Bertelsmann Tech Scholarship - Data Track

27.4 Text + Quiz: Your First JOIN

A query using a JOIN statement.

- SELECT clause indicates column(s) of data will see in the output
- FROM clause indicates 1st table we're pulling data.
- JOIN indicates 2nd table. JOINing the matching PK-FK links from 2 tables.
- ON clause specifies col on which you'd like to merge 2 tables together.

```
SELECT orders.*
 FROM orders
 JOIN accounts
 ON orders.account_id = accounts.id;
```

table name / column

-->>> Notice > we pull data from 2 tables: orders - accounts
but SELECT only pulling data from orders table, reference columns from orders table.

ON statement holds 2 columns that get linked across 2 tables, will be the focus in the next concepts.

--->>> If we wanted to only pull individual elements from either the orders or accounts table, using the exact same information in the FROM and ON statements. However, SELECT, will need to know how to specify tables and columns in the SELECT:

This query only pulls two columns

```
SELECT accounts.name,
       orders.occurred_at
 FROM orders
 JOIN accounts
 ON orders.account_id = accounts.id;
```

This query pulls all columns from both accounts + orders table.

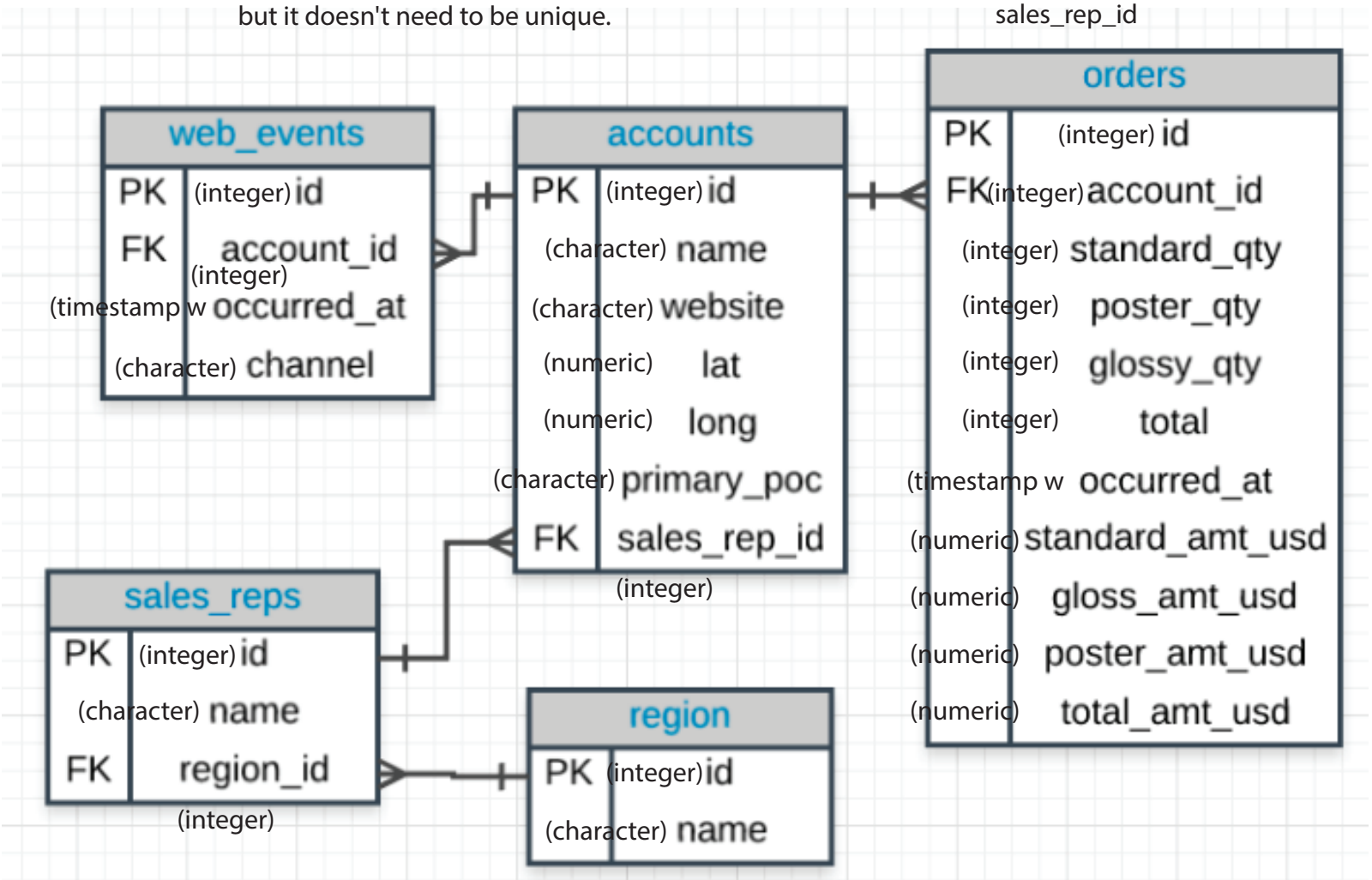
```
SELECT *
 FROM orders
 JOIN accounts
 ON orders.account_id = accounts.id;
```

