two together. So let's talk a little bit more about the nuances of them.

What is our **unique identifier** for a weather station?

```
#standardSQL
# Is usaf unique over time?
  SELECT
    COUNT(usaf) AS total_count,
    COUNT(DISTINCT usaf) AS distinct_count
FROM
`bigquery-public-data.noaa_gsod.stations`;
```

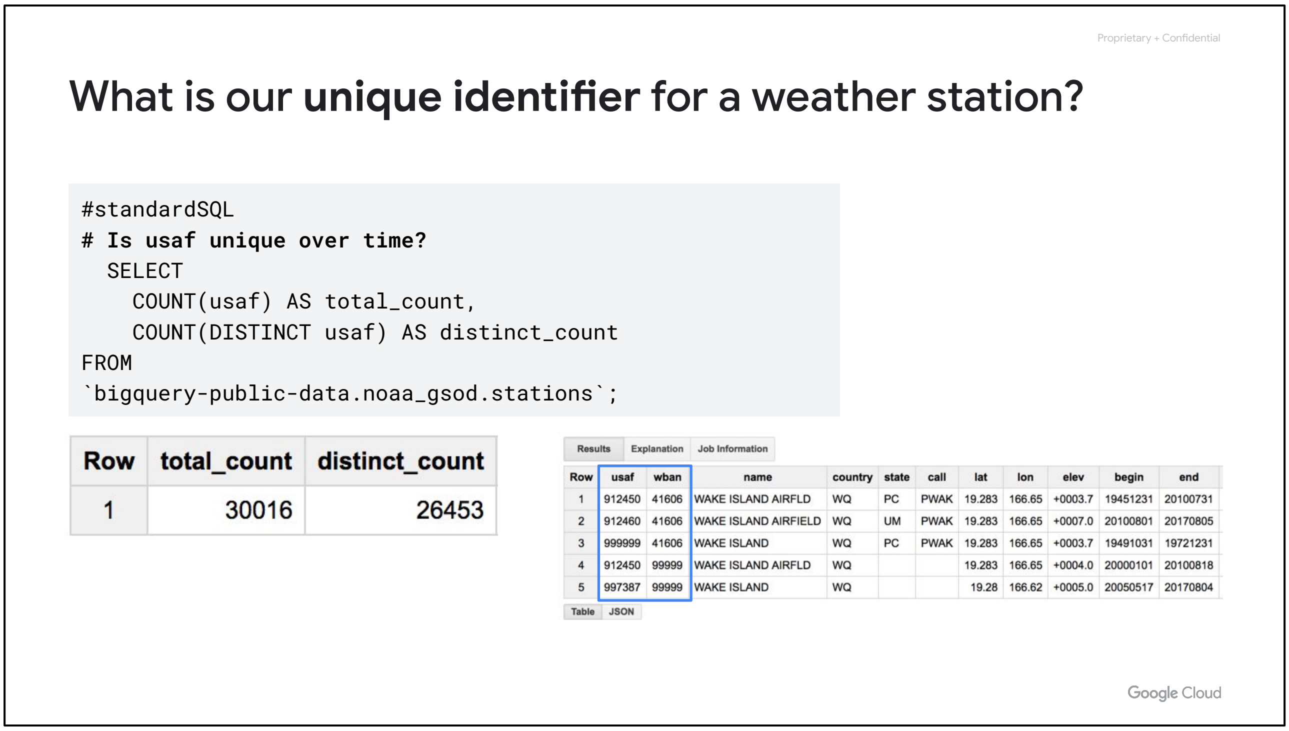


Row	total_count	distinct_count
1	30016	26453



Row	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
1	912450	41606	WAKE ISLAND AIRFLD	WQ	PC	PWAK	19.283	166.65	+0003.7	19451231	20100731
2	912460	41606	WAKE ISLAND AIRFIELD	WQ	UM	PWAK	19.283	166.65	+0007.0	20100801	20170805
3	999999	41606	WAKE ISLAND	WQ	PC	PWAK	19.283	166.65	+0003.7	19491031	19721231
4	912450	99999	WAKE ISLAND AIRFLD	WQ			19.283	166.65	+0004.0	20000101	20100818
5	997387	99999	WAKE ISLAND	WQ			19.28	166.62	+0005.0	20050517	20170804

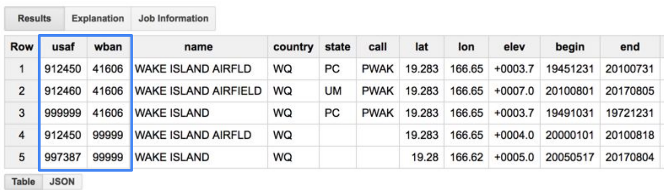
So, we need to find a unique identifier for our weather stations before we can even hope to JOIN them against another table. So when you inherit a data set, this is honestly one of the most, the trickiest things to uncover, is how do you uniquely identify a row in a dataset that you've been given? So there's two fields that look like they're identifiers.



What is our **unique identifier** for a weather station?

```
#standardSQL
# Is usaf unique over time?
  SELECT
    COUNT(usaf) AS total_count,
    COUNT(DISTINCT usaf) AS distinct_count
FROM
`bigquery-public-data.noaa_gsod.stations`;
```

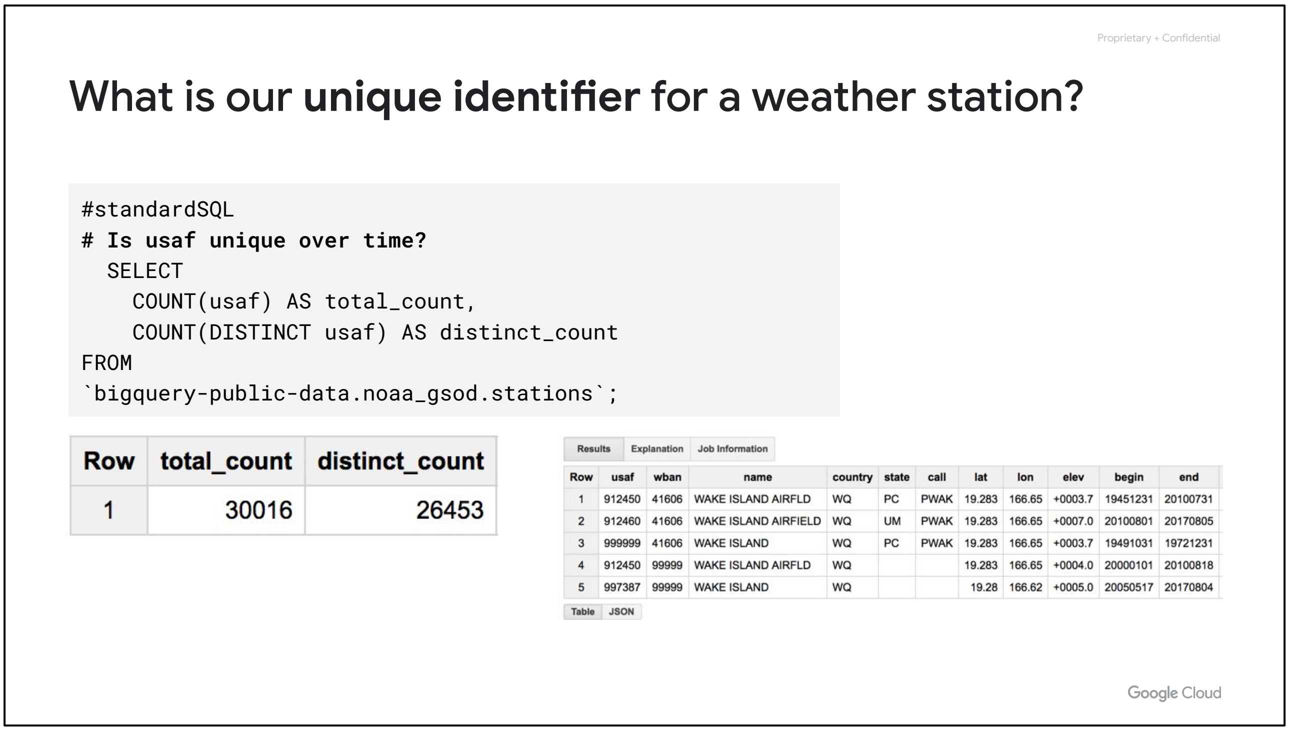
Row	total_count	distinct_count
1	30016	26453



Row	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
1	912450	41606	WAKE ISLAND AIRFLD	WQ	PC	PWAK	19.283	166.65	+0003.7	19451231	20100731
2	912460	41606	WAKE ISLAND AIRFIELD	WQ	UM	PWAK	19.283	166.65	+0007.0	20100801	20170805
3	999999	41606	WAKE ISLAND	WQ	PC	PWAK	19.283	166.65	+0003.7	19491031	19721231
4	912450	99999	WAKE ISLAND AIRFLD	WQ			19.283	166.65	+0004.0	20000101	20100818
5	997387	99999	WAKE ISLAND	WQ			19.28	166.62	+0005.0	20050517	20170804

Google Cloud

The USAF or the United States Air Force weather identifier there, USAF, and then you have the WBAN, the WBAN there.



What is our **unique identifier** for a weather station?

```
#standardSQL
# Is usaf unique over time?
  SELECT
    COUNT(usaf) AS total_count,
    COUNT(DISTINCT usaf) AS distinct_count
FROM
`bigquery-public-data.noaa_gsod.stations`;
```

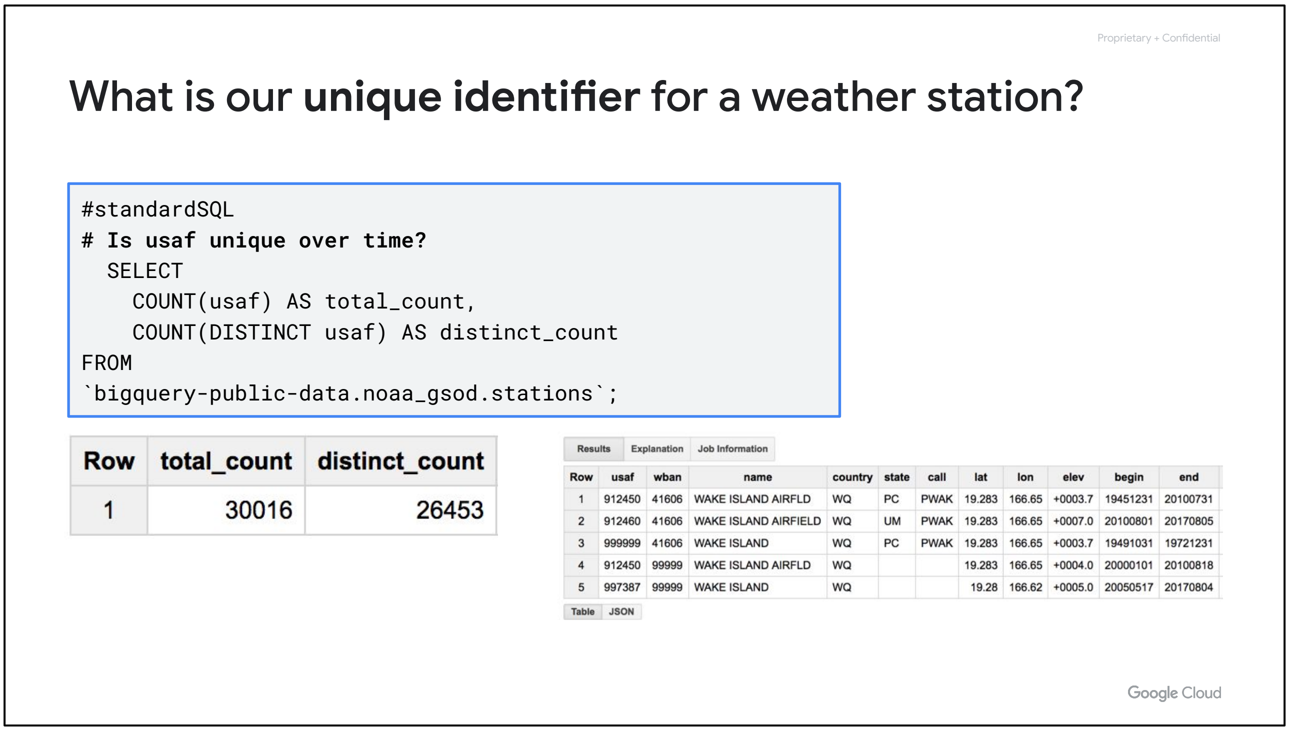
Row	total_count	distinct_count
1	30016	26453

Results
Explanation
Job Information

Row	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
1	912450	41606	WAKE ISLAND AIRFLD	WQ	PC	PWAK	19.283	166.65	+0003.7	19451231	20100731
2	912460	41606	WAKE ISLAND AIRFIELD	WQ	UM	PWAK	19.283	166.65	+0007.0	20100801	20170805
3	999999	41606	WAKE ISLAND	WQ	PC	PWAK	19.283	166.65	+0003.7	19491031	19721231
4	912450	99999	WAKE ISLAND AIRFLD	WQ			19.283	166.65	+0004.0	20000101	20100818
5	997387	99999	WAKE ISLAND	WQ			19.28	166.62	+0005.0	20050517	20170804

Google Cloud

So, what you likely saw in the first lab as part of the first course, is introduce you to counting duplicative or distinct records.



What is our **unique identifier** for a weather station?

```
#standardSQL
# Is usaf unique over time?
  SELECT
    COUNT(usaf) AS total_count,
    COUNT(DISTINCT usaf) AS distinct_count
FROM
`bigquery-public-data.noaa_gsod.stations`;
```

Row	total_count	distinct_count
1	30016	26453

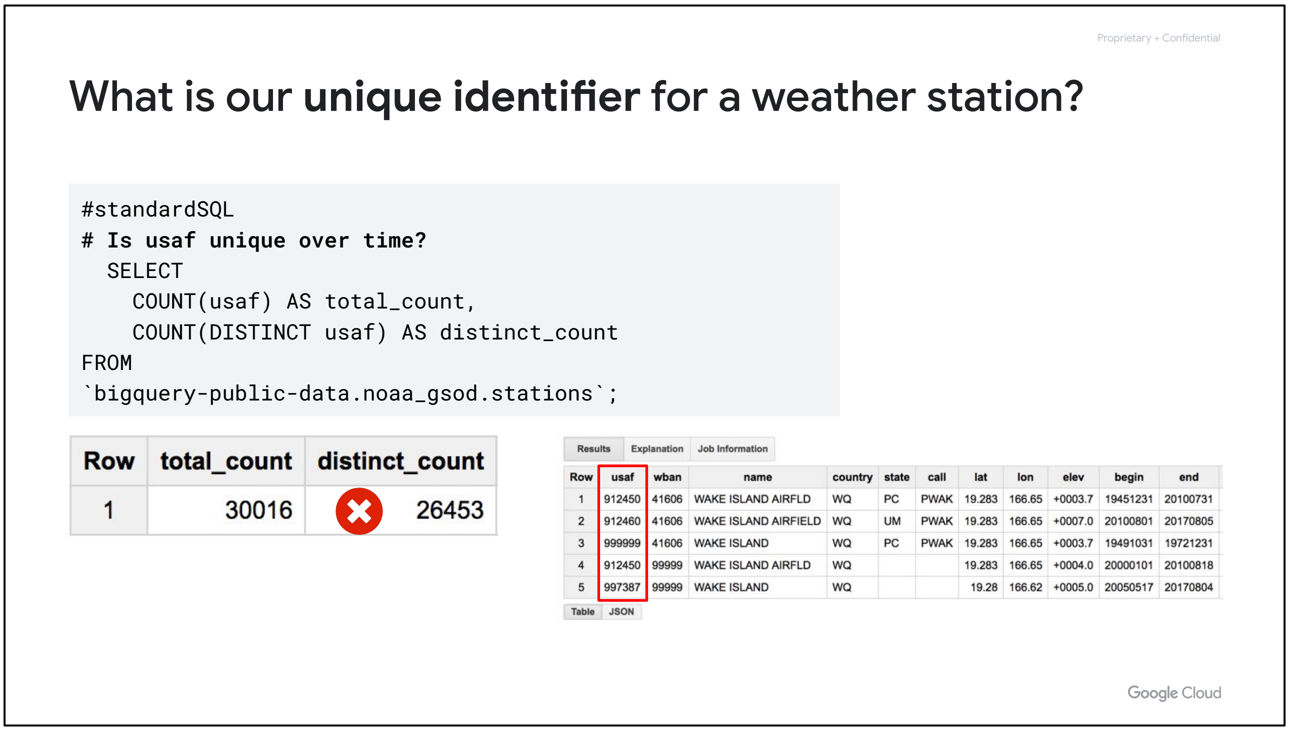
Results | Explanation | Job Information

Row	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
1	912450	41606	WAKE ISLAND AIRFLD	WQ	PC	PWAK	19.283	166.65	+0003.7	19451231	20100731
2	912460	41606	WAKE ISLAND AIRFIELD	WQ	UM	PWAK	19.283	166.65	+0007.0	20100801	20170805
3	999999	41606	WAKE ISLAND	WQ	PC	PWAK	19.283	166.65	+0003.7	19491031	19721231
4	912450	99999	WAKE ISLAND AIRFLD	WQ			19.283	166.65	+0004.0	20000101	20100818
5	997387	99999	WAKE ISLAND	WQ			19.28	166.62	+0005.0	20050517	20170804

Table | JSON

Google Cloud

We use two simple count functions as you see there on the SQL code. The regular COUNT and the COUNT DISTINCT. That's a quick way to see whether or not those records are the same, meaning that it is unique for those row counts.



What is our **unique identifier** for a weather station?

```
#standardSQL
# Is usaf unique over time?
  SELECT
    COUNT(usaf) AS total_count,
    COUNT(DISTINCT usaf) AS distinct_count
FROM
`bigquery-public-data.noaa_gsod.stations`;
```

Row	total_count	distinct_count
1	30016	❌ 26453

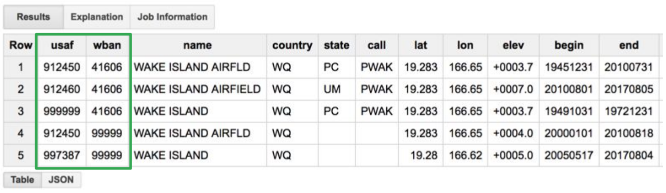
Row	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
1	912450	41606	WAKE ISLAND AIRFLD	WQ	PC	PWAK	19.283	166.65	+0003.7	19451231	20100731
2	912460	41606	WAKE ISLAND AIRFIELD	WQ	UM	PWAK	19.283	166.65	+0007.0	20100801	20170805
3	999999	41606	WAKE ISLAND	WQ	PC	PWAK	19.283	166.65	+0003.7	19491031	19721231
4	912450	99999	WAKE ISLAND AIRFLD	WQ			19.283	166.65	+0004.0	20000101	20100818
5	997387	99999	WAKE ISLAND	WQ			19.28	166.62	+0005.0	20050517	20170804

And as you see here with the red exes they are not. So, you have an interesting dilemma. If you don't have any one field that can uniquely identify a row, what could you potentially do? So two general options.

We need to use a **combination key**

```
#standardSQL
# Is usaf wban combo unique over time?
  SELECT
    COUNT(CONCAT(usaf,wban)) AS total_stations,
    COUNT(DISTINCT CONCAT(usaf,wban)) AS distinct_stations
FROM
`bigquery-public-data.noaa_gsod.stations`;
```

Row	total_stations	distinct_stations
1	✓ 30016	✓ 30016


Results | Explanation | Job Information

Row	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
1	912450	41606	WAKE ISLAND AIRFLD	WQ	PC	PWAK	19.283	166.65	+0003.7	19451231	20100731
2	912460	41606	WAKE ISLAND AIRFIELD	WQ	UM	PWAK	19.283	166.65	+0007.0	20100801	20170805
3	999999	41606	WAKE ISLAND	WQ	PC	PWAK	19.283	166.65	+0003.7	19491031	19721231
4	912450	99999	WAKE ISLAND AIRFLD	WQ			19.283	166.65	+0004.0	20000101	20100818
5	997387	99999	WAKE ISLAND	WQ			19.28	166.62	+0005.0	20050517	20170804

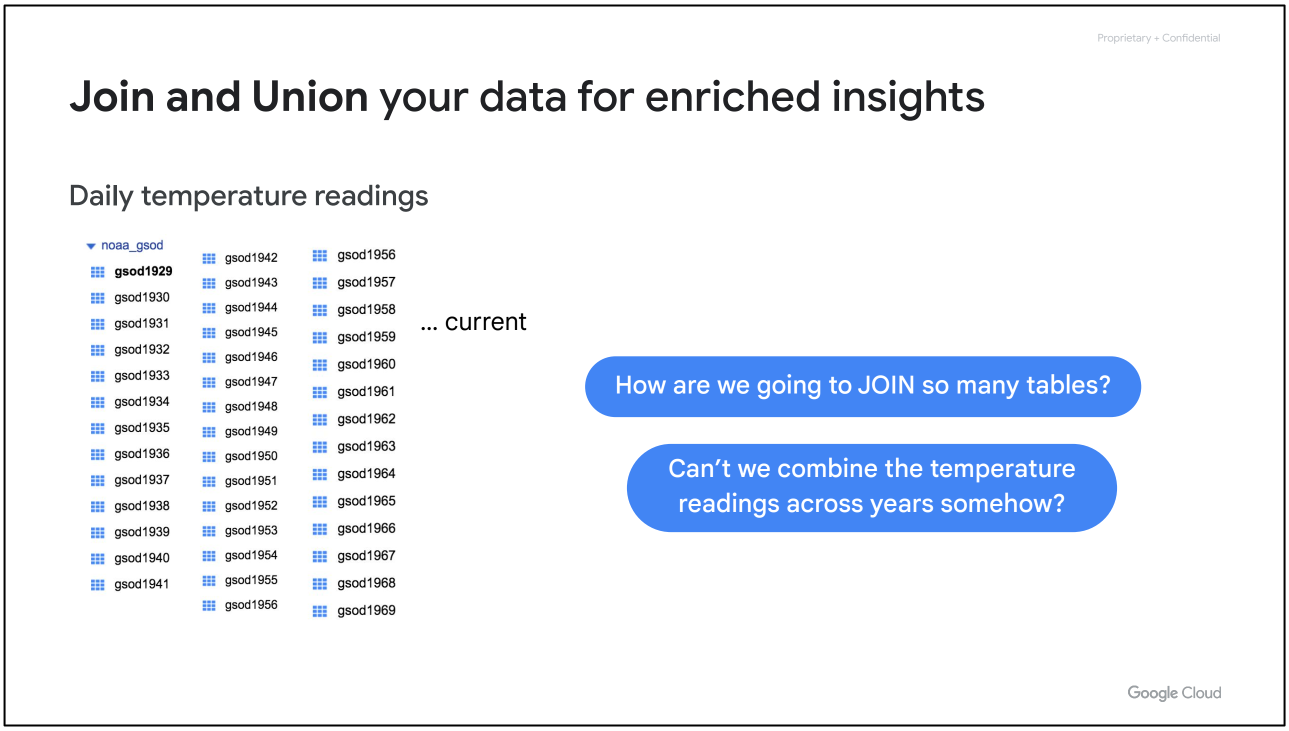
Table | JSON

Google Cloud

One, you could create your own primary key, you could insert an arbitrary row number and have that be unique identifier for that record set. But if we're going to have any hope of joining it against a different dataset, maybe we'll have to make do with what we have.

And one of the things that you can do is create a combination key, where the combination of these two identifiers, the USAF field and the WBAN field, together kind of smashing together through that concatenation or that merging is in itself a unique identifier.

So you can see for that first record. So for the 912450, you can see row one and row four are duplicative for the usaf, but if you merge them together with the WBAN, even though the WBAN has 41606 repeated three times, the combination of both of those fields together uniquely identifies them.



Join and Union your data for enriched insights

Daily temperature readings

noaa_gsod

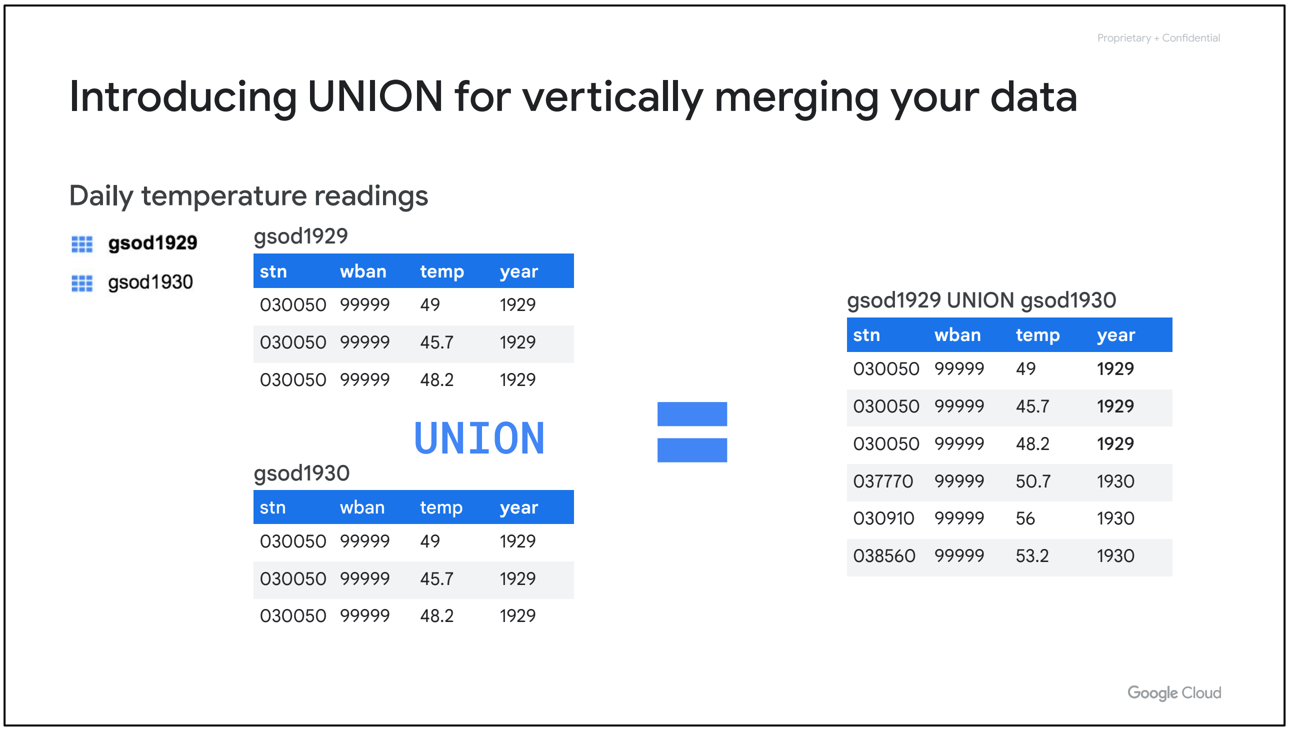
gsod1929
gsod1930
gsod1931
gsod1932
gsod1933
gsod1934
gsod1935
gsod1936
gsod1937
gsod1938
gsod1939
gsod1940
gsod1941
gsod1942
gsod1943
gsod1944
gsod1945
gsod1946
gsod1947
gsod1948
gsod1949
gsod1950
gsod1951
gsod1952
gsod1953
gsod1954
gsod1955
gsod1956
gsod1956
gsod1957
gsod1958
gsod1959
gsod1960
gsod1961
gsod1962
gsod1963
gsod1964
gsod1965
gsod1966
gsod1967
gsod1968
gsod1969

… current

How are we going to JOIN so many tables?

Can’t we combine the temperature readings across years somehow?

So, how are we actually going to JOIN that combination key on all of these different tables? We need a somehow UNION potentially bring all these data tables historically together, mash them vertically, but that might be a really long effort.



Introducing UNION for vertically merging your data

Daily temperature readings

- **gsod1929**
- gsod1930

gsod1929

stn	wban	temp	year
030050	99999	49	1929
030050	99999	45.7	1929
030050	99999	48.2	1929

UNION

gsod1930

stn	wban	temp	year
030050	99999	49	1929
030050	99999	45.7	1929
030050	99999	48.2	1929

=

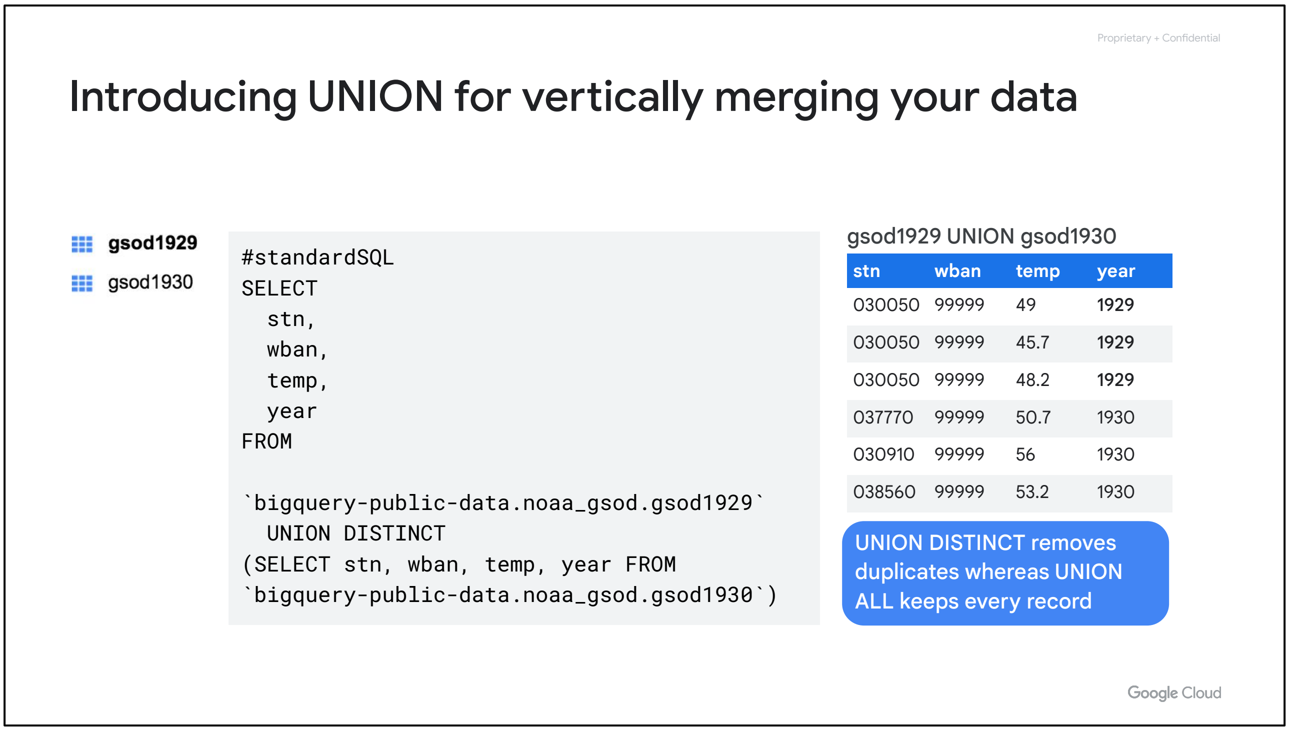
gsod1929 UNION gsod1930

stn	wban	temp	year
030050	99999	49	**1929**
030050	99999	45.7	**1929**
030050	99999	48.2	**1929**
037770	99999	50.7	1930
030910	99999	56	1930
038560	99999	53.2	1930

Google Cloud

So let's cover exactly what a UNION is first before we talk about the syntax of how to do it.

So UNION in brief, if you have tables with the same schema or at least from the fields that you're selecting across the tables are the same, what you can then do is apply to a UNION and it mashes those two tables together vertically. So if you have records from the year 1929 and then 1930 into two tables, as you can see the result they're added vertically, is going to give you a consolidated table with records from both 1929 and 1930.



Introducing UNION for vertically merging your data

gsod1929
gsod1930

```
#standardSQL
SELECT
  stn,
  wban,
  temp,
  year
FROM

`bigquery-public-data.noaa_gsod.gsod1929`
  UNION DISTINCT
(SELECT stn, wban, temp, year FROM
`bigquery-public-data.noaa_gsod.gsod1930`)
```

gsod1929 UNION gsod1930

stn	wban	temp	year
030050	99999	49	**1929**
030050	99999	45.7	**1929**
030050	99999	48.2	**1929**
037770	99999	50.7	1930
030910	99999	56	1930
038560	99999	53.2	1930

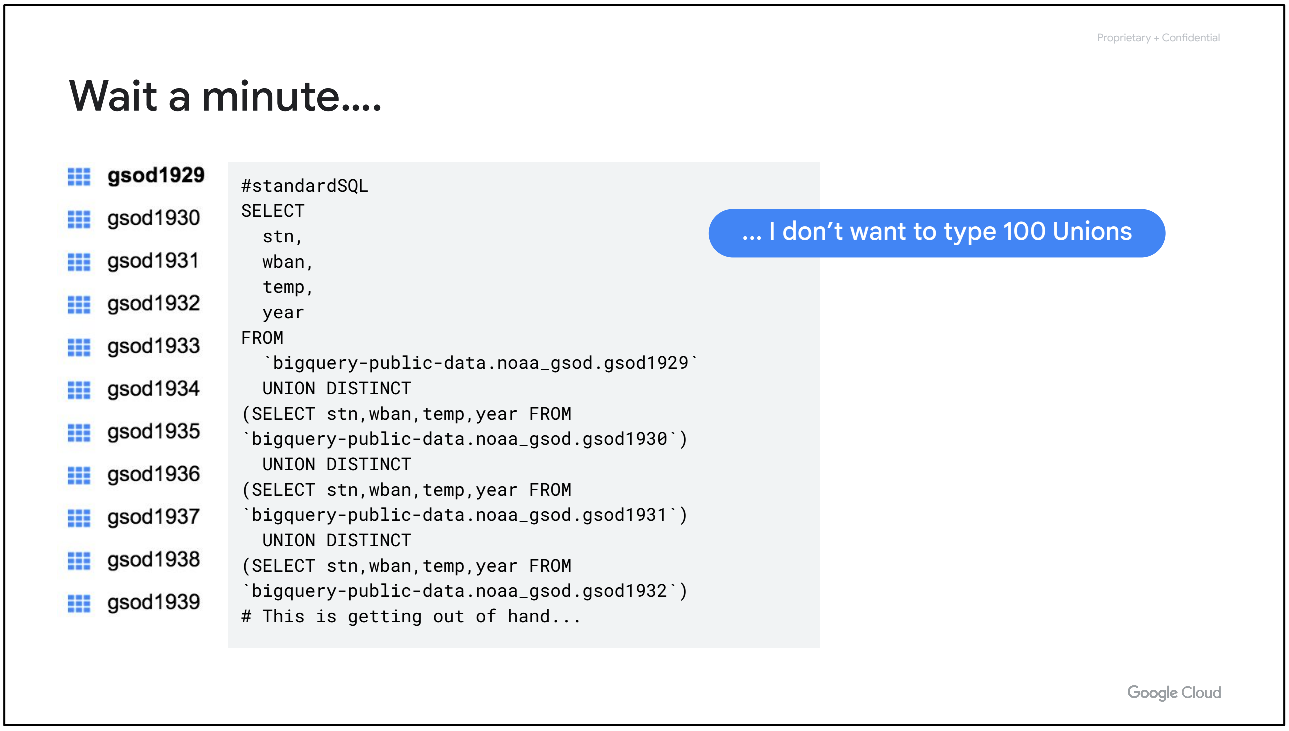
UNION DISTINCT removes duplicates whereas UNION ALL keeps every record

So let's go into a little bit more of nuance. So there's two different types of UNIONs. You can have a DISTINCT UNION or you can have what's called a UNION ALL.

So if you just wanted to blindly mash together all the records without worrying about whether or not one table had a duplicative record from another table, you'd be using the UNION ALL.

So now say if you wanted to actually be mindful of duplicates across both the different tables, you would be using a UNION DISTINCT.

Now here we shouldn't see repetitive temperatures across different tables since those temperatures are fenced in by the year of that particular table. But if you just want to be extra cautious, you could do that UNION DISTINCT.



Proprietary + Confidential

Wait a minute....

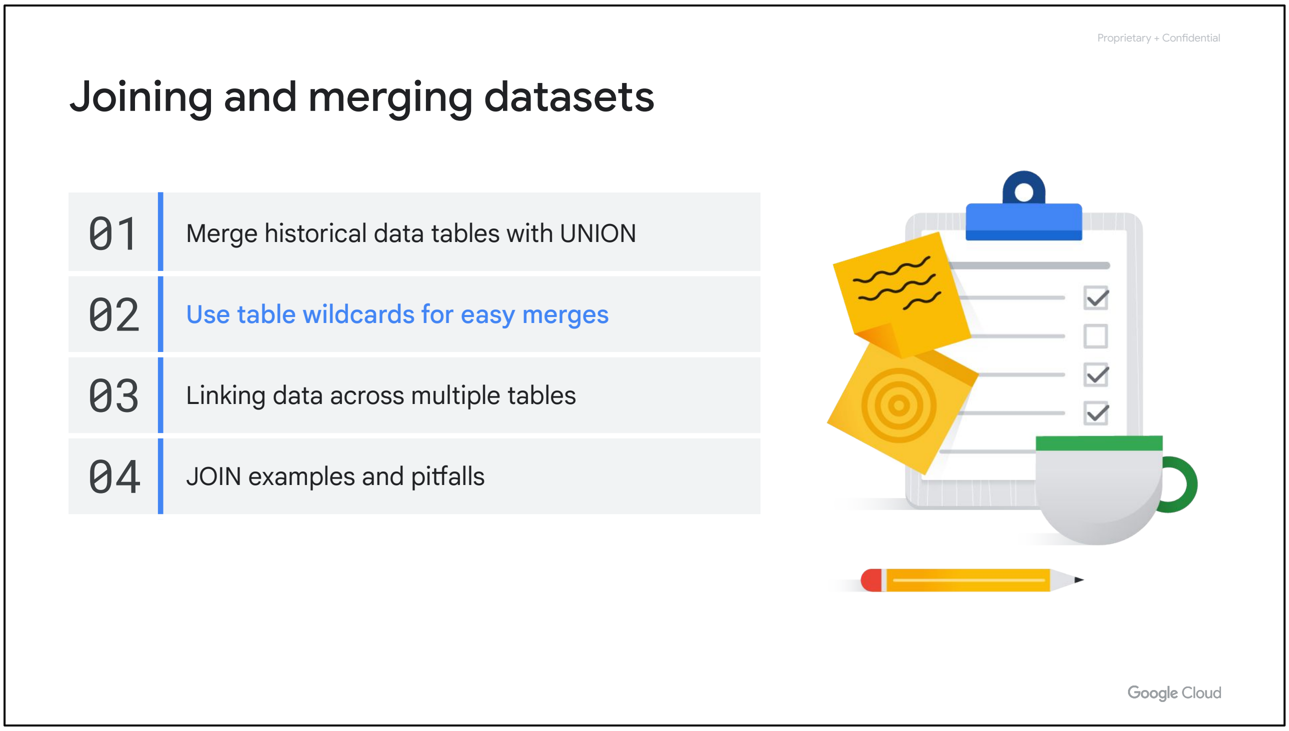
- gsod1929
- gsod1930
- gsod1931
- gsod1932
- gsod1933
- gsod1934
- gsod1935
- gsod1936
- gsod1937
- gsod1938
- gsod1939

```
#standardSQL
SELECT
  stn,
  wban,
  temp,
  year
FROM
  `bigquery-public-data.noaa_gsod.gsod1929`
  UNION DISTINCT
(SELECT stn,wban,temp,year FROM
`bigquery-public-data.noaa_gsod.gsod1930`)
  UNION DISTINCT
(SELECT stn,wban,temp,year FROM
`bigquery-public-data.noaa_gsod.gsod1931`)
  UNION DISTINCT
(SELECT stn,wban,temp,year FROM
`bigquery-public-data.noaa_gsod.gsod1932`)
# This is getting out of hand...
```

... I don’t want to type 100 Unions

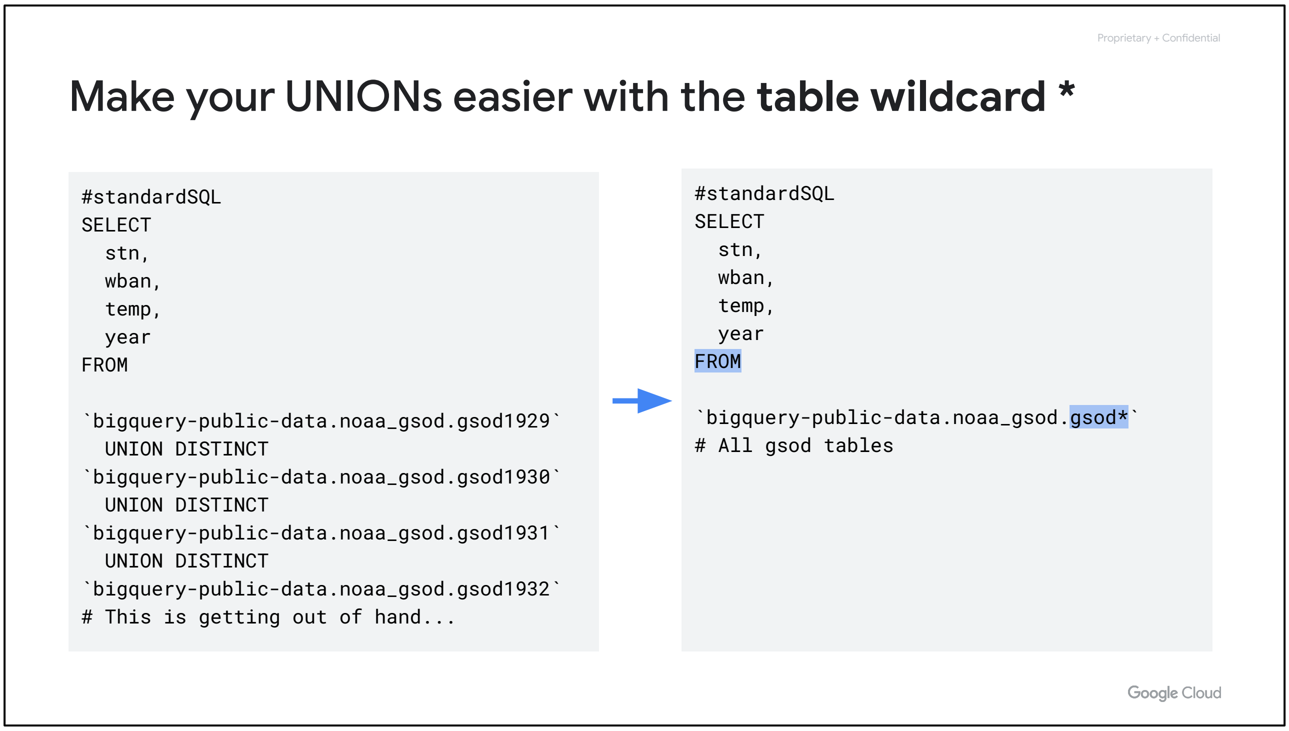
Google Cloud

So here's the dilemma, you have quite a few tables that you need to bring together. This is the syntax for the UNION. Select all the fields that we want from one of these particular tables, UNION DISTINCT just write it out. And then you just do another table, and UNION DISTINCT and you do another table, and UNION DISTINCT and you do another table. And if you have more than 10 tables, your fingers will get tired of typing all these tables over time. So, I don't really want to type on UNIONs, it's going to make my code just extremely long for the query to read. So there must be a better way.



Joining and merging datasets

And, now we introduce the concept of a table wildcard. This is pretty cool.