27.6 Text: ERD Reminder = Entity Relationship Diagrams

PK = primary key = a col has a unique value for every row ~ id PK is usually the first column in our table.

Foreign Key - FK = a column in one table that is a primary key in a different table >> the foreign key is when the primary key appears in another table, but it doesn't need to be unique.

region_id
account_id
sales_rep_id



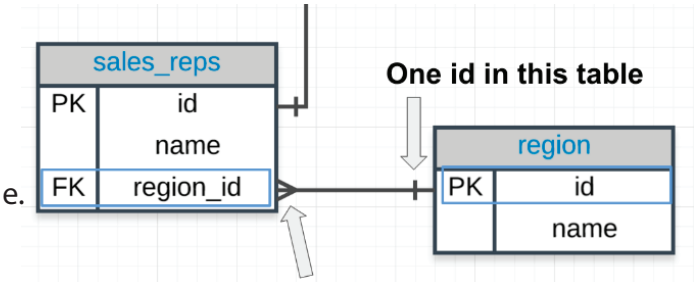
Parch & Posey database

Bertelsmann Tech Scholarship - Data Track

27.7 Text: Primary PK and Foreign Keys FK

PK = primary key = a col has a unique value for every row ~ id, common 1st col in each of our tables, in most dbs.

FK = a column in one table that is a primary key in a different table. FK linked to PK of another table.



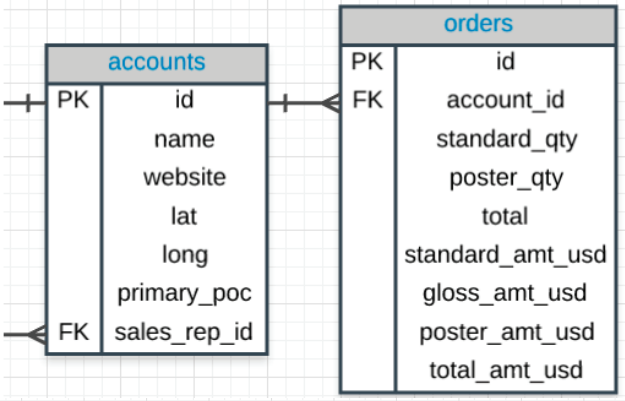
primary-foreign key link that connects 2 tables.
crow's foot shows FK can appear in many rows in table
PK single line shows id appears only once per row in table

27.9 Text + Quiz: JOIN Revisited

```
SELECT orders.*
FROM orders
JOIN accounts
ON orders.account_id = accounts.id;
```

- SQL query want to join 2 table - 1 in FROM and the other in JOIN Then in ON, ALWAYs have PK = FK;
- The actual ordering of which table name goes first in this statement doesn't matter so much. So, we could also write