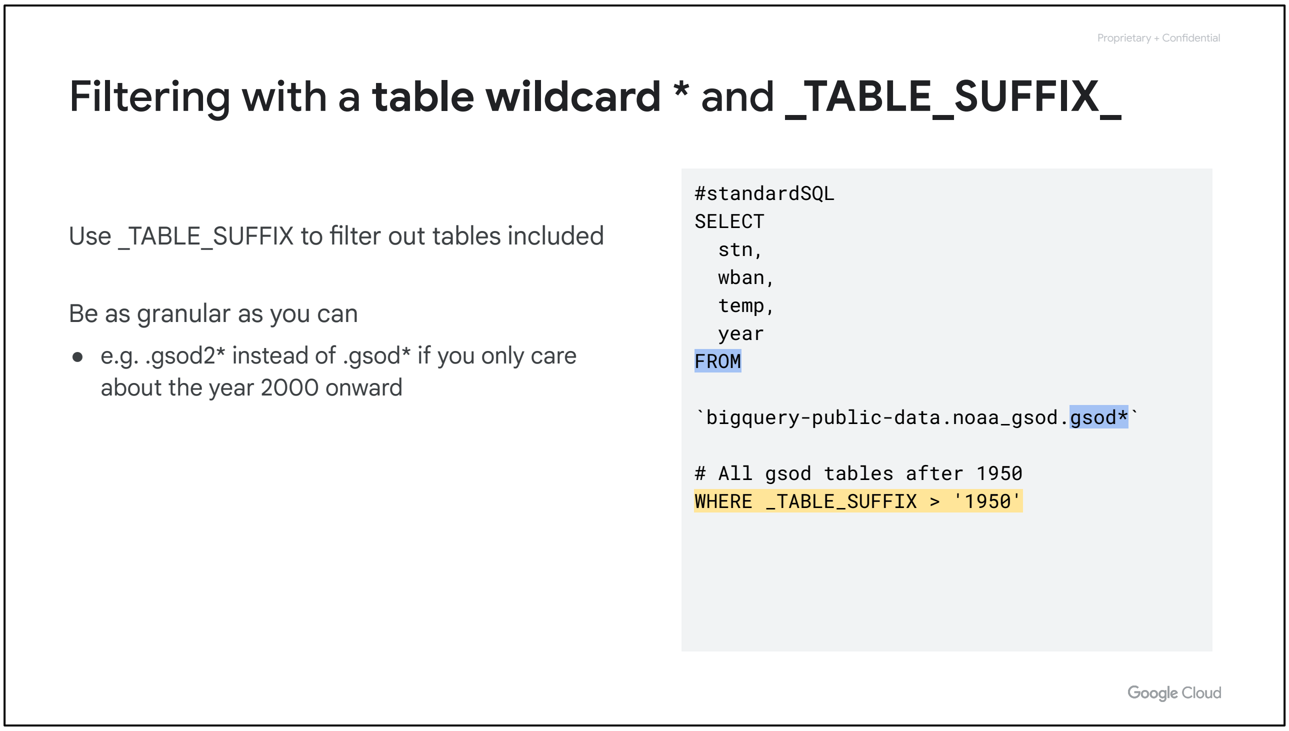
Make your UNIONs easier with the **table wildcard** *

```
#standardSQL
SELECT
  stn,
  wban,
  temp,
  year
FROM

`bigquery-public-data.noaa_gsod.gsod1929`
  UNION DISTINCT
`bigquery-public-data.noaa_gsod.gsod1930`
  UNION DISTINCT
`bigquery-public-data.noaa_gsod.gsod1931`
  UNION DISTINCT
`bigquery-public-data.noaa_gsod.gsod1932`
# This is getting out of hand...
```

```
#standardSQL
SELECT
  stn,
  wban,
  temp,
  year
FROM

`bigquery-public-data.noaa_gsod.gsod*`
# All gsod tables
```

So, much like using that like operator, when we looked at finding the charities that had help in the name, you could actually operate over the table names in your FROM statement and do a wildcard match in UNION, through the use of this asterisk here. This is pretty cool.

So let's cover the SQL syntax there on the right. So same thing, you're selecting the fields that you want from all the tables. And then, you're adding the FROM clause, the datasets there. And for that table, you see it begins with a prefix gsod and then, there's an asterisk. So, any tables that are in that dataset in its entirety that match that prefix are going to be included. And because the dataset is very rigid and structured, so it's going to be year, year, year for each of those different tables. It's going to bring together all the historical, weather data tables from 1929 to current. You're talking about a lot of data, millions and millions of records, just with one single line of code there.



Filtering with a **table wildcard** * and **_TABLE_SUFFIX_**

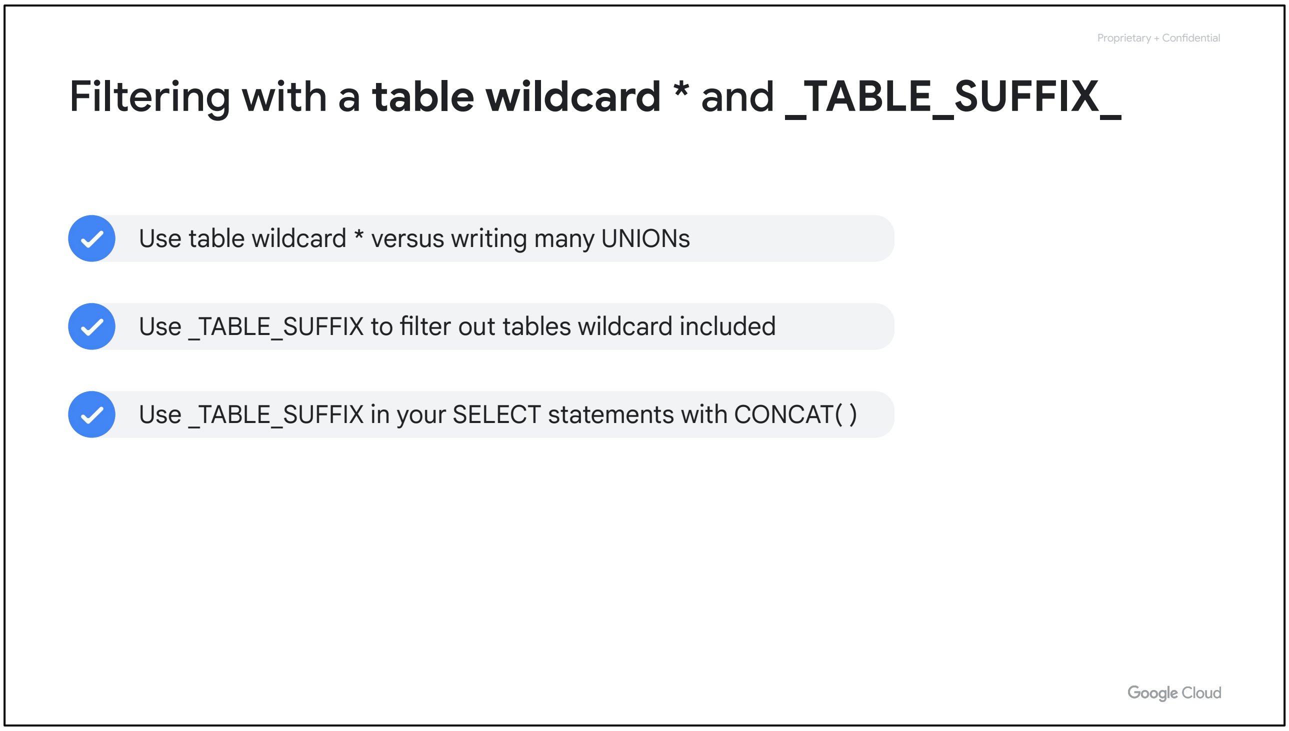
Use _TABLE_SUFFIX to filter out tables included

Be as granular as you can

- e.g. .gsod2* instead of .gsod* if you only care about the year 2000 onward

```
#standardSQL
SELECT
  stn,
  wban,
  temp,
  year
FROM

`bigquery-public-data.noaa_gsod.gsod*`

# All gsod tables after 1950
WHERE _TABLE_SUFFIX > '1950'
```

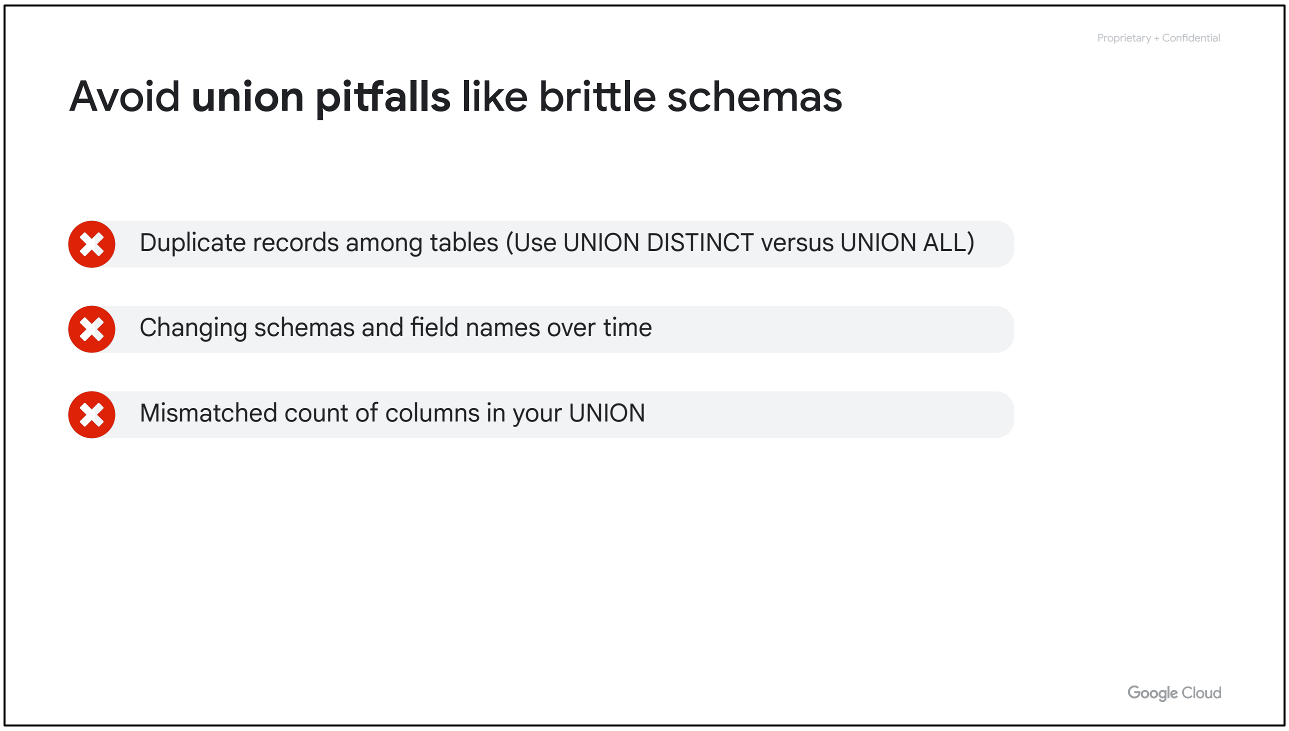
All right, so what happens if you wanted to filter out for just a subset of the tables? So say you just wanted to find and match together all the temperatures from 1950 or after 1950. And here's another reserved keyword that's specific to BigQuery. You can use the TABLE SUFFIX to grab what you're actually matching on in that wildcard. And then, you can use that TABLE SUFFIX in a normal SQL operations. So here you can say, all right well, match everything but only include those tables that are after 1950 year. And, of course, you normally want to be as granular or make that prefix as long as you can for performance reasons just through limiting the amount that you're actually matching on. So TABLE SUFFIX and the wildcard are two very useful concepts.



Filtering with a **table wildcard** * and _TABLE_SUFFIX_

- ✓ Use table wildcard * versus writing many UNIONs
- ✓ Use _TABLE_SUFFIX to filter out tables wildcard included
- ✓ Use _TABLE_SUFFIX in your SELECT statements with CONCAT( )

And that's the subject of our next key message. So, if you have many, many, many, many UNIONs across data sets that have data tables with structured standardized names, make use of that table wildcard. And also, what's really, really nice is this table suffix. You can use that to filter out the tables that are actually included. And, in addition, to filtering out in your WHERE clause, you can actually return the table name in your select statement by just calling that table suffix.


Proprietary + Confidential

Avoid union pitfalls like brittle schemas

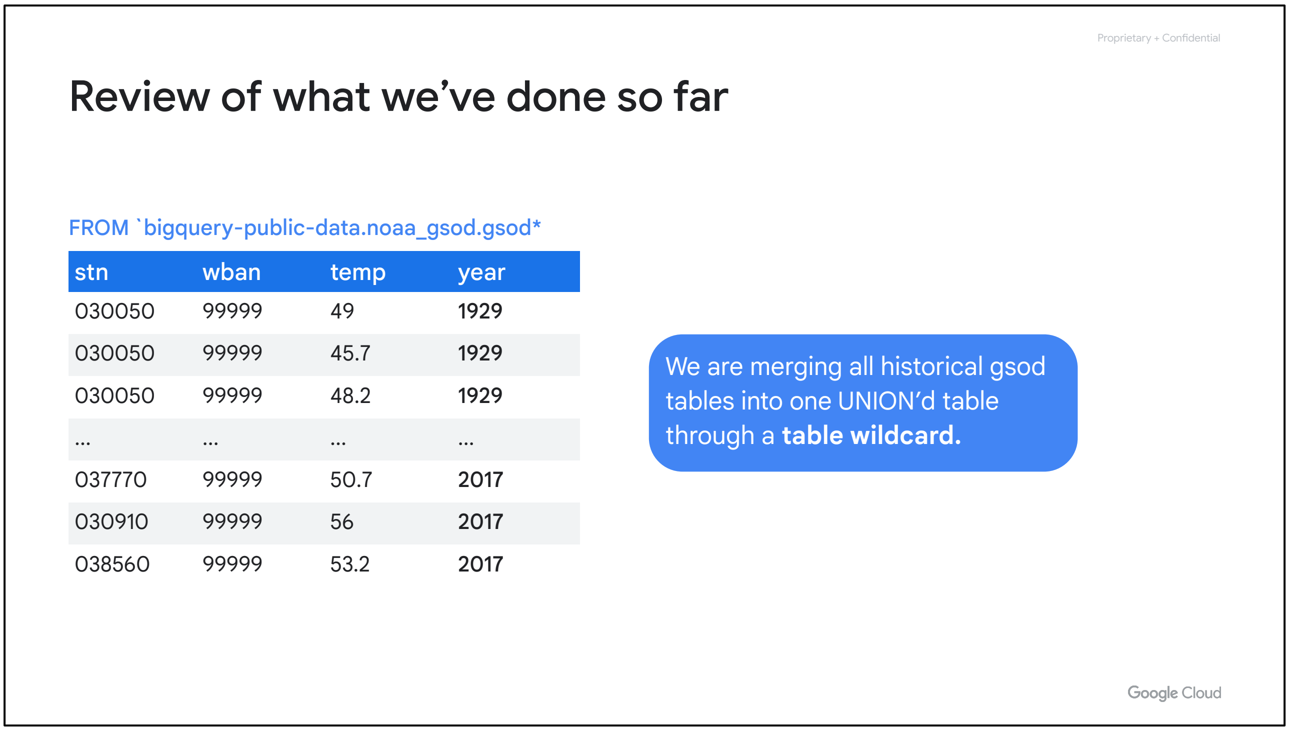
- Duplicate records among tables (Use UNION DISTINCT versus UNION ALL)
- Changing schemas and field names over time
- Mismatched count of columns in your UNION

Google Cloud

So here are some pitfalls when it comes to doing UNIONs inside of SQL. So keep in mind the duplicative records piece that we mentioned before, when you want to remove the duplicative records when you're mashing multiple tables together, you'll be using a UNION DISTINCT. And when you want to just have all the records, no matter if there are duplicate records across multiple tables, you'll be using that UNION ALL. Now it does require that you choose a DISTINCT or an ALL option.

So, as you might imagine, you're hoping that the way these temperature recordings were recorded over time, that the field names haven't changed or the amount of fields that were required hasn't changed over time. Because if they do, when you're mashing these tables together vertically, if you're trying to put tables that don't have the same amount of columns, you're going to get a mismatch and you get an error in your UNION.

So, it's very good to do a preprocessing exercise to make sure that unlike what we saw in Dataprep, where you can have fields that are present in one data set and not in the other, in a true SQL UNION, those tables need to match up exactly in the count of columns. And now you can do technically, you can have different column names and match together based on what's called the index of where that column appears. But generally, what I like to do just for my own sanity is make sure the names and the count of columns matched as well.



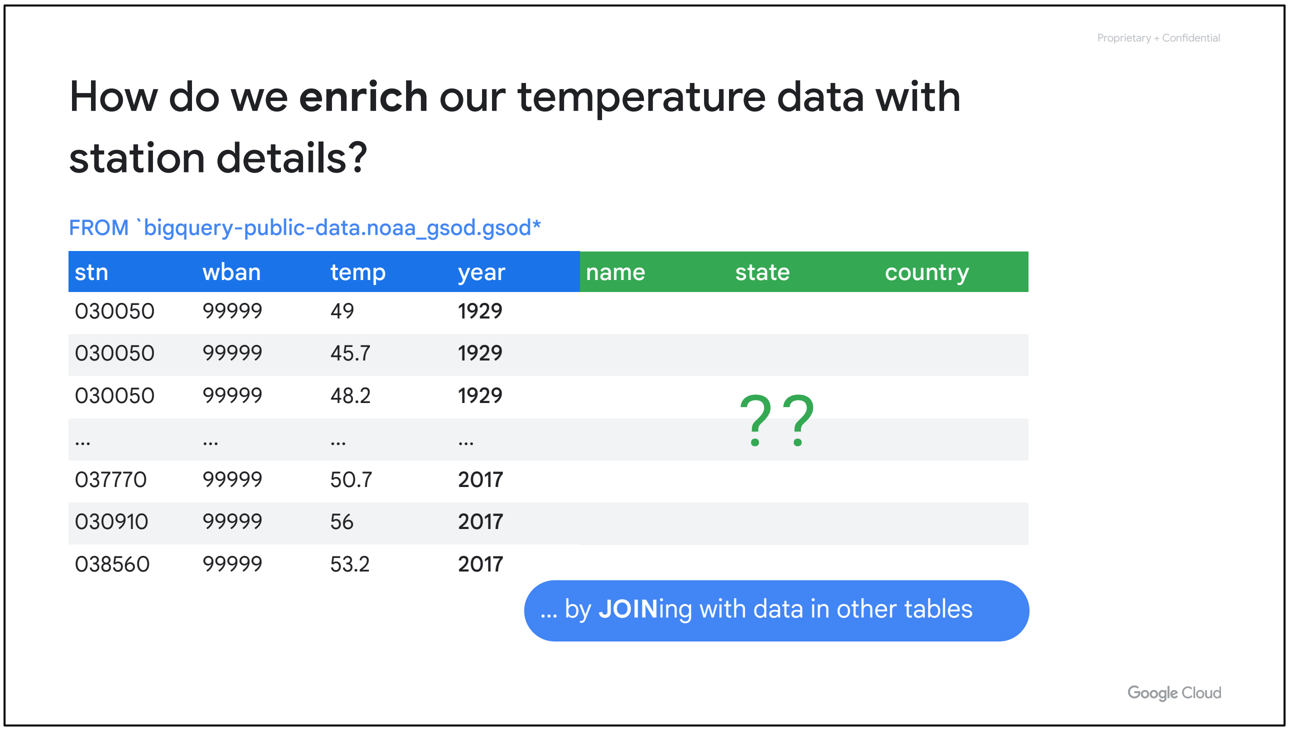
Review of what we've done so far

FROM `bigquery-public-data.noaa_gsod.gsod*

stn	wban	temp	year
030050	99999	49	**1929**
030050	99999	45.7	**1929**
030050	99999	48.2	**1929**
...	...	...	...
037770	99999	50.7	**2017**
030910	99999	56	**2017**
038560	99999	53.2	**2017**

We are merging all historical gsod tables into one UNION'd table through a **table wildcard.**

So recap, we have join, UNION'd rather, we've matched together all that historical data from the past all the way to the current, and that's a lot of temperature readings.



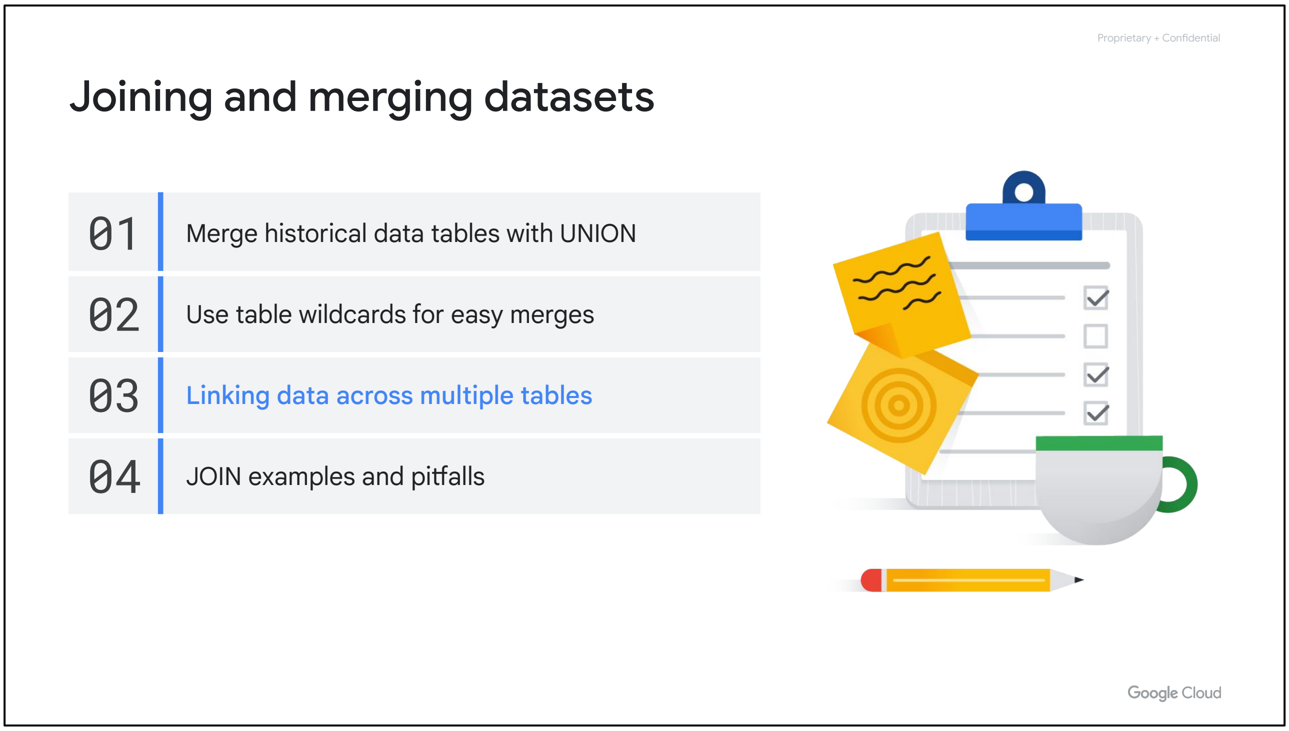
How do we **enrich** our temperature data with station details?

FROM `bigquery-public-data.noaa_gsod.gsod*

stn	wban	temp	year	name	state	country
030050	99999	49	1929			
030050	99999	45.7	1929			
030050	99999	48.2	1929		??	
...	...	...	...			
037770	99999	50.7	2017			
030910	99999	56	2017			
038560	99999	53.2	2017			

... by **JOIN**ing with data in other tables

But, we still don't know where those temperature readings came from. And that's where we're actually going to be joining this massive amounts of consolidated temperature recordings with another dataset, that is just a single table on that station reading information.



Joining and merging datasets

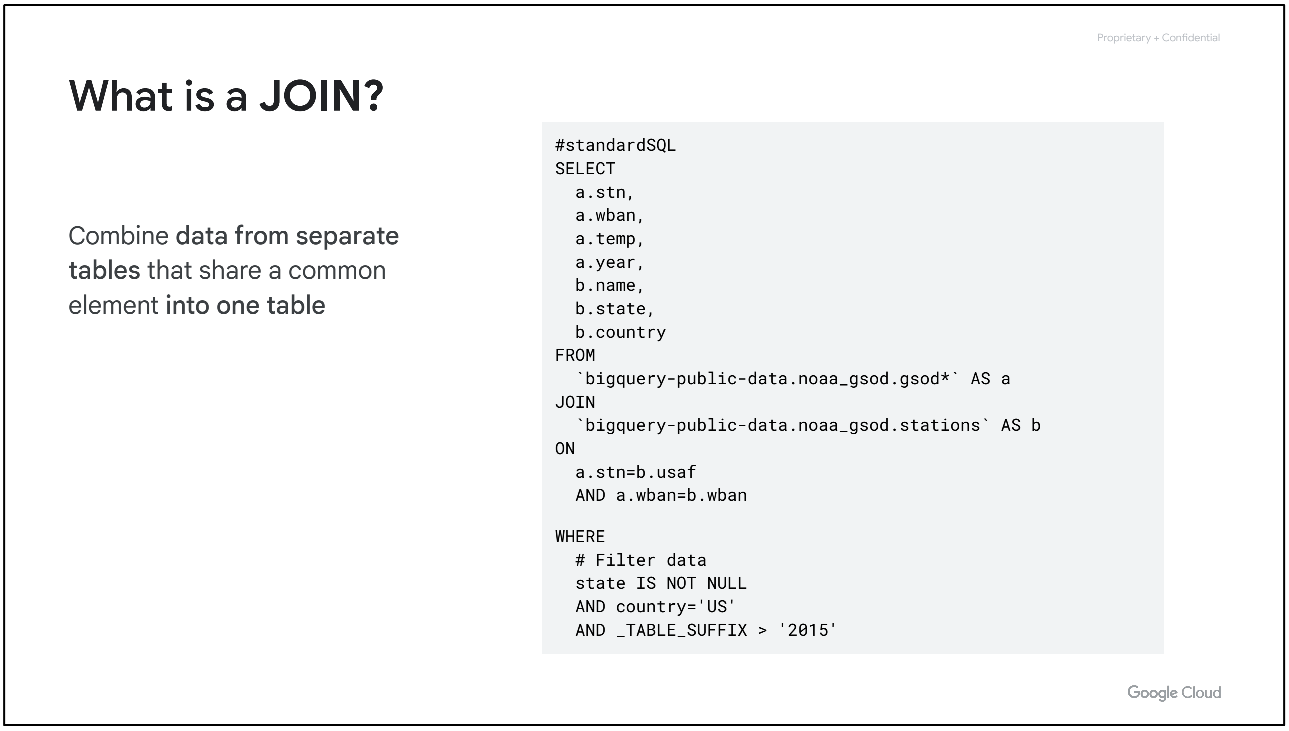
01 Merge historical data tables with UNION

02 Use table wildcards for easy merges

03 Linking data across multiple tables

04 JOIN examples and pitfalls

OK, so let's work on JOINs. How do we link across multiple tables?

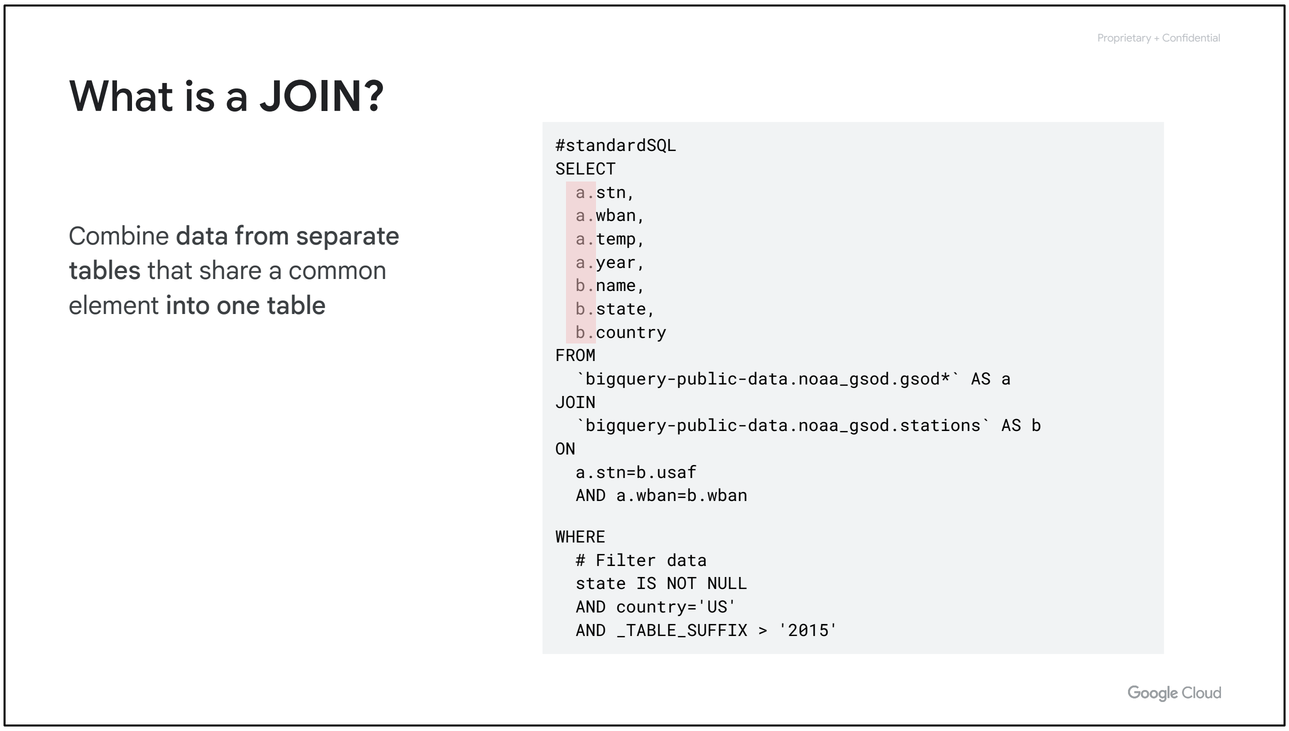


What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

So conceptually, a JOIN combines data from separate tables into one table. Now technically, a UNION will do that too because it will consolidate that output, but JOINs, you can think about doing it horizontally. So you could actually add in more fields, things like the charity name for example in your IRS example, or the station name in this weather recording example. And this is your JOIN syntax. So you are selecting the fields that you want from both tables.

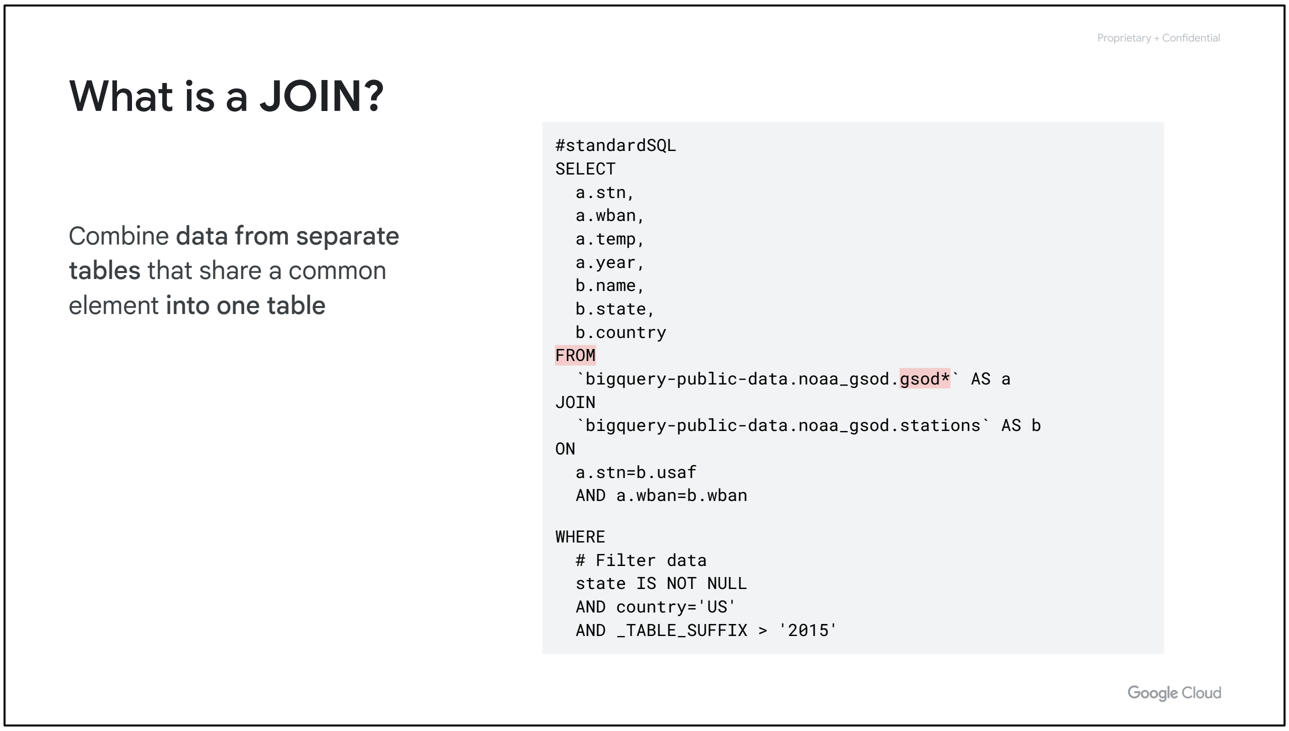


What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

You'll notice that a. b. when you get into that in just a second.

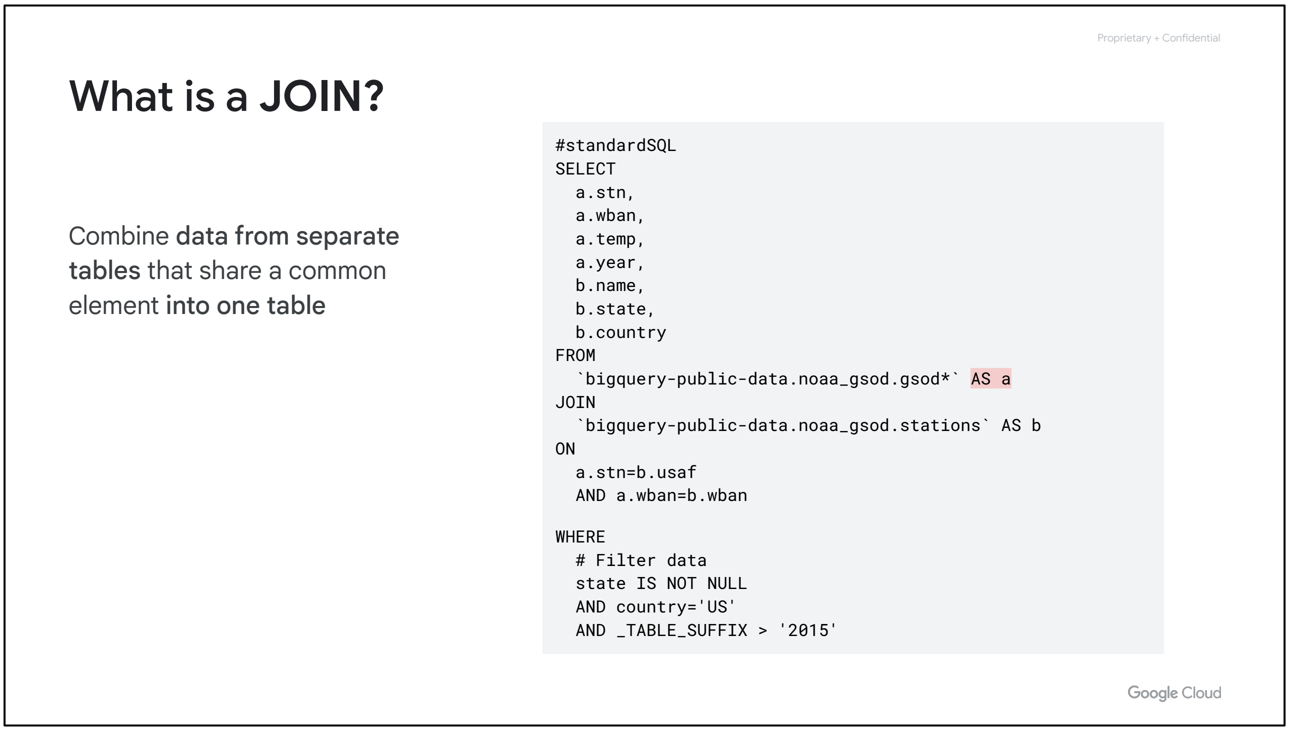


What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

FROM, and you see the UNION wildcard here, on all of that historical gsod year, year, year, year tables.

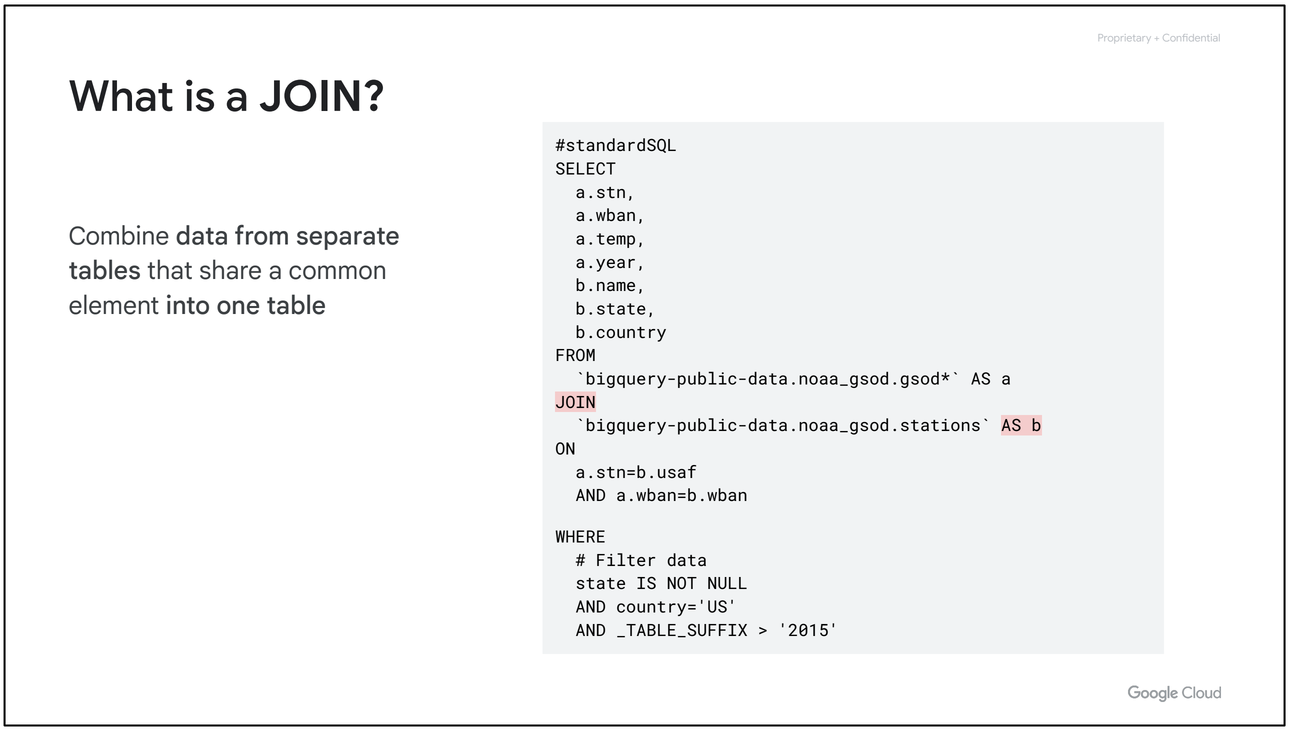


What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

And you're specifying that as an alias, you're saying AS a. The reason why you're going to do that has become apparent in just a second.

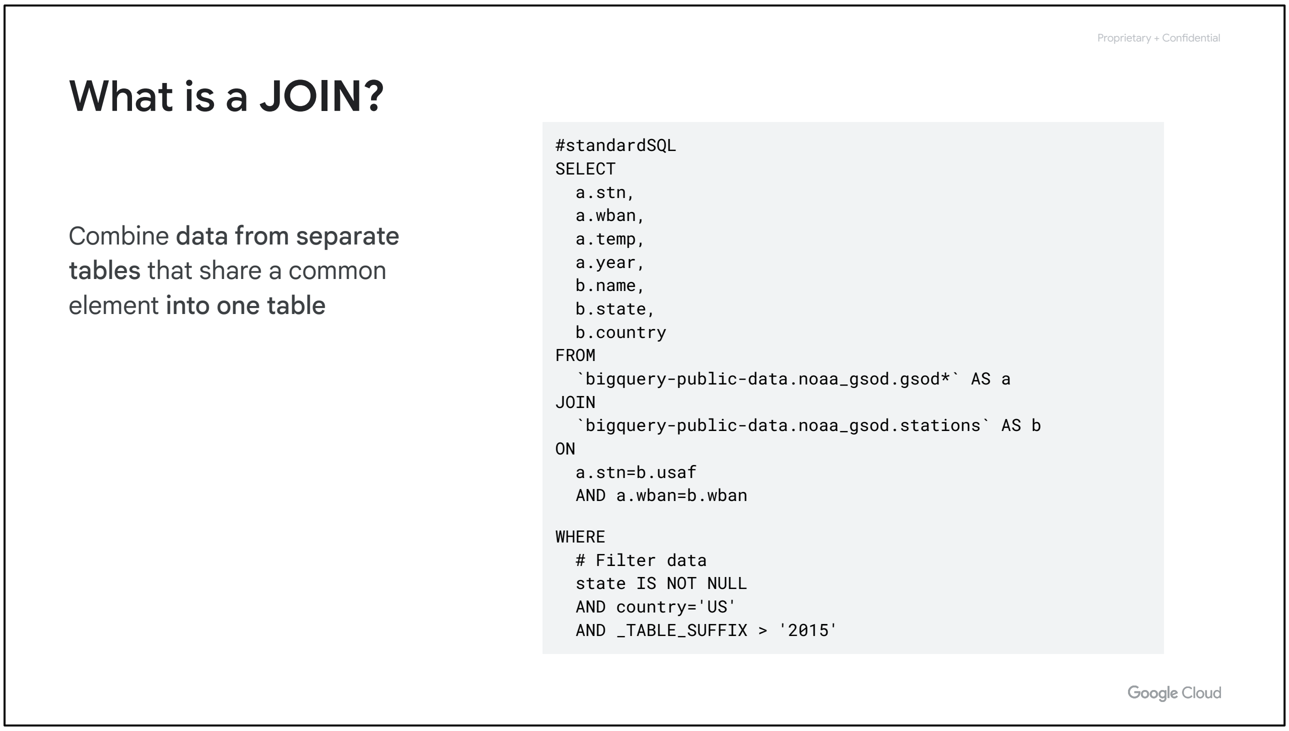


What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

And you're joining that on a key, which we're going to cover, into the station's data, and we're saying that AS b.