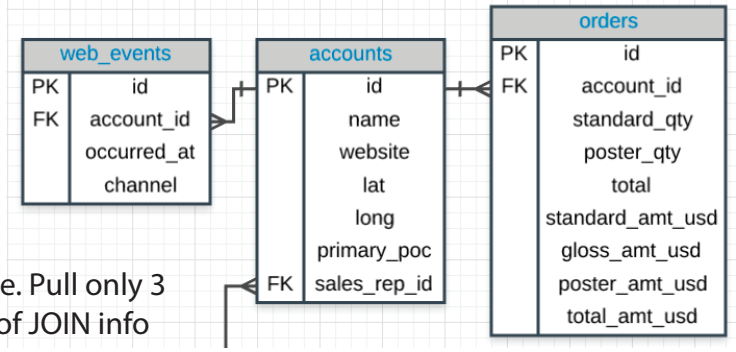
ON accounts.id = orders.account_id;



- JOIN More than Two Tables

This same logic can actually assist in joining more than two tables together. To join all 3 tables, use the same logic.

```
SELECT *
FROM web_events
JOIN accounts
ON web_events.account_id = accounts.id
JOIN orders
ON accounts.id = orders.account_id
```



- To pull specific cols, SELECT need to specify table + col name. Pull only 3 cols in these 3 tables by changing SELECT, but maintain rest of JOIN info

SELECT web_events.channel, accounts.name, orders.total

27.10 Video: Alias

- When JOIN tables, nice to give table an alias = 1st letter of table name.

```
SELECT orders.*
  FROM orders
  JOIN accounts
  ON orders.account_id = accounts.id;
```

=

```
SELECT o.*, a.*
  FROM orders o
  JOIN accounts a
  ON o.account_id = a.id;
```

27.11 Quiz: JOIN Questions Part I

```
SELECT a.primary_poc, w.occurred_at, w.channel, a.name
FROM web_events w
JOIN accounts a
ON w.account_id = a.id
WHERE a.name = 'Walmart';
```

```
SELECT r.name region, s.name rep, a.name account
FROM sales_reps s
JOIN region r
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
ORDER BY a.name;
```

```
SELECT r.name region, a.name account,
    o.total_amt_usd/(o.total + 0.01) unit_price
FROM region r
JOIN sales_reps s
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
JOIN orders o
ON o.account_id = a.id;
```

Bertelsmann Tech Scholarship - Data Track

INNER JOIN

27.13 Video: Motivation for Other JOINs

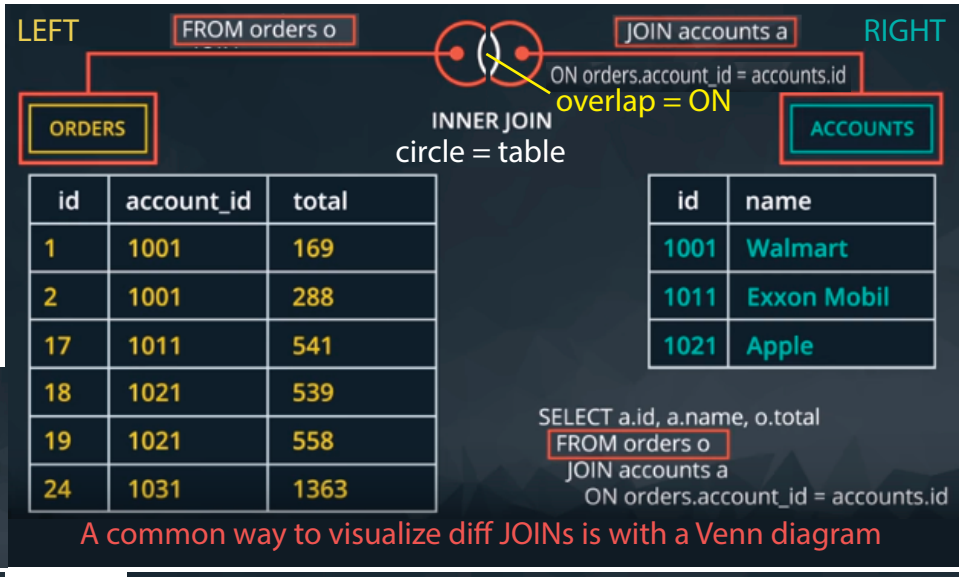
INNER JOIN = 1-to-1 and 1-to-many relationships when we introduced PKs and FKs. Notice, traditional dbs do not allow for many-to-many relationships, as these break the schema down pretty quickly.