Proprietary + Confidential

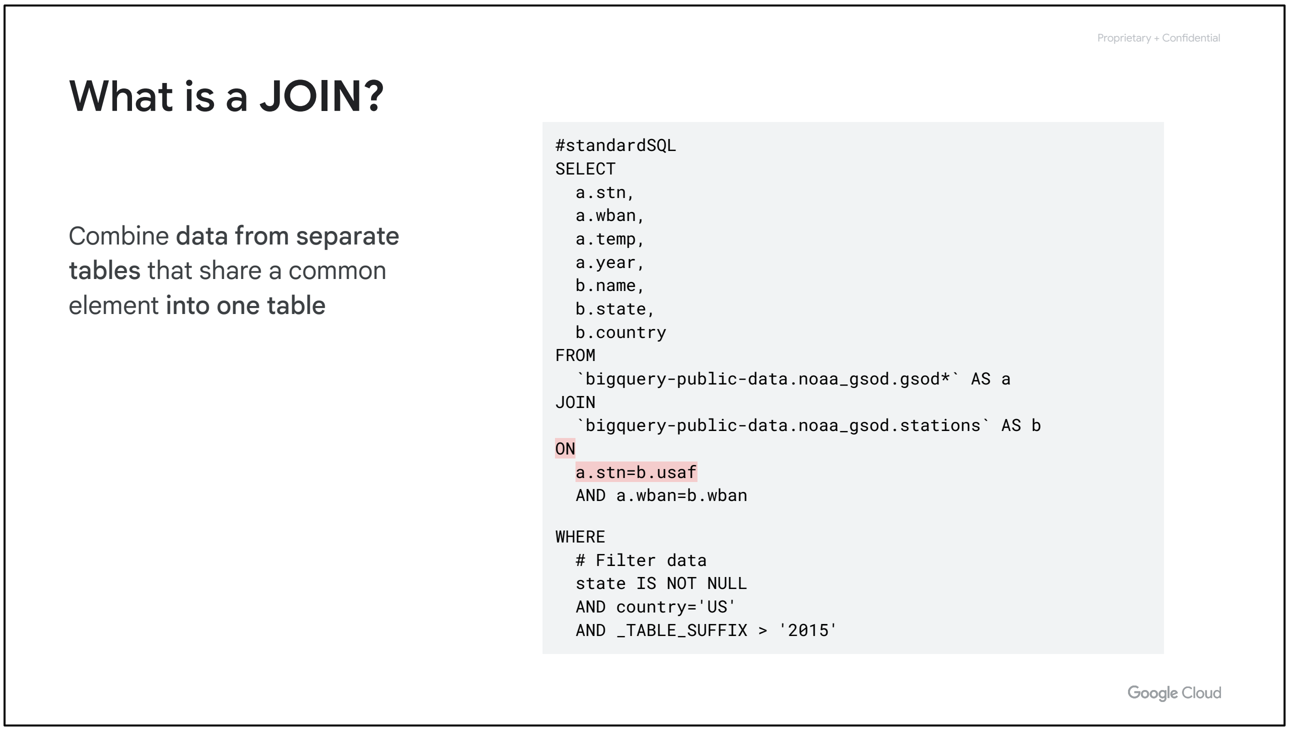
What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

Google Cloud

And why are we using these aliases a and b? It's because if there are ambiguous field names, like say the word name or say the word station is common across both tables, when you join these two together and you're adding new fields, you can't have any name collisions. So having an alias actually specifies where that comes from. And it's actually technically not required for a. field name if that name is unique across all the tables that you're joining.

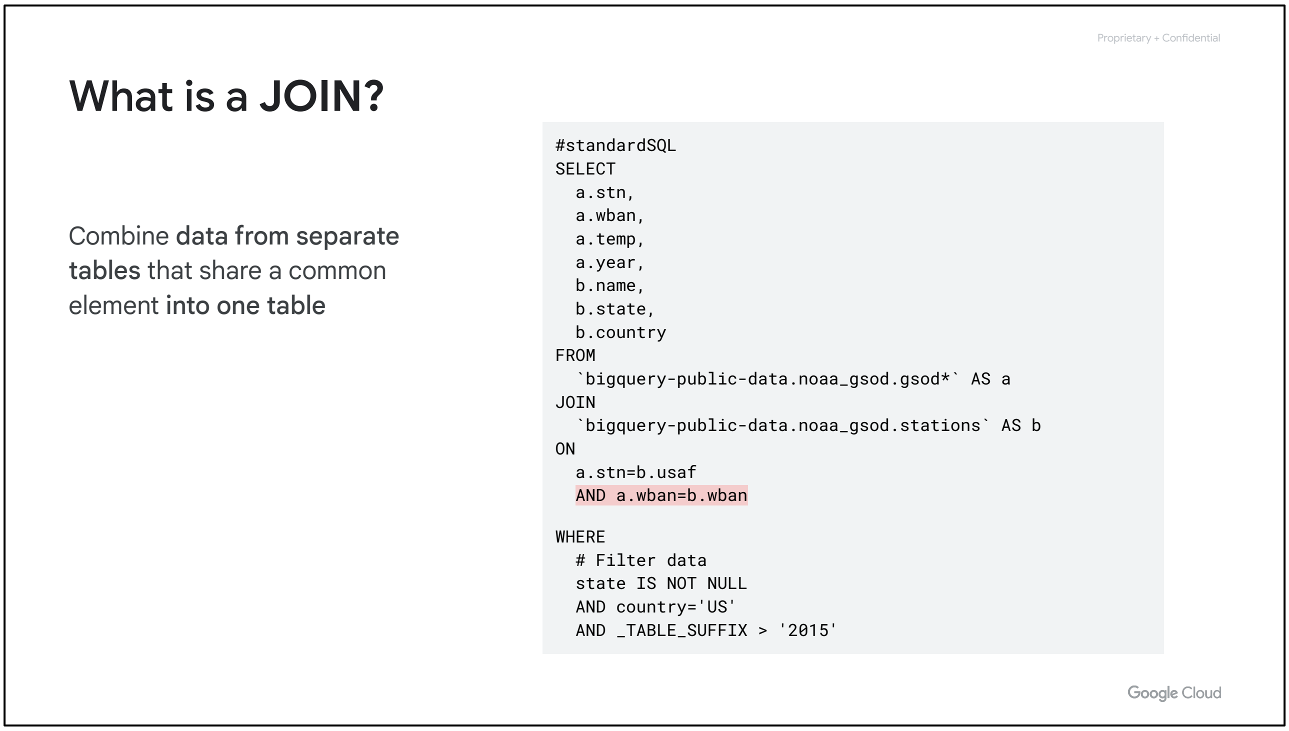


What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

So that as the actual JOIN itself, now the condition is match these records, bring these tables together horizontally on the weather recordings which is a, a.station is equal to the b.USAF. And that's again, you can actually match on fields that have names that don't match. But again, it's just going to look at those values and that's the case here, we see the station name actually doesn't match the USAF name in the station's table.



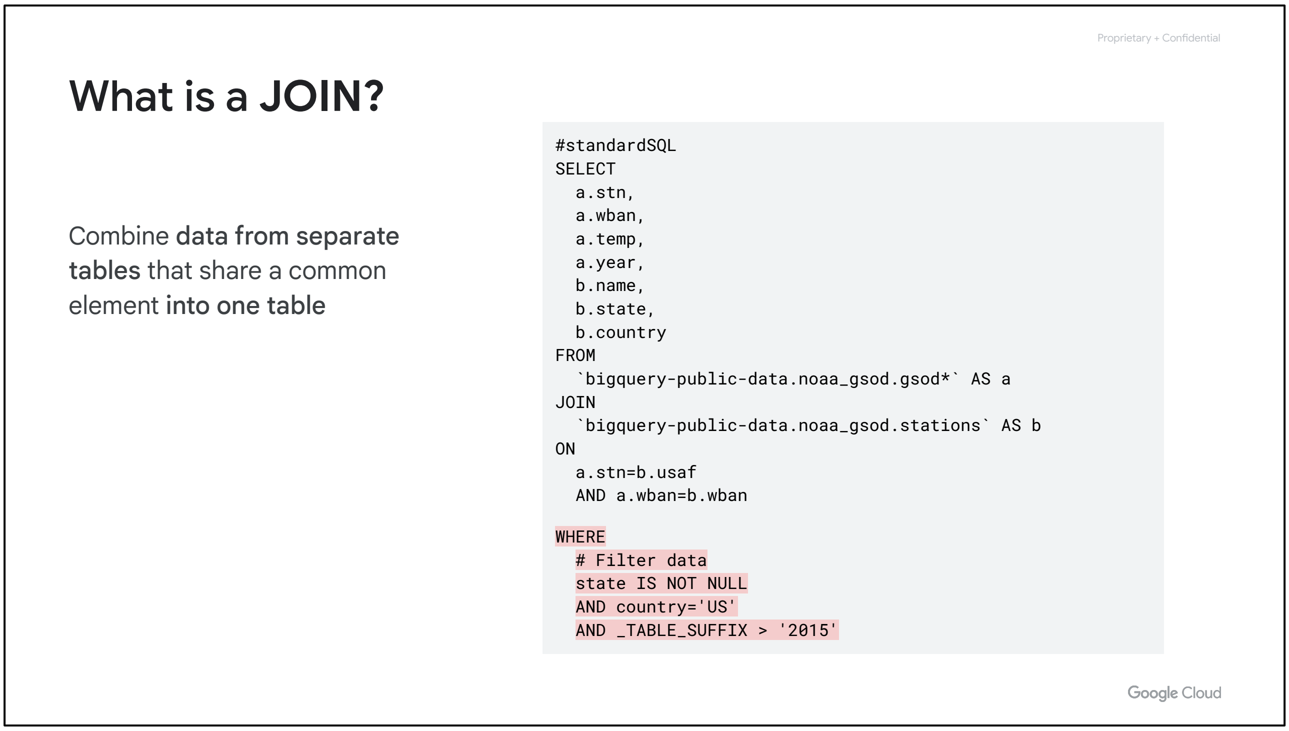
What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

Google Cloud

And also, you can have that additional piece of the JOIN key, a.wban=b.wban. So there's a lot going on there.

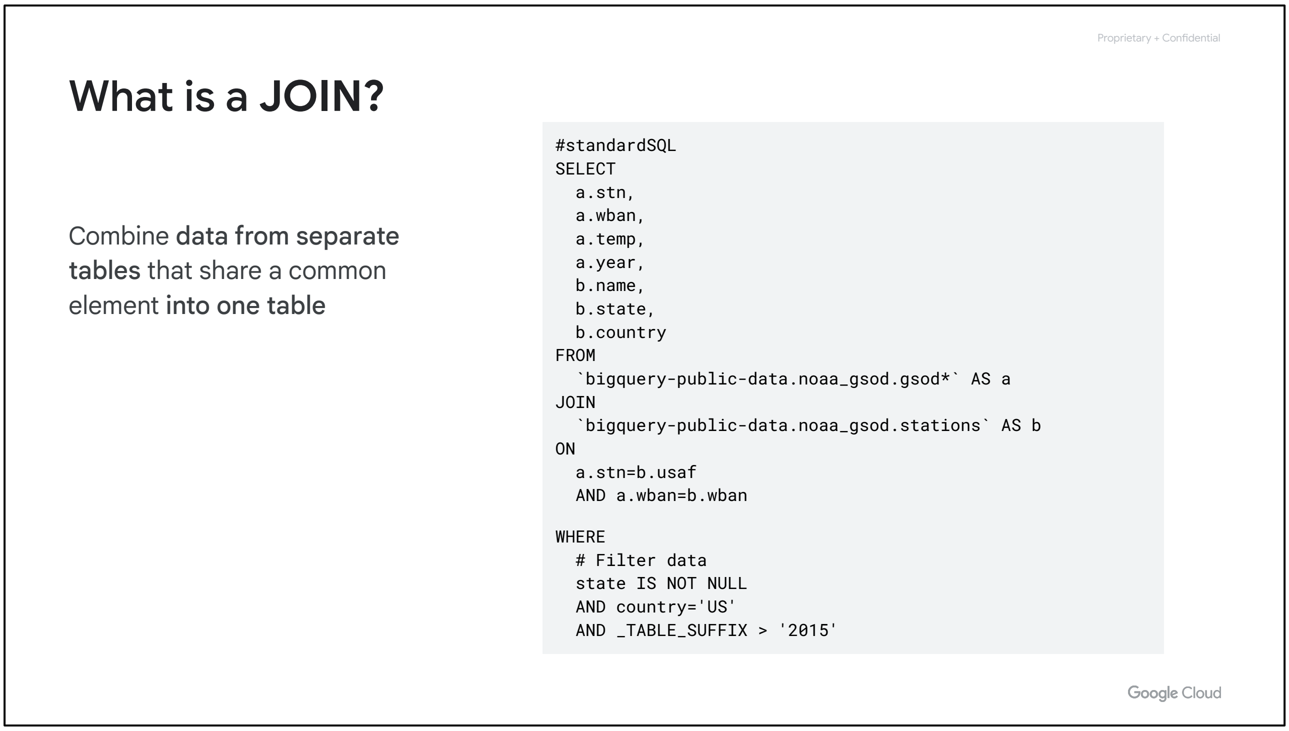


What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

And then of course, we apply a filter all the way at the bottom. That just does a bunch of different things. It filters for only US temperature readings and table data that's after 2015. And the US state is not null.



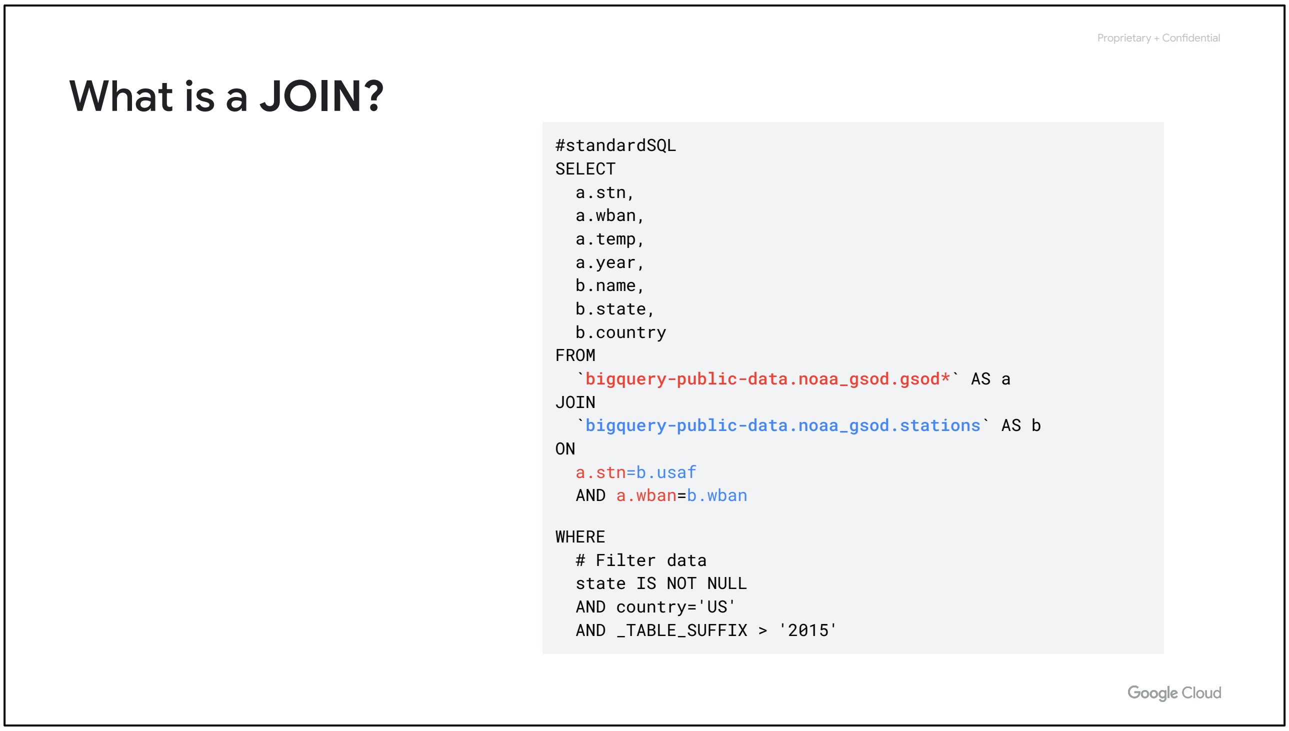
What is a JOIN?

Combine **data from separate tables** that share a common element **into one table**

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

Google Cloud

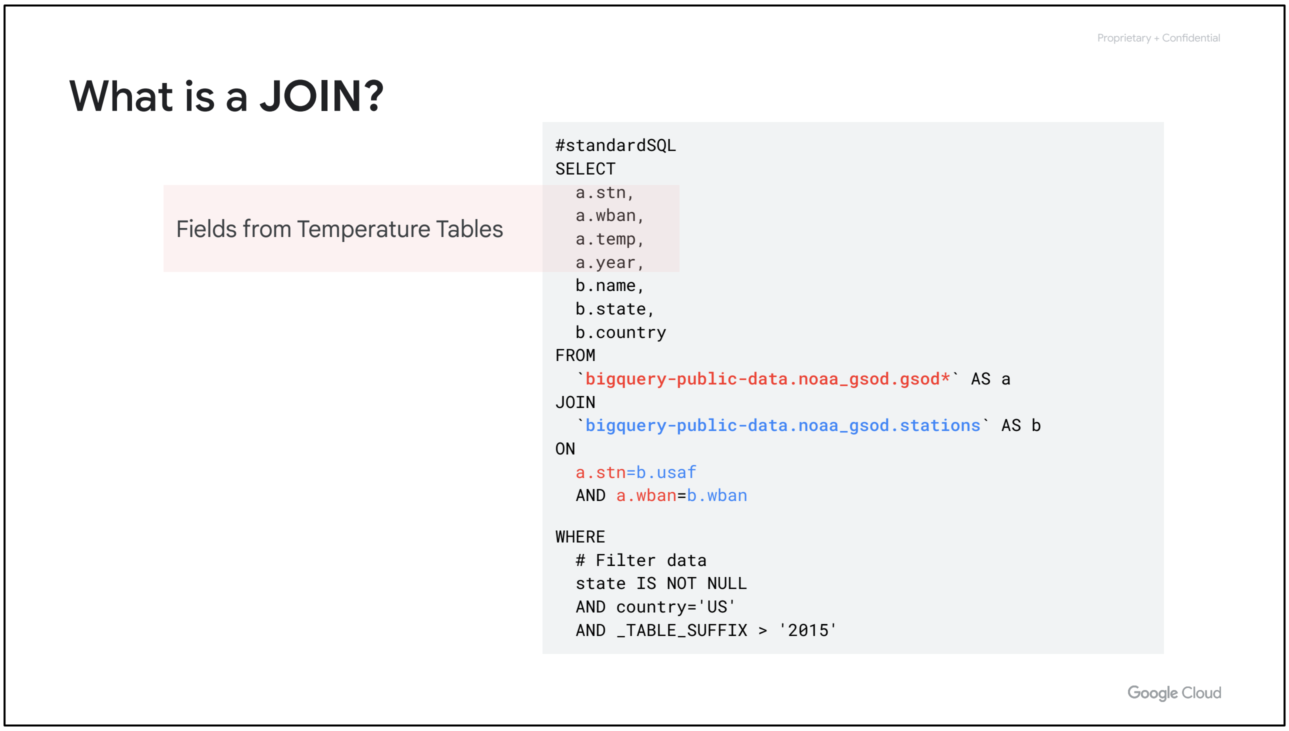
But the main thing that we want to focus on right now is that JOIN condition. So there's a lot of things that are going on. One of the things that we haven't covered yet is what type of JOIN that we're actually doing to link these two things together.



What is a JOIN?

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

So let's do a quick review before we get into the different types of JOINs.



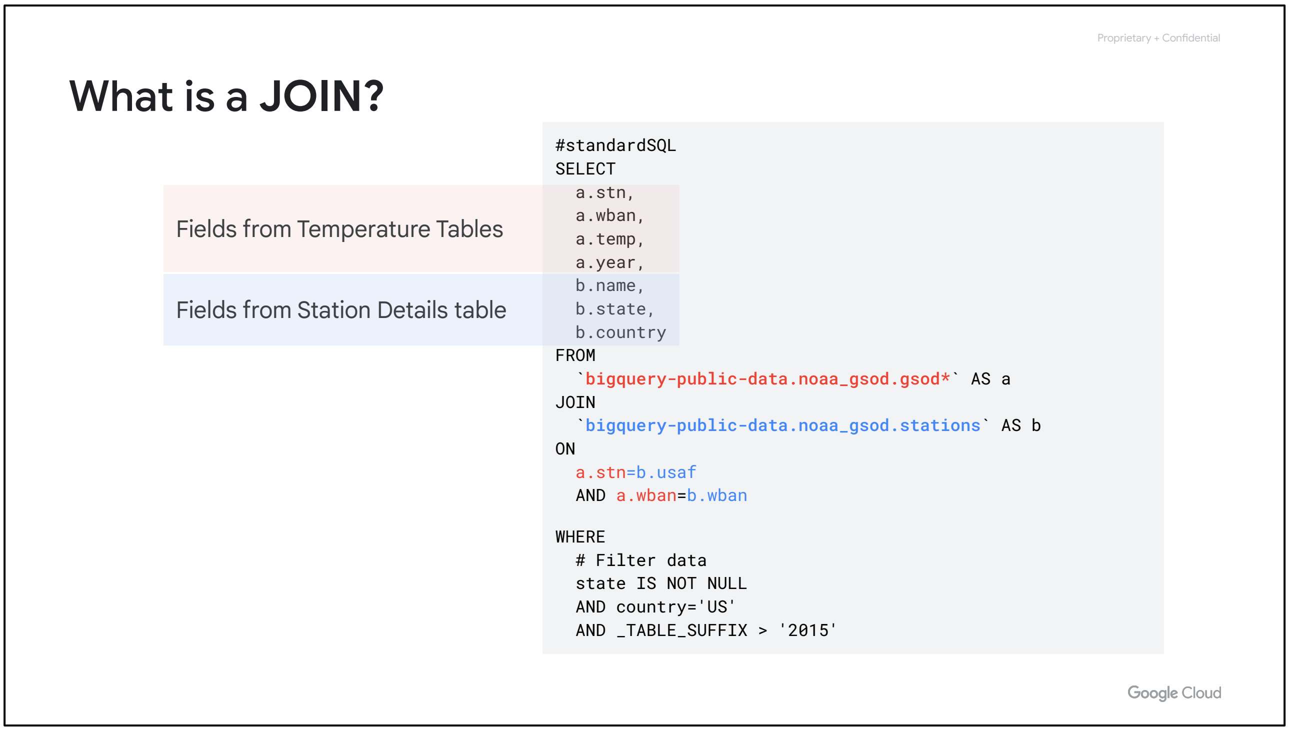
What is a JOIN?

Fields from Temperature Tables

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

Google Cloud

The top table listed in the red box there, those are all the fields that we're pulling from the individual temperature recordings across all those different tables.



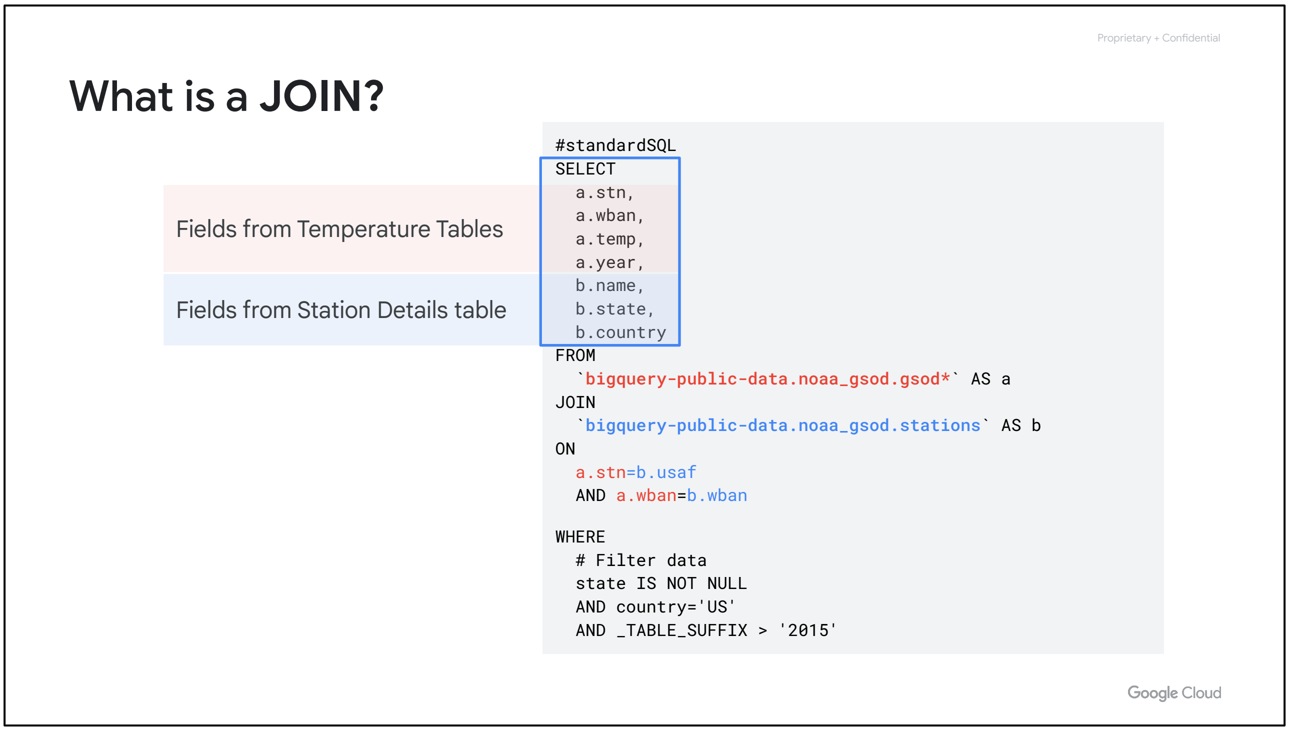
What is a JOIN?

Fields from Temperature Tables

Fields from Station Details table

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

And then the blue box, those are the fields that are coming from that separate table.



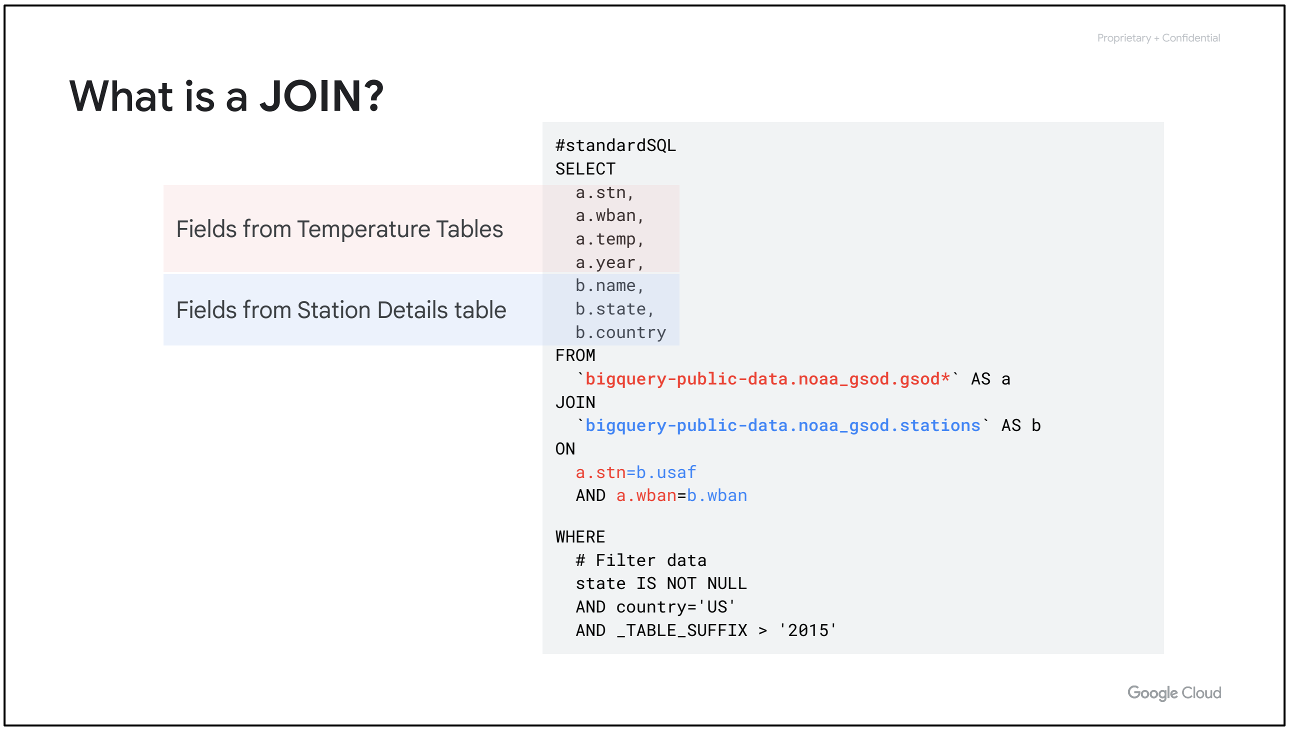
What is a JOIN?

Fields from Temperature Tables

Fields from Station Details table

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

Then notice we're listing them all together in that SELECT statement.


Proprietary + Confidential

What is a JOIN?

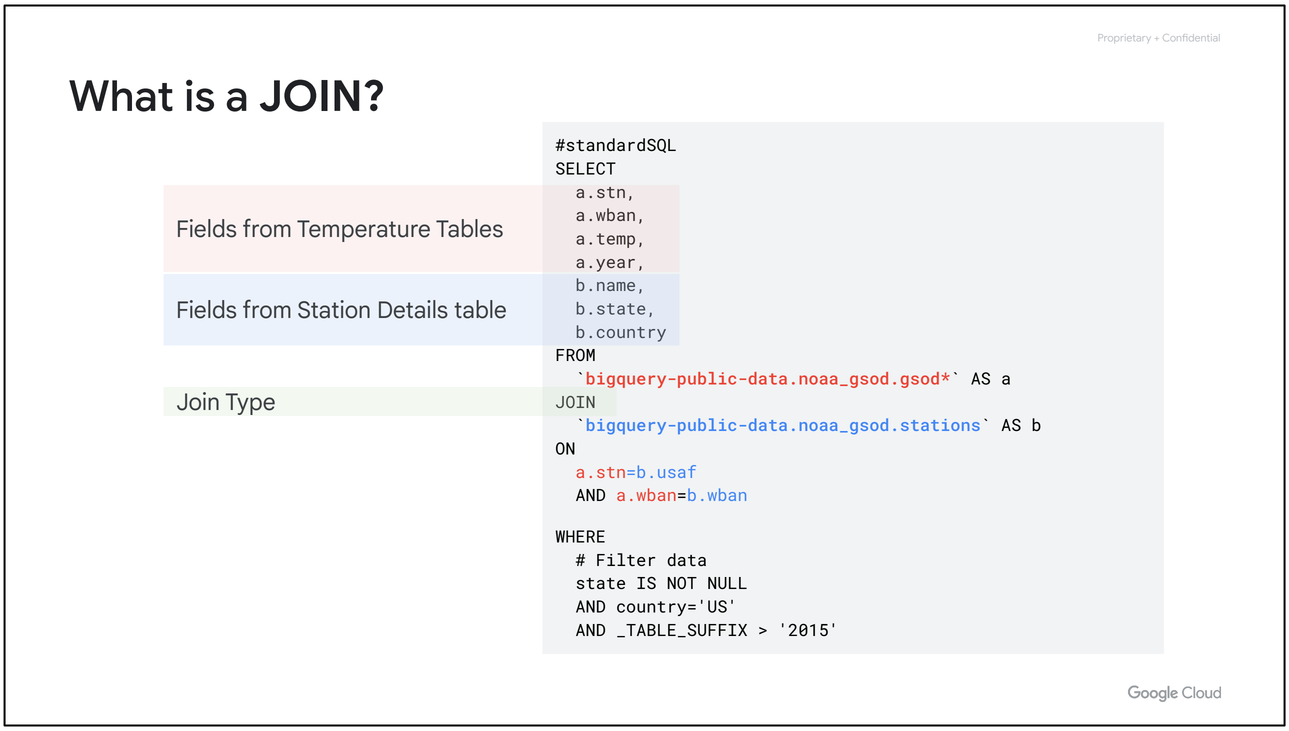
Fields from Temperature Tables

Fields from Station Details table

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

Google Cloud

A general good best practice for you is, if you have table data that's coming from multi-different sources, you can actually break up in your SELECT statement with a comment, basically saying "Oh, this is your temperature readings and then these are your station readings as well." That's purely for the readability of your code.



What is a JOIN?

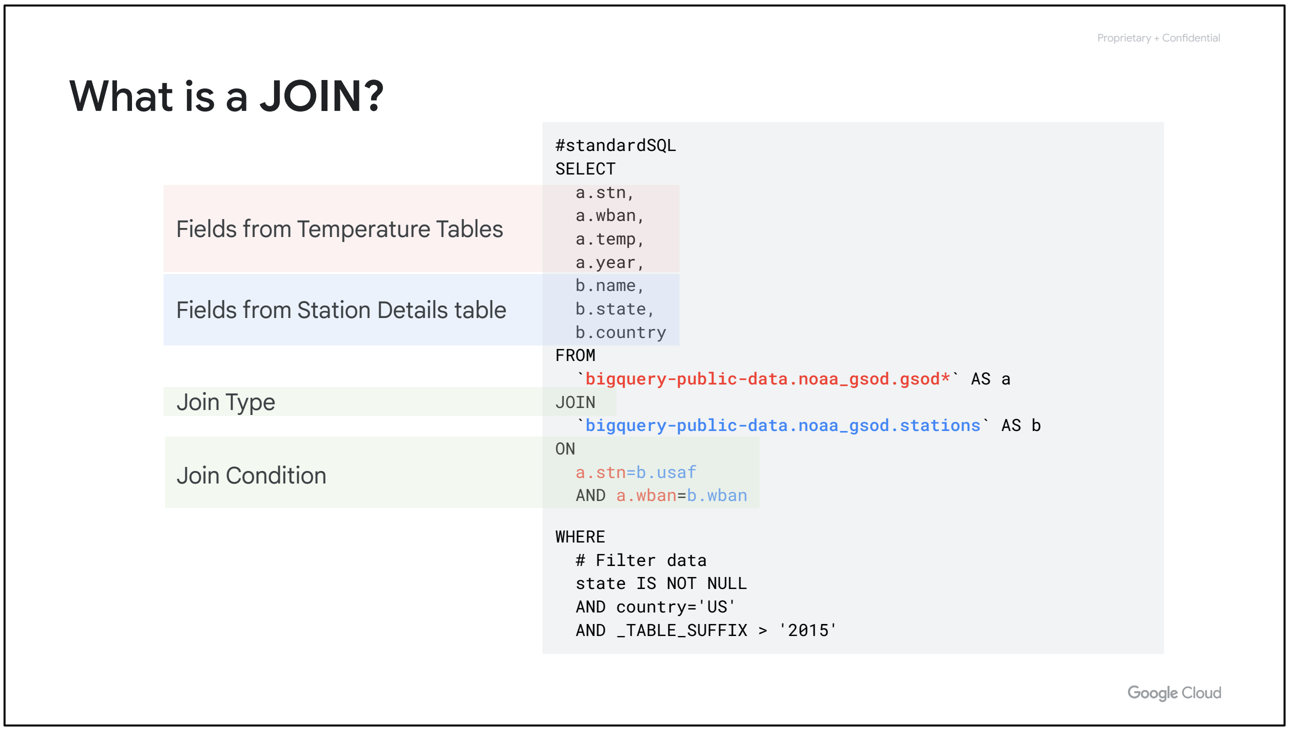
Fields from Temperature Tables

Fields from Station Details table

Join Type

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

And then capital JOIN is your join. We'll go on to the types in a minute.


Proprietary + Confidential

What is a JOIN?

Fields from Temperature Tables

Fields from Station Details table

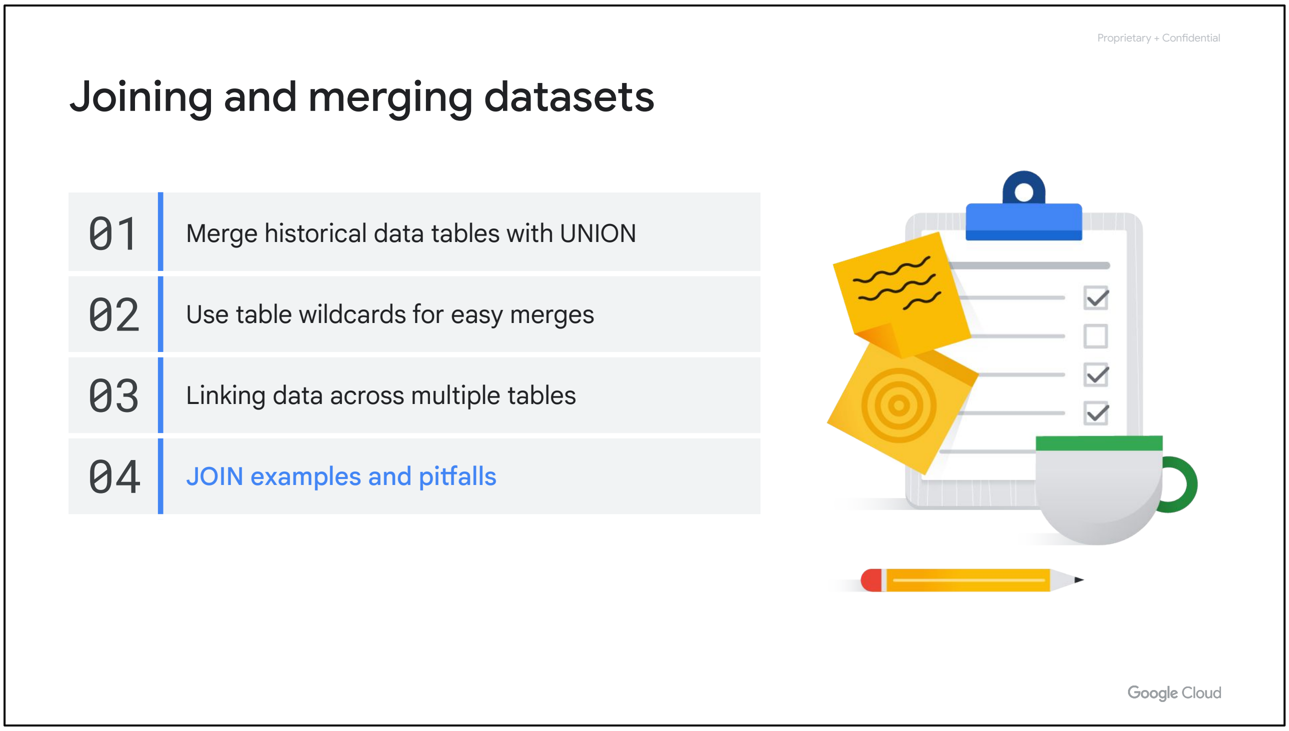
Join Type

Join Condition

```
#standardSQL
SELECT
  a.stn,
  a.wban,
  a.temp,
  a.year,
  b.name,
  b.state,
  b.country
FROM
  `bigquery-public-data.noaa_gsod.gsod*` AS a
JOIN
  `bigquery-public-data.noaa_gsod.stations` AS b
ON
  a.stn=b.usaf
  AND a.wban=b.wban

WHERE
  # Filter data
  state IS NOT NULL
  AND country='US'
  AND _TABLE_SUFFIX > '2015'
```

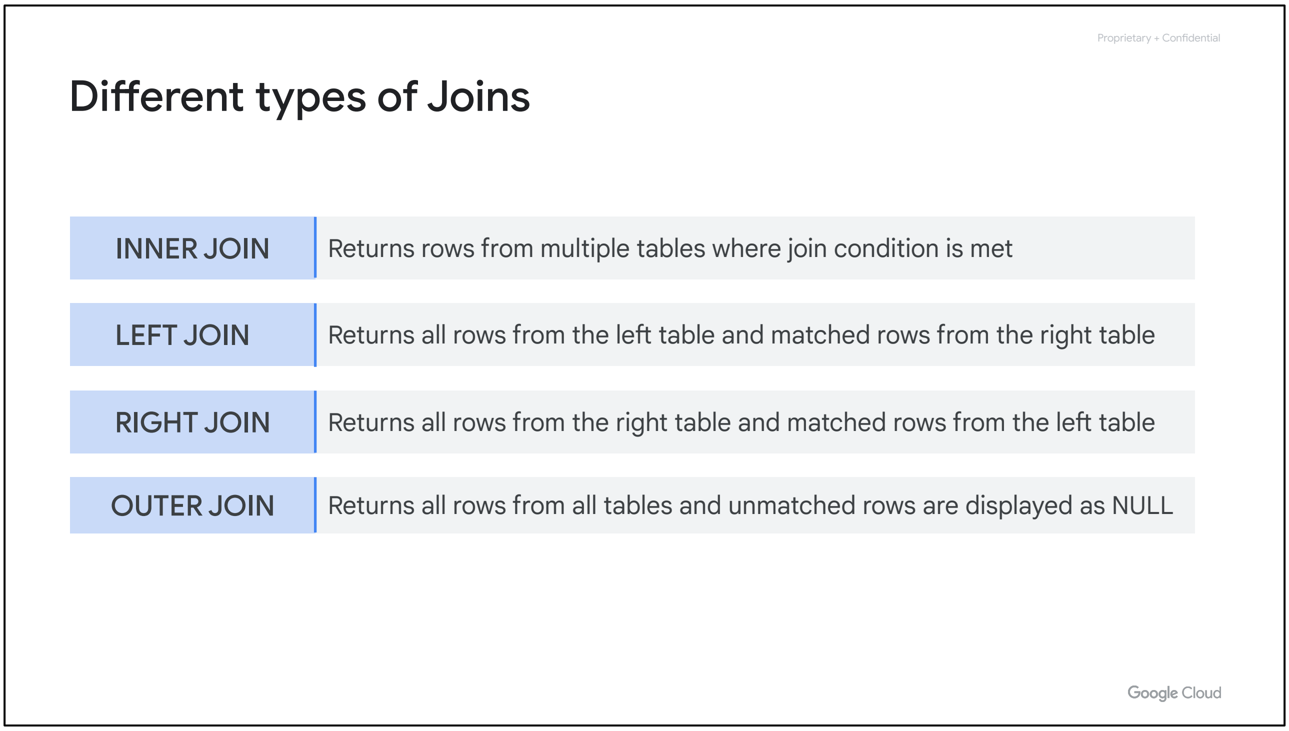
Google Cloud

And then your joining condition, again you can have more than one JOIN key.



Joining and merging datasets

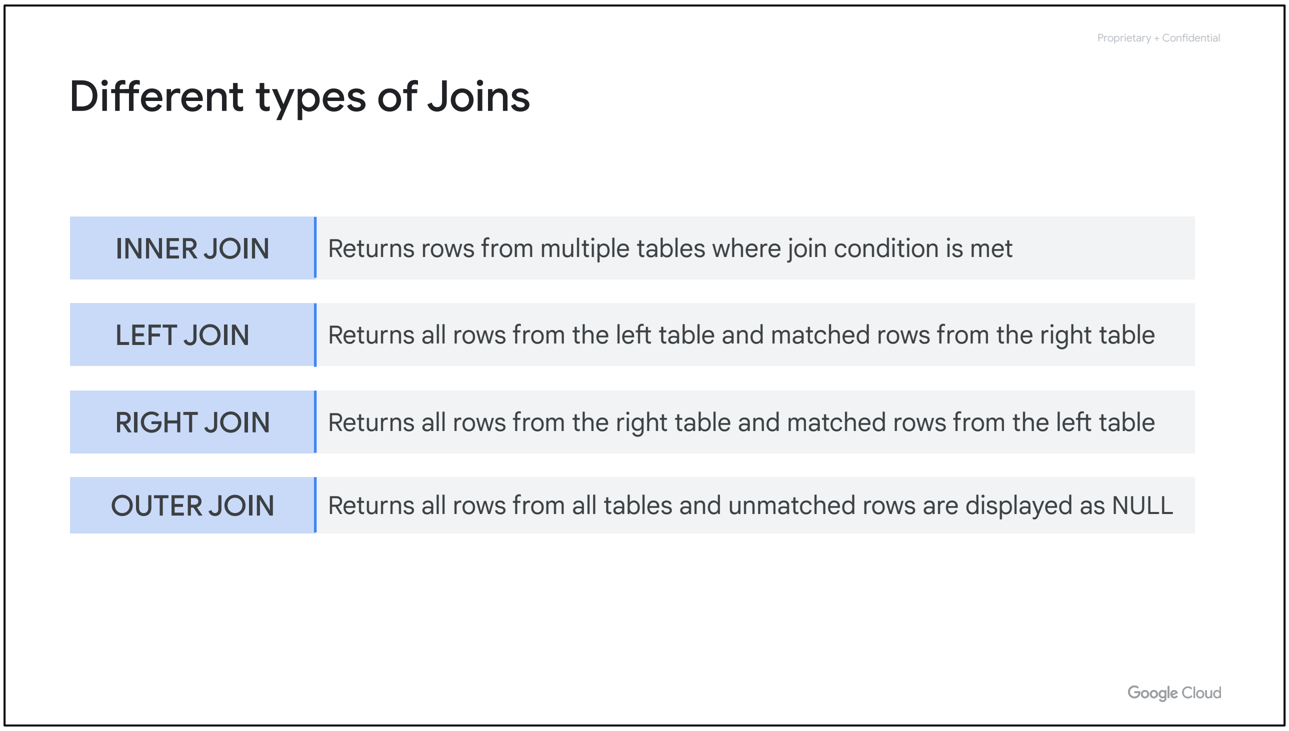
Let's talk a little bit more about JOIN specifics and get into some examples.