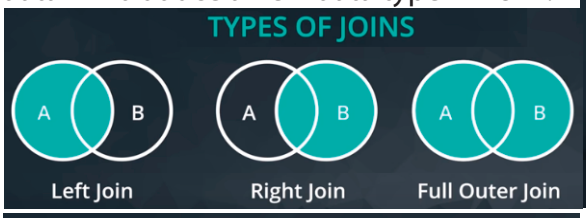
The types of relationships that exist in a db matter less to analysts, but you do need to understand why you would perform diff types of JOINs, & what data you are pulling from db. These ideas 'll be expanded upon in next concepts.

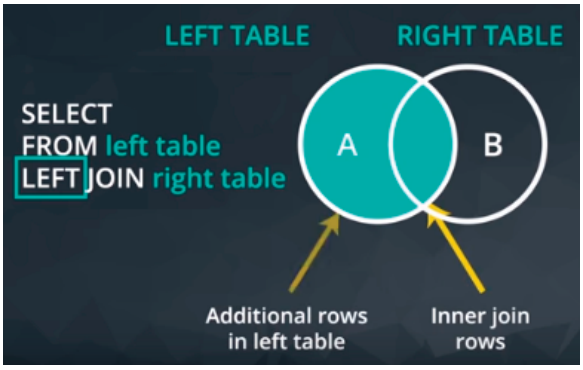
27.14 LEFT and RIGHT JOINs

OUTER JOIN = JOIN data depending on the question we are asking.

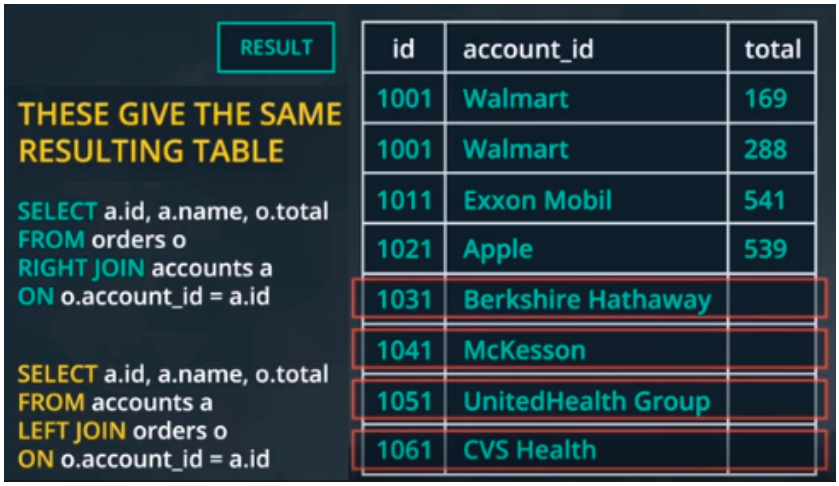
Notice each of these new JOIN states pulls all the same rows as an INNER JOIN, which you saw by just use JOIN, but they also potentially pull some additional rows. If there is not matching info in JOINed table, then will have cols w/ empty cells, without data = introduce a new data type = NULL.







Will use LEFT JOIN on this course



27.15 Text: Other JOIN Notes

INNER JOIN = pulled rows only if they exist as a match across 2 tables.

OUTER JOIN = allow us to pull rows that might only exist in 1 of the 2 tables, intro a new data type NULL.

LEFT OUTER JOIN = LEFT JOIN = obtain all rows of INNER JOIN, & additional rows from the table in the FROM.

The last type of join = full outer join. This will return the inner join result set, as well as any unmatched rows from either of 2 tables being joined. FULL OUTER JOIN, same as OUTER JOIN.

examples of outer join = https://www.w3resource.com/sql/joins/perform-a-full-outer-join.php

5	Sri Lanka	NULL
6	Brazil	NULL

If we were to flip the tables, we would get same exact result as the JOIN statement:

27.18 Video: JOINs and Filtering

- When db executes query, it executes JOIN and ON clause first
  => building new result set
  => then filtered using the WHERE clause.
- ON clause of an INNER JOIN will produce the same result as keeping it in the WHERE clause.