Different types of Joins

INNER JOIN	Returns rows from multiple tables where join condition is met
LEFT JOIN	Returns all rows from the left table and matched rows from the right table
RIGHT JOIN	Returns all rows from the right table and matched rows from the left table
OUTER JOIN	Returns all rows from all tables and unmatched rows are displayed as NULL

So, there's many different types of joins that are out there and those of you who have practiced SQL before, the Venn diagram slide is coming, I assure you. So, you have an inner join, left join, right join, a full outer join. Now, the best way to think of this is through an example and I have a handy-dandy whiteboard that I'm just going to pull up,...

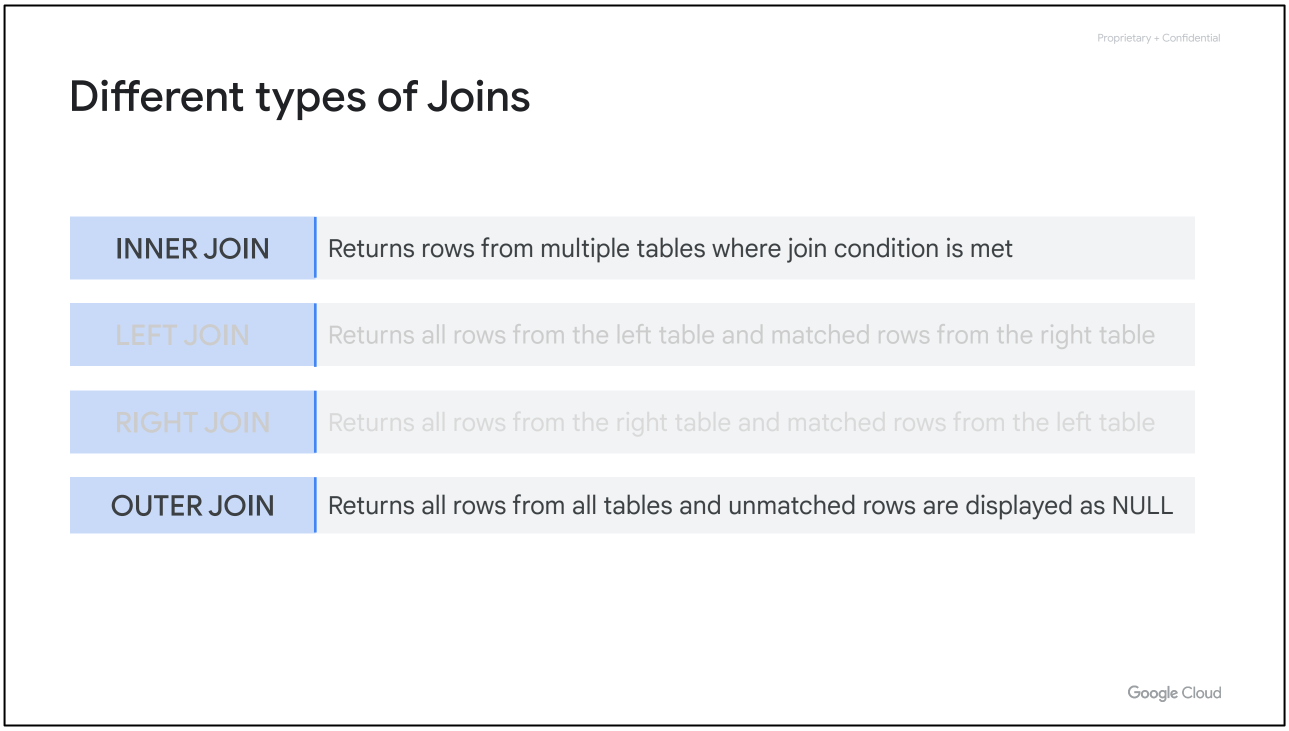


Different types of Joins

INNER JOIN	Returns rows from multiple tables where join condition is met
LEFT JOIN	Returns all rows from the left table and matched rows from the right table
RIGHT JOIN	Returns all rows from the right table and matched rows from the left table
OUTER JOIN	Returns all rows from all tables and unmatched rows are displayed as NULL

Okay and that's exactly what we talked about here.

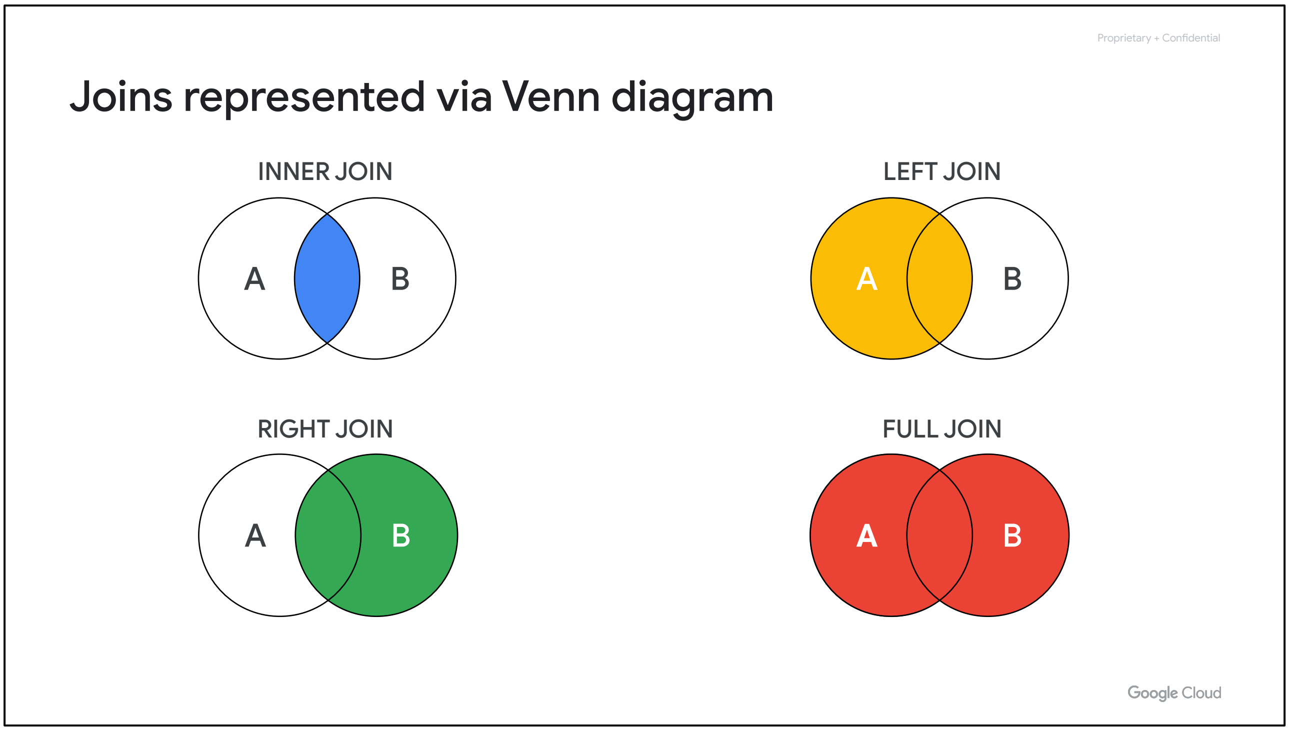
- So, **inner join** is that intersection piece of that Venn diagram.
- **Left join** is all the rows from that left side and where they match with the right side.
- **Right join** again it's never seen used because you're just switching the order in SQL literally of your table name.
- And an **outer join** is just a full outer join that is just a way of returning all rows from all tables.



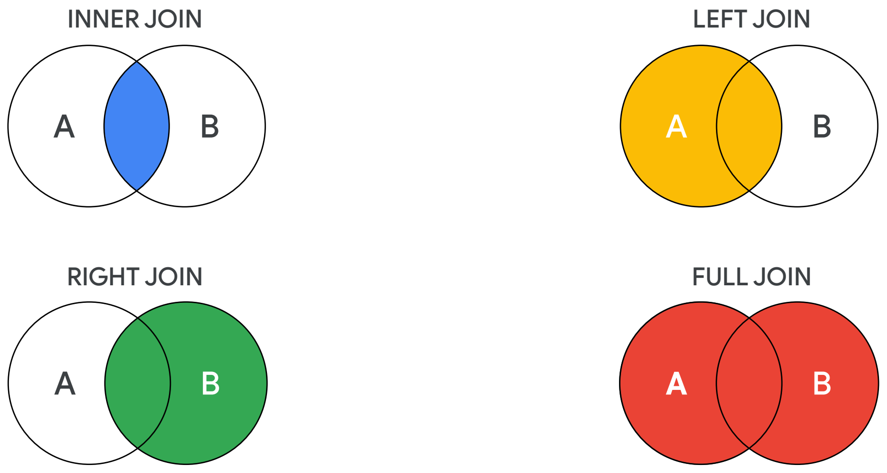
Different types of Joins

INNER JOIN	Returns rows from multiple tables where join condition is met
LEFT JOIN	Returns all rows from the left table and matched rows from the right table
RIGHT JOIN	Returns all rows from the right table and matched rows from the left table
OUTER JOIN	Returns all rows from all tables and unmatched rows are displayed as NULL

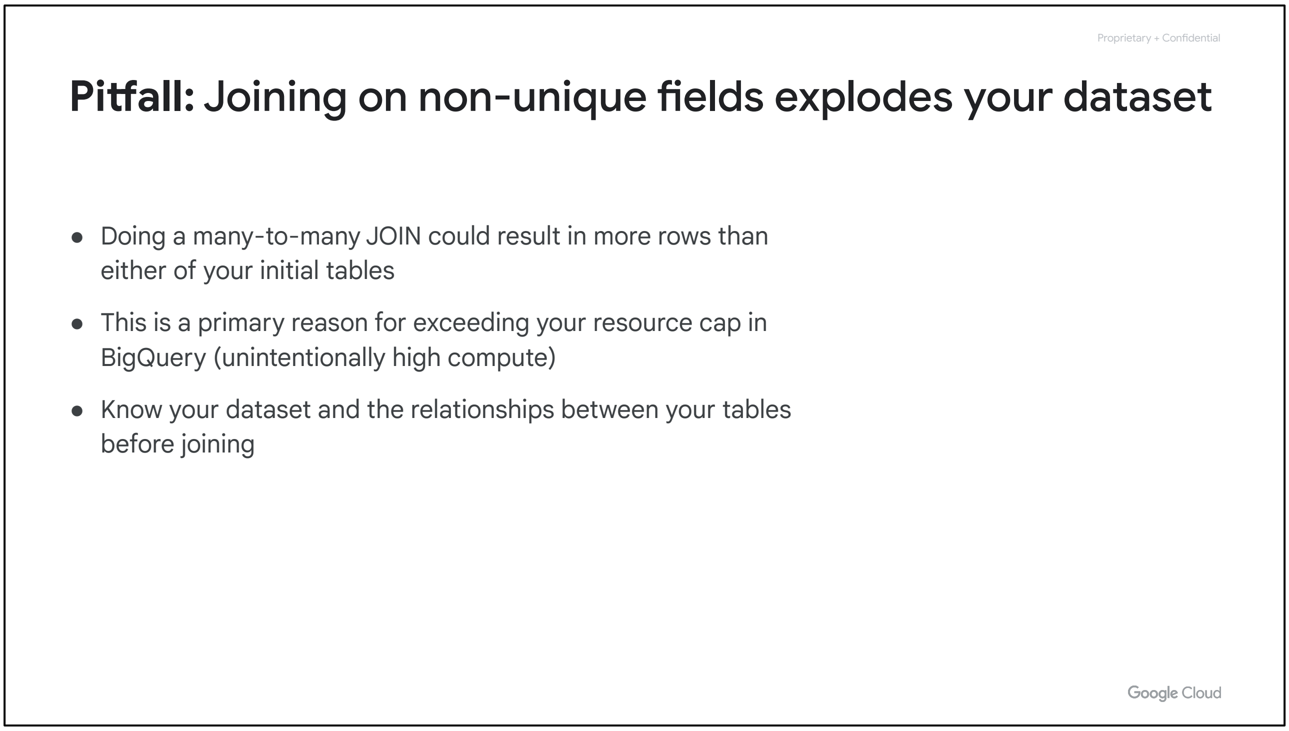
So, there's actually two major types of different joins; inner joins versus outer join. A left is considered an outer join so it's actually left outer join, right outer join and full outer join. Again if you just typed in "join" into SQL, that will default to an inner join, so only when you have matching records.



Joins represented via Venn diagram



As a review, that's exactly what we just covered with our Venn diagram example.



Pitfall: Joining on non-unique fields explodes your dataset

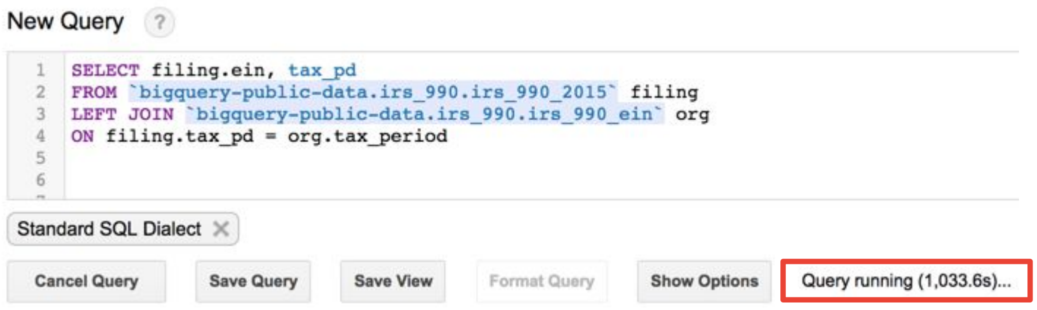
- Doing a many-to-many JOIN could result in more rows than either of your initial tables
- This is a primary reason for exceeding your resource cap in BigQuery (unintentionally high compute)
- Know your dataset and the relationships between your tables before joining

So now one of the key pitfalls that you could run into, and this is why we talk a lot about eliminating those duplicative records, is doing what we call a many-to-many JOIN. Now a many-to-many JOIN, for example, if we had not gone through the exercise for our stations and weather temperature readings exercise, what we could've done is joined on, not necessarily, a distinct or unique key. So you have five records on the right-hand side and you have five records on the left-hand side, and they all match each other. So how many output rows would actually be returned? Would it be zero, would it be five? Or could it be potentially way more than five? And the answer's way more than five, so you'd actually get what's called the cartesian product or 5 times 5. So you'd get 25 output rows, even though your input tables only had 5 rows.

And if you can imagine this at scale, if you don't get that joining key correct, that's when you'll really just blow out all of the resources that you're using. Because having a million record table, joined even on like a 10,000 record table, you could just be exploiting by many, many, many factors of 10, and get a dataset that's outputting way, way, way more rows than what you're expecting. So the end results is a situation you want to avoid.

And the way you can avoid it is the third bullet point here, is knowing the relationship between the tables in your data, before you do that JOIN. So let's walk through an example.

Pitfall: Joining on non-unique fields explodes your dataset

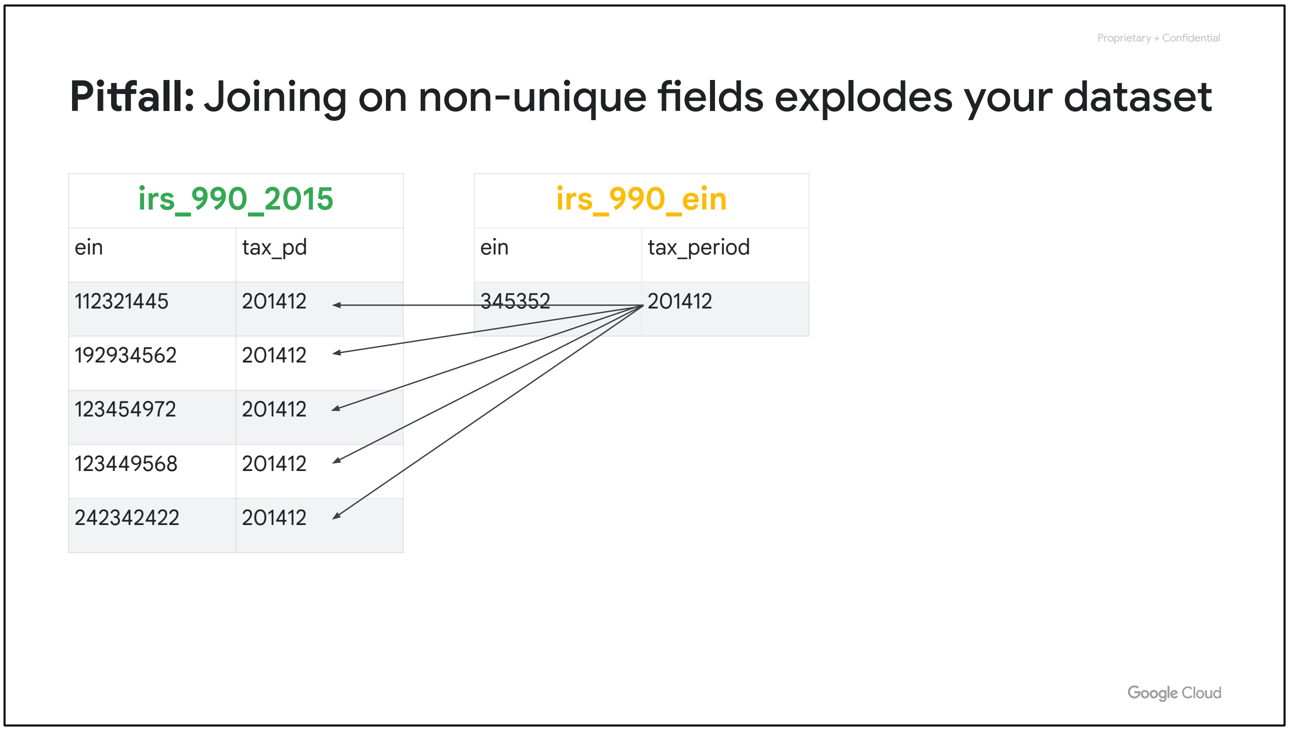


New Query

```
SELECT filing.ein, tax_pd
FROM `bigquery-public-data.irs_990.irs_990_2015` filing
LEFT JOIN `bigquery-public-data.irs_990.irs_990_ein` org
ON filing.tax_pd = org.tax_period
```

Woah, what happened here?

Here, this is an IRS example, where we, instead of joining on the unique identifier, which is the EIN, we joined on the tax period. And as many common tax period values that could be shared across many of those EINs. So let's work through what that actually looks like conceptually.



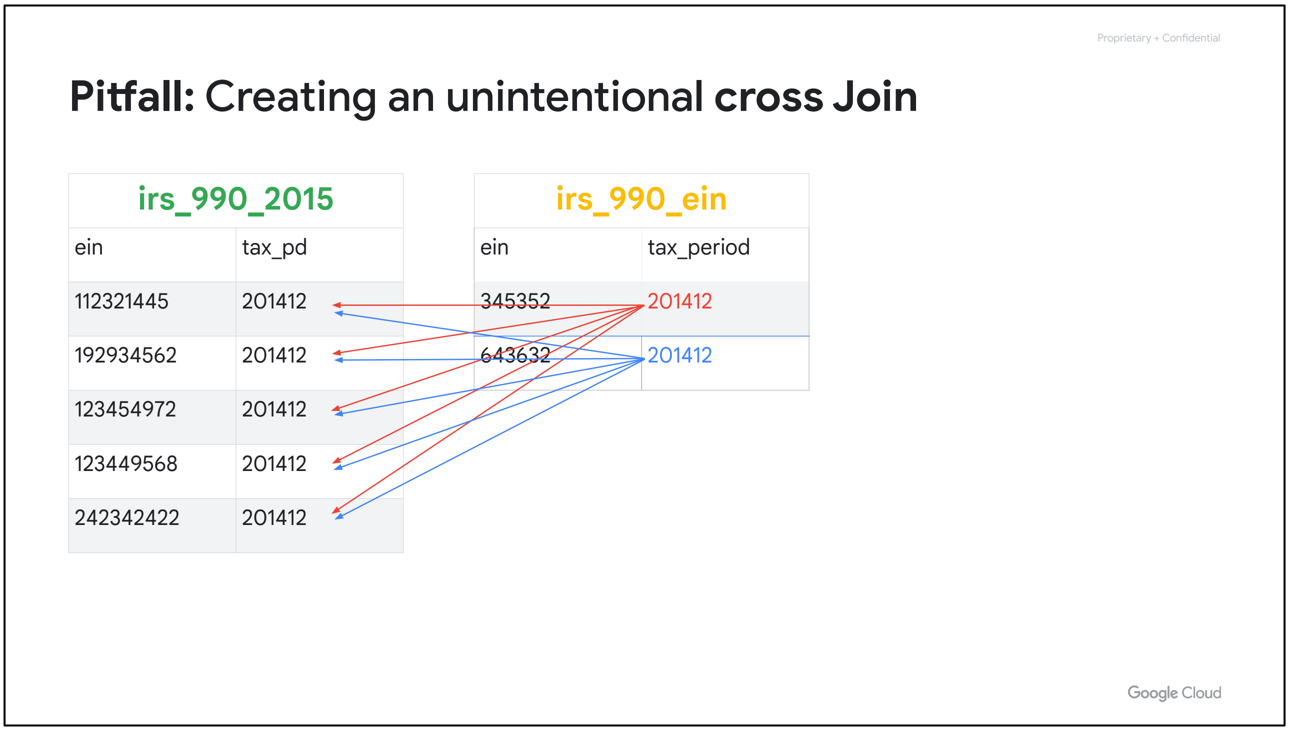
Pitfall: Joining on non-unique fields explodes your dataset

irs_990_2015	
ein	tax_pd
112321445	201412
192934562	201412
123454972	201412
123449568	201412
242342422	201412

irs_990_ein	
ein	tax_period
345352	201412

So, here we have our two different tables. On the left-hand side, we have the 2015 filings and the right-hand side is the organization or charity details table, which just has a tax period and when they last filed.