LOGIC IN THE ON CLAUSE REDUCES THE ROWS BEFORE COMBINING THE TABLES

LOGIC IN THE WHERE CLAUSE OCCURS AFTER THE JOIN OCCURS

```
SELECT orders.*,
       accounts.*
  FROM demo.orders
  LEFT JOIN demo.accounts
    ON orders.account_id = accounts.id
 WHERE accounts.sales_rep_id = 321500
```

AND replace WHERE, as the continue of ON clause, will add more rows.

Bertelsmann Tech Scholarship - Data Track

27.19 Quiz: Last Check 27.20 Solutions: Last Check

SOLUTION 1

```
SELECT r.name region, s.name rep, a.name account
FROM sales_reps s
JOIN region r
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
WHERE r.name = 'Midwest'
ORDER BY a.name;
```

SOLUTION 2

```
SELECT r.name region, s.name rep, a.name account
FROM sales_reps s
JOIN region r
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
WHERE r.name = 'Midwest' AND s.name LIKE 'S%'
ORDER BY a.name;
```

SOLUTION 3

```
SELECT r.name region, s.name rep, a.name account
FROM sales_reps s
JOIN region r
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
WHERE r.name = 'Midwest' AND s.name LIKE 'K%'
ORDER BY a.name;
```

SOLUTION 4

```
SELECT r.name region, a.name account,
       o.total_amt_usd/(o.total + 0.01) unit_price
FROM region r
JOIN sales_reps s
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
JOIN orders o
ON o.account_id = a.id
WHERE o.standard_qty > 100;
```

SOLUTION 5

```
SELECT r.name region, a.name account,
       o.total_amt_usd/(o.total + 0.01) unit_price
FROM region r
JOIN sales_reps s
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
JOIN orders o
ON o.account_id = a.id
WHERE o.standard_qty > 100 AND o.poster_qty > 50
ORDER BY unit_price;
```

SOLUTION 6

```
ORDER BY unit_price DESC;
```

SOLUTION 7

```
SELECT DISTINCT a.name, w.channel
FROM accounts a
JOIN web_events w
ON a.id = w.account_id
WHERE a.id = '1001';
```

SOLUTION 8

```
SELECT DISTINCT o.occurred_at, a.name, o.total, o.total_amt_usd
FROM accounts a
JOIN orders o
ON a.id = o.account_id
WHERE o.occurred_at  BETWEEN '01-01-2015' AND '01-01-2016'
ORDER BY o.occurred_at DESC;
```

SELECT DISTINCT
to narrow down the results to
only the unique values.

27.21 Text: Recap & Looking Ahead

PK - are unique for every row in a table, generally 1st col in db (like id col for every table in our db).

FK - are primary key appearing in another table, which allows the rows to be non-unique.

Choosing the set up of data in our database is very important, but not usually the job of a data analyst. This process is known as Database Normalization.

JOINs = combine data from multiple tables, 3 JOIN statements most likely to use are:

- JOIN - an INNER JOIN that only pulls data that exists in both tables.
- LEFT JOIN - pulls all data exists in both tables, as well as all of the rows from table in FROM even if they do not exist in the JOIN statement.
- RIGHT JOIN - pulls all data exists in both tables, as well as all of the rows from the table in the JOIN even if they do not exist in the FROM statement.
- Advanced JOINs used in very specific use cases. UNION and UNION ALL, CROSS JOIN, and the tricky SELF JOIN.

Alias tables and cols using AS or not using it, allows to be more efficient in the number of characters you need to write, while at the same time you can assure that your column headings are informative of the data in your table.

- Next is aggregating data. Our SQL might still a bit disconnected from statistics and using Excel like platforms.

Aggregations allow to write SQL code more complex queries, which assist in answering questions like:

Which channel generated more revenue?
Which account had an order with the most items?
Which sales_rep had the most orders? or least orders? How many orders did they have?