So instead of joining in on EIN, you would normally expect your EINs to match up one for one. One EIN organizational details per one filing. But instead, if you incorrectly put the tax period as your joining key, what you've done here is basically said: All right, well, this tax period and this organizational details table on the right matches five records there on the left, so that's great. What we're going to be returning here is just this EIN number, the 345352, as an additional column for all of those five. So it's not that bad, and we just have one record here. This query doesn't make too much sense joining on tax period. But what happens if we join…



Pitfall: Creating an unintentional cross Join

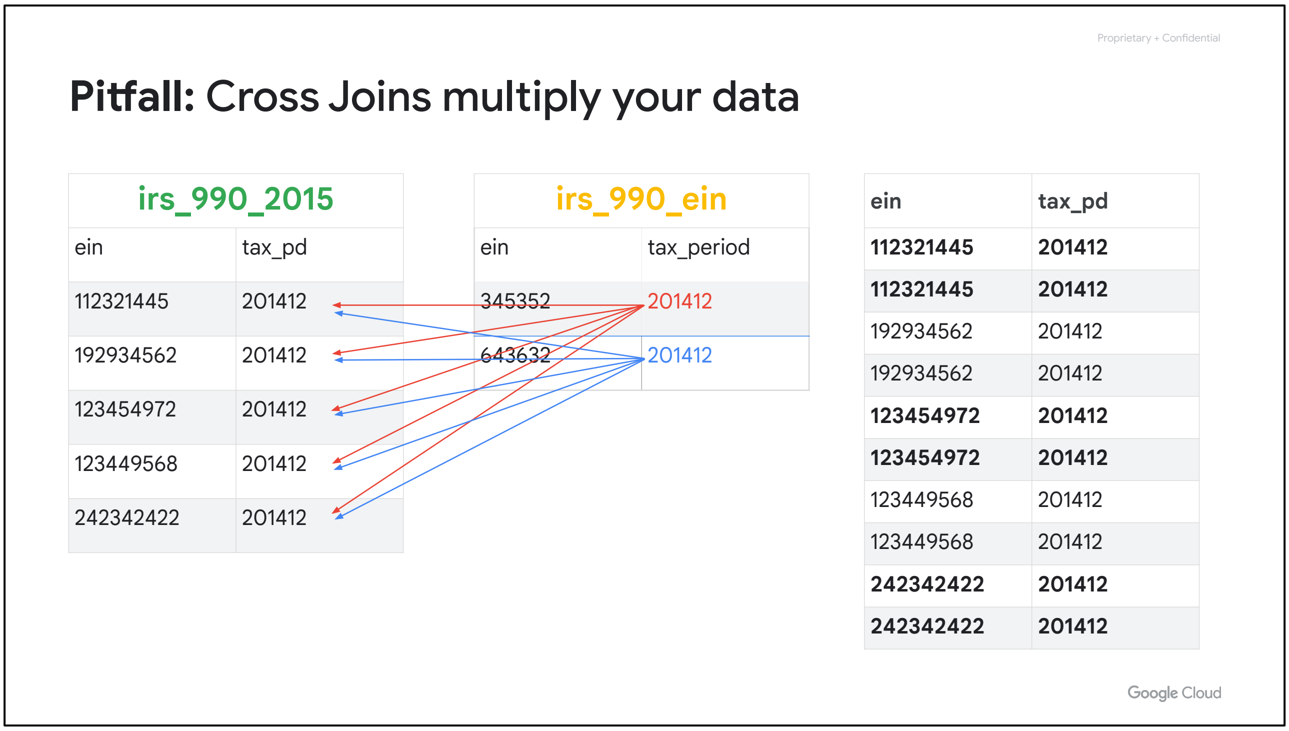
irs_990_2015

ein	tax_pd
112321445	201412
192934562	201412
123454972	201412
123449568	201412
242342422	201412

irs_990_ein

ein	tax_period
345352	201412
643632	201412

…on another EIN, that also has the same tax period of December 2014? Now, you see where it's starting to multiply, in a very bad way, the amount of output rows that we have. So this one also matches those five records.



Pitfall: Cross Joins multiply your data

irs_990_2015

ein	tax_pd
112321445	201412
192934562	201412
123454972	201412
123449568	201412
242342422	201412

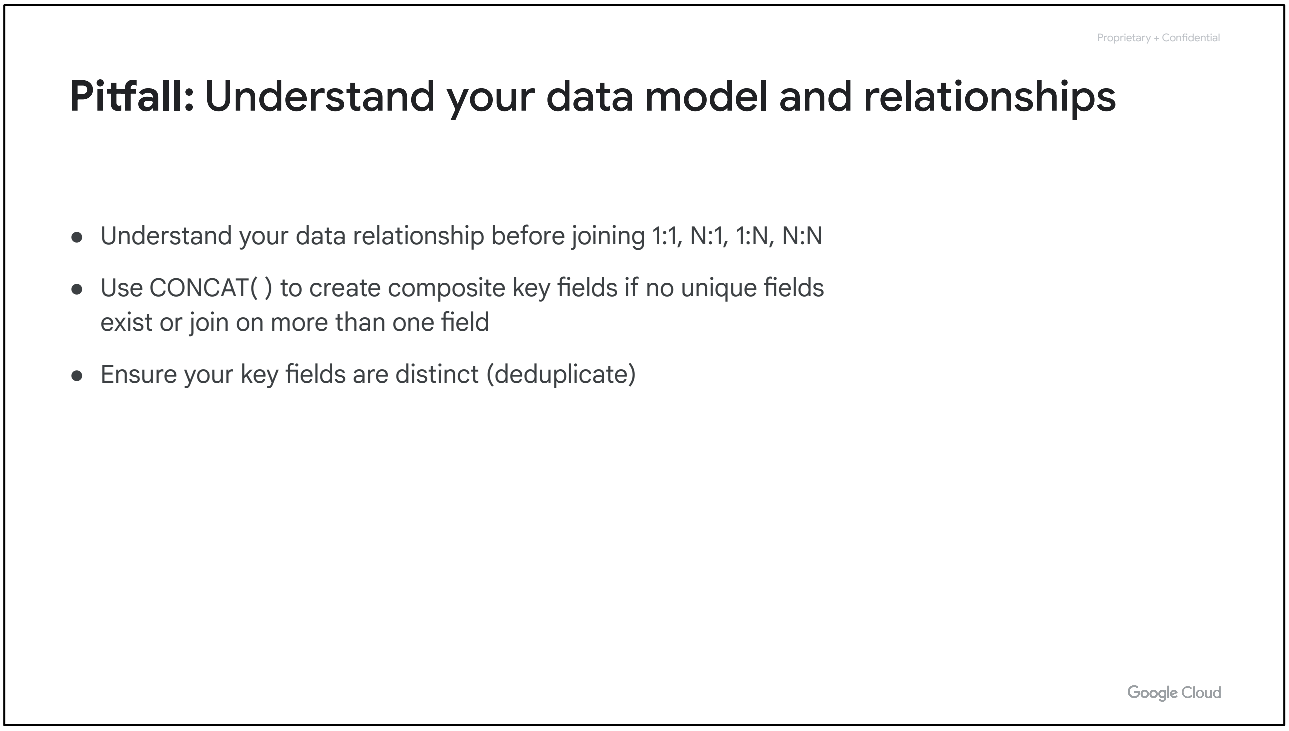
irs_990_ein

ein	tax_period
345352	201412
643632	201412

ein	tax_pd
112321445	**201412**
112321445	**201412**
192934562	201412
192934562	201412
123454972	**201412**
123454972	**201412**
123449568	201412
123449568	201412
242342422	**201412**
242342422	**201412**

Google Cloud

And what this looks like in your resulting dataset is again the cross-product or what's called an unintentional cross-join of your data. And again, this is a very small example where you just have two rows cross-joining against five, and you get the result there. But you can imagine if you have this on a million-row dataset. You could be potentially just have even a billion or a trillion rows accidentally outputted. And this is generally where BigQuery will run for more than 60 seconds or a couple of minutes. And then you realize, wait a minute, there's something that's going terribly wrong with our queries.



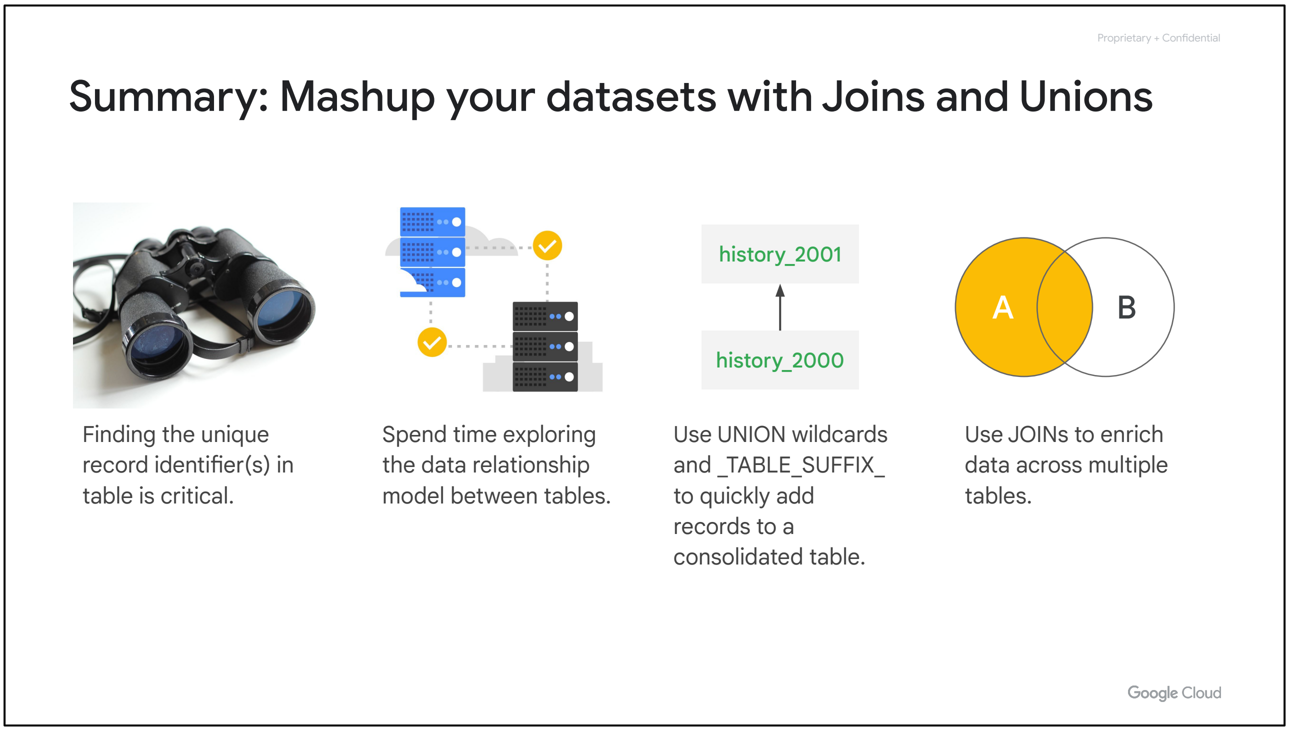
Pitfall: Understand your data model and relationships

- Understand your data relationship before joining 1:1, N:1, 1:N, N:N
- Use CONCAT( ) to create composite key fields if no unique fields exist or join on more than one field
- Ensure your key fields are distinct (deduplicate)

And again, the way to cure this is really understand what are the unique identifying fields in your dataset and what's the relationship between all of your different data tables. So you could have 1:1. For example, you could have one charity that corresponds to one tax year of filing for 2015. Or you could have many to one, that's what the N represents. Many to one could potentially represent if you had multiple years of tax filings, you could have many tax filings per one organization or you could have the opposite one to many.

Or you could have that many to many scenario, where you might need to create it through concatenation, or another method, a unique identifier key that you have.

And this is why the very first thing that we taught in the first course here is practicing counts and count distinct to see which fields in your dataset are the actual unique identifiers for those rows.



Summary: Mashup your datasets with Joins and Unions

history_2001
history_2000
A
B

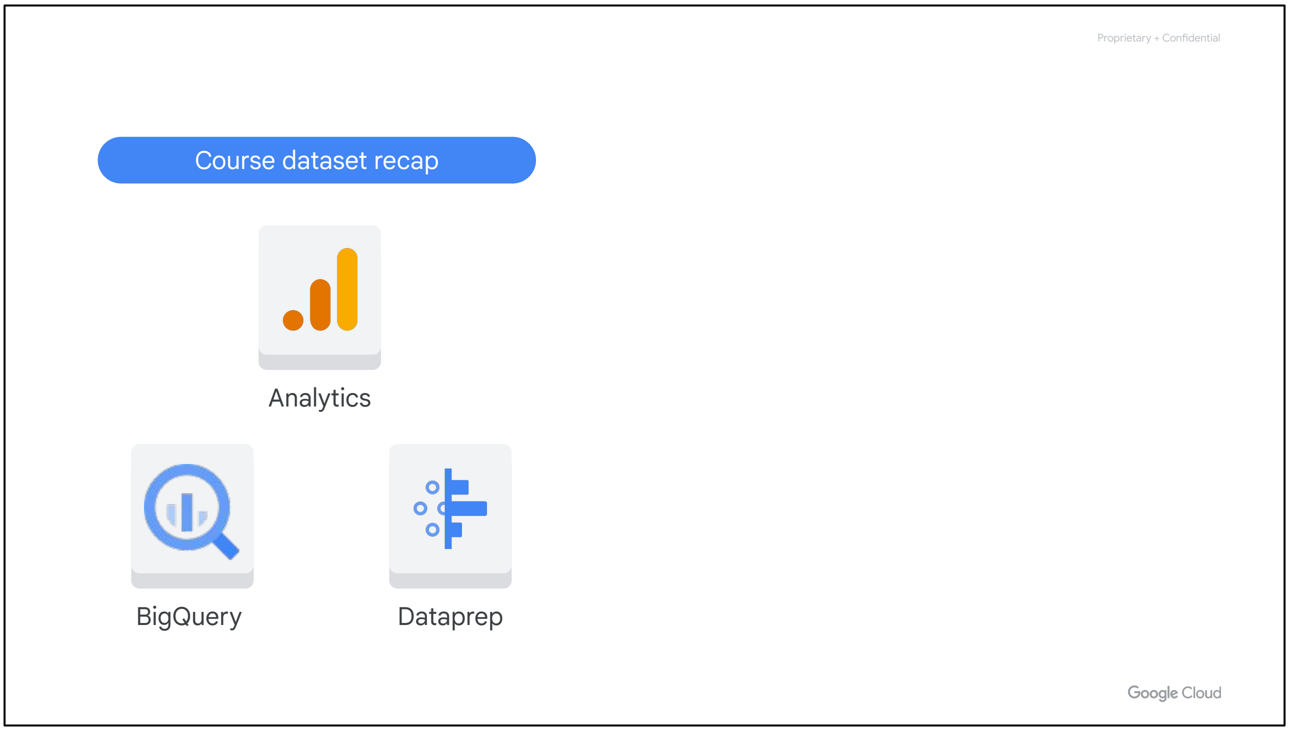
Finding the unique record identifier(s) in table is critical.

Spend time exploring the data relationship model between tables.

Use UNION wildcards and _TABLE_SUFFIX_ to quickly add records to a consolidated table.

Use JOINs to enrich data across multiple tables.

Understanding when and how to use Joins and UNIONS in SQL is a concept that's easy to pick up, but honestly takes a while to truly master. The best advice I can give you when starting out is to really understand how your data tables are supposed to be related to each other, like customers to orders, supplier to inventory. But being able to back that up by verifying those relationships through SQL. Remember, all data's dirty data and it's your job to investigate it and interrogate it before it potentially pollutes your larger dataset with JOINS and UNIONS. And once you understand the relationships between your tables, use UNIONS to append records into a consolidated table and Join to enrich your data with other data sources. Let's practice these concepts and pitfalls in our next lab.



Course dataset recap

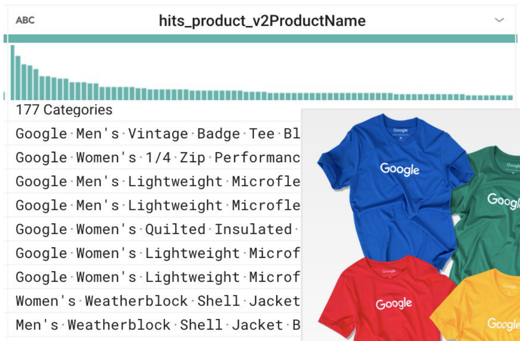
Analytics

BigQuery

Dataprep

If you've taken the first course in this course series and exploring your data with BigQuery, you're already familiar with the e-commerce course dataset that we've been using so far. If you skip the first course, here's a brief recap of the dataset that we'll be working with.

Real-world ecommerce dataset

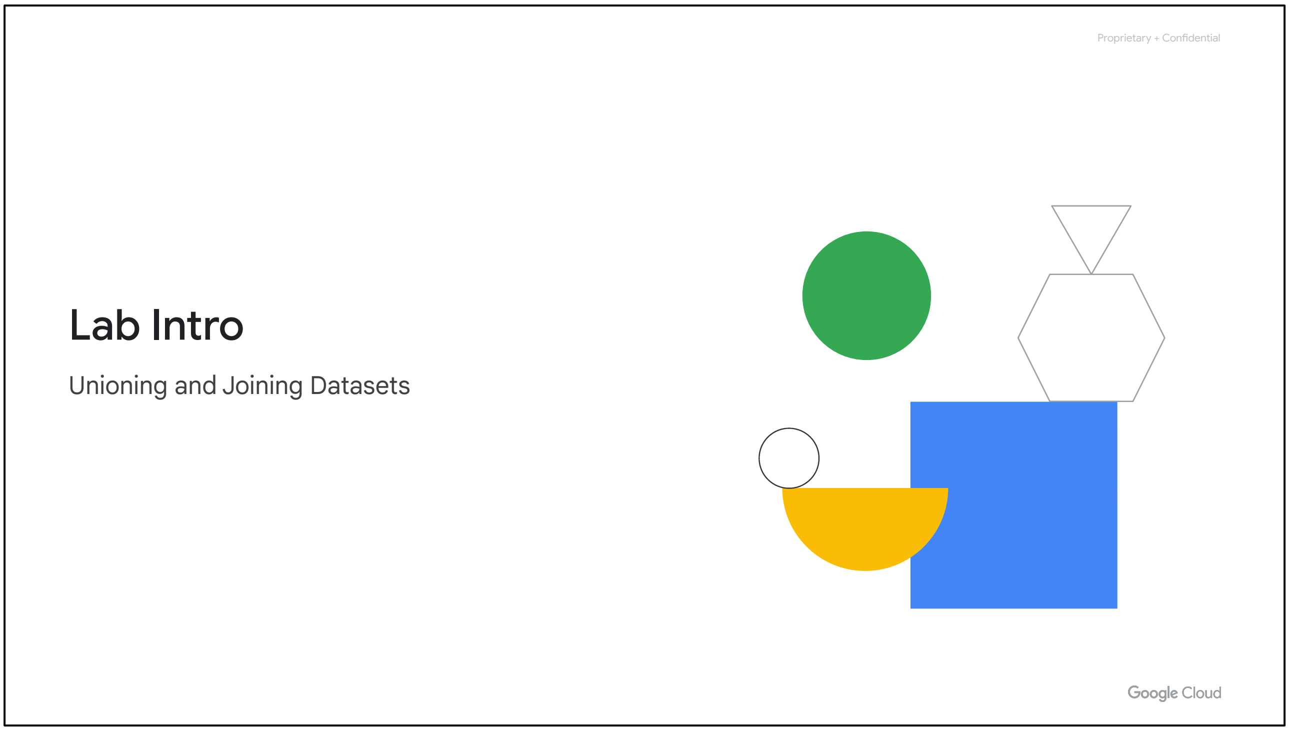


ABC hits_product_v2ProductName

177 Categories

```
Google Men's Vintage Badge Tee Bl
Google Women's 1/4 Zip Performanc
Google Men's Lightweight Microfle
Google Men's Lightweight Microfle
Google Women's Quilted Insulated
Google Women's Lightweight Microf
Google Women's Lightweight Microf
Women's Weatherblock Shell Jacket
Men's Weatherblock Shell Jacket B
```

The dataset that you use for your hands-on labs is a year of Google Analytics records behind the Google merchandise store, which sells Google brand and merchandise like sunglasses and even t-shirts. That's over a million site hits and transaction records, to include insights like, what users were close to transacting but never completed? What top keywords are high-value customers using to reach the site? And what top channels are customers going through to reach your site?



Lab Intro

Unioning and Joining Datasets

In this next lab, we're going to focus entirely on bringing data together from multiple different data tables through UNIONS and JOINS.

Specifically what we'll tackle is merging together historical annual IRS filings and join them against organizational lookup details so you can get things like the business's name. And the end result is it can be a consolidated reporting table that we can query off of.