Week 2: SQL JOINs and Data Aggregation

LittleDabase, Employee Self Join, and Parch and Posey Database

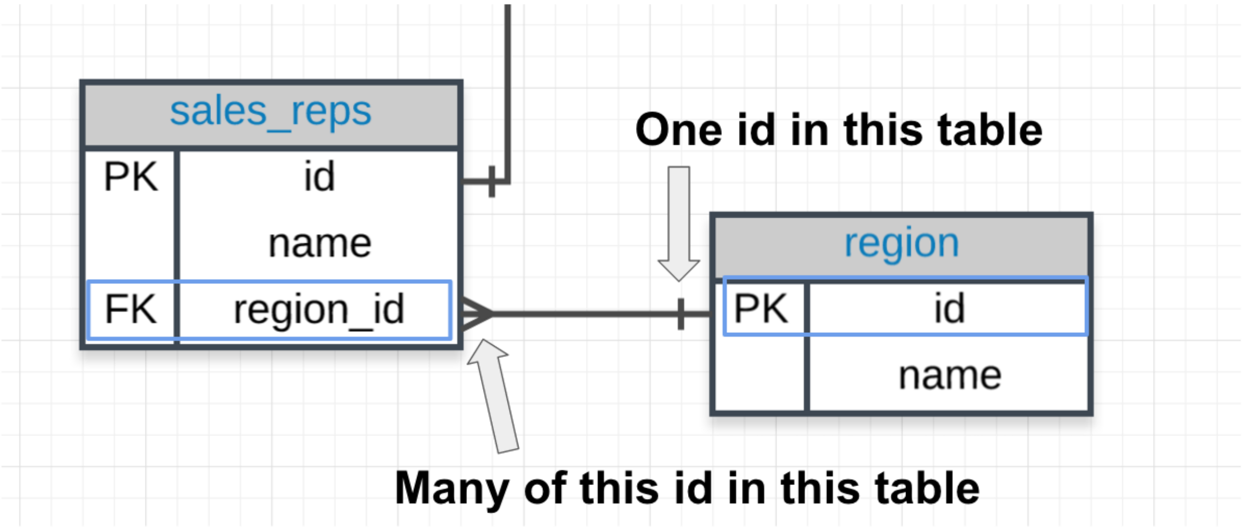
Primary and Foreign Keys

A *primary key* (PK) is a unique column in a particular table. In Parch and Posey, this is the first column in each of our tables. Here, those columns are all called `id`, but that doesn't necessarily have to be the name. It is common that the primary key is the first column in our tables in most databases.

A *foreign key* (FK) is when we see a primary key in another table. We can see these in the previous ERD the foreign keys are provided as: `region_id`, `account_id`, `sales_rep_id`. Each of these is linked to the primary key of another table. An example is shown in the image below:

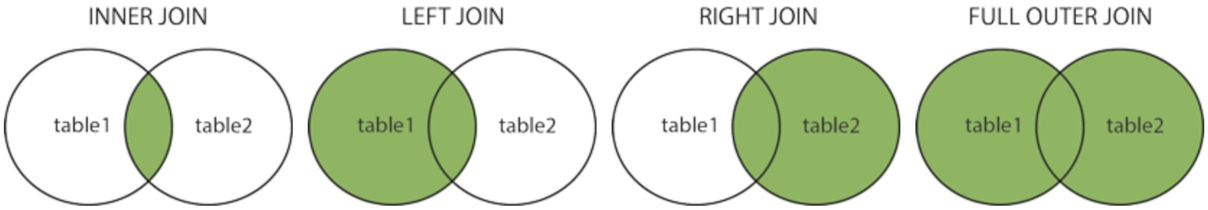
In the following figure, you can see:

- The region_id is the foreign key.
- The region_id is linked to id - this is the primary-foreign key link that connects these two tables.
- The "crow's" foot shows that the FK can actually appear in many rows in the sales_reps table.
- While the single line is telling us that the PK shows that id appears only once per row in this table.



['h']

Questions



['h']

Little Database

The example is based on W3resource.com.

1. Create a PostgreSQL database called `littleDatabase` in pgAdmin.
2. Create two tables in `littleDatabase` called: `table_a` and `table_b`. Table A has two columns (id = [1,2,4], A = [‘m’, ‘n’, ‘o’]) and Table B has two columns (id = [2,3,4], A = [‘p’, ‘q’, ‘r’]).

Table 1: Table A

id	a
1	m
2	n
4	o

Table 2: Table B

id	b
2	p
3	q
5	r

3. Apply inner joins, outer joins (left, right, and full), natural join, cross join, and self join.

Inner Joins

The INNER JOIN selects all rows from both participating tables as long as there is a match between the columns. An SQL INNER JOIN is same as JOIN clause, combining rows from two or more tables.

```
select *
from table_a
JOIN table_b
on table_a.id = table_b.id;
```

Table 3: Inner Join

id	a	id	b
2	n	2	p

Left OUTER JOIN (LEFT JOIN)

The SQL LEFT JOIN, joins two tables and fetches rows based on a condition, which are matching in both the tables.The unmatched rows will also be available from the table before the JOIN clause.

```
select *
from table_a
LEFT JOIN table_b
on table_a.id = table_b.id;
```

Table 4: Left Join

id	a	id	b
1	m	NA	NA
2	n	2	p
4	o	NA	NA

RIGHT OUTER JOIN (Right JOIN)

The SQL RIGHT JOIN, joins two tables and fetches rows based on a condition, which are matching in both the tables. The unmatched rows will also be available from the table written after the JOIN clause.

```sql
select *
from table_a
RIGHT JOIN table_b
on table_a.id = table_b.id;
```

Table 5: Right Join

id	a	id	b
2	n	2	p
NA	NA	3	q
NA	NA	5	r

FULL OUTER JOIN (Full JOIN)

The FULL JOIN Combines the results of both left and right outer joins. It returns all matched or unmatched rows, and includes tables on both sides of the join clause.

```sql
select *
from table_a
FULL JOIN table_b
on table_a.id = table_b.id;
```

Table 6: Full Join

id	a	id	b
1	m	NA	NA
2	n	2	p
4	o	NA	NA
NA	NA	5	r
NA	NA	3	q

NATURAL JOIN

The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with same name of associate tables will appear once only. The associated tables have one or more pairs of identically named columns. The columns must be the same data type. Don't use ON clause in a natural join.

```
select *
from table_a
NATURAL JOIN table_b;
```

Table 7: Natural Join

id	a	b
2	n	p

CROSS JOIN

The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table, if no WHERE clause is used along with CROSS JOIN. This kind of result is called as Cartesian Product. If, WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

```
select *
from table_a
CROSS JOIN table_b;
```

Table 8: Cross Join

id	a	id	b
1	m	2	p
1	m	3	q
1	m	5	r
2	n	2	p
2	n	3	q
2	n	5	r
4	o	2	p
4	o	3	q
4	o	5	r

SELF JOINS

A self join is a join in which a table is joined with itself (Unary relationships), specially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table. The self join can be viewed as a join of two copies of the same table.

```
select *
from table_a as x
JOIN table_a as y
ON x.id = y.id;
```

Table 9: Self Join

id	a	id	a
1	m	1	m
2	n	2	n
4	o	4	o

Employee Database (SELF JOIN EXAMPLE)

This example is based on w3resources.com

4. Create a PostgreSQL database called `employee_self_join` in pgAdmin.
5. Create the `employee` table using the `employee_self_join.sql` file.
6. What is the organization hierarchy structure of the company? i.e. List all the employees who work for each manager.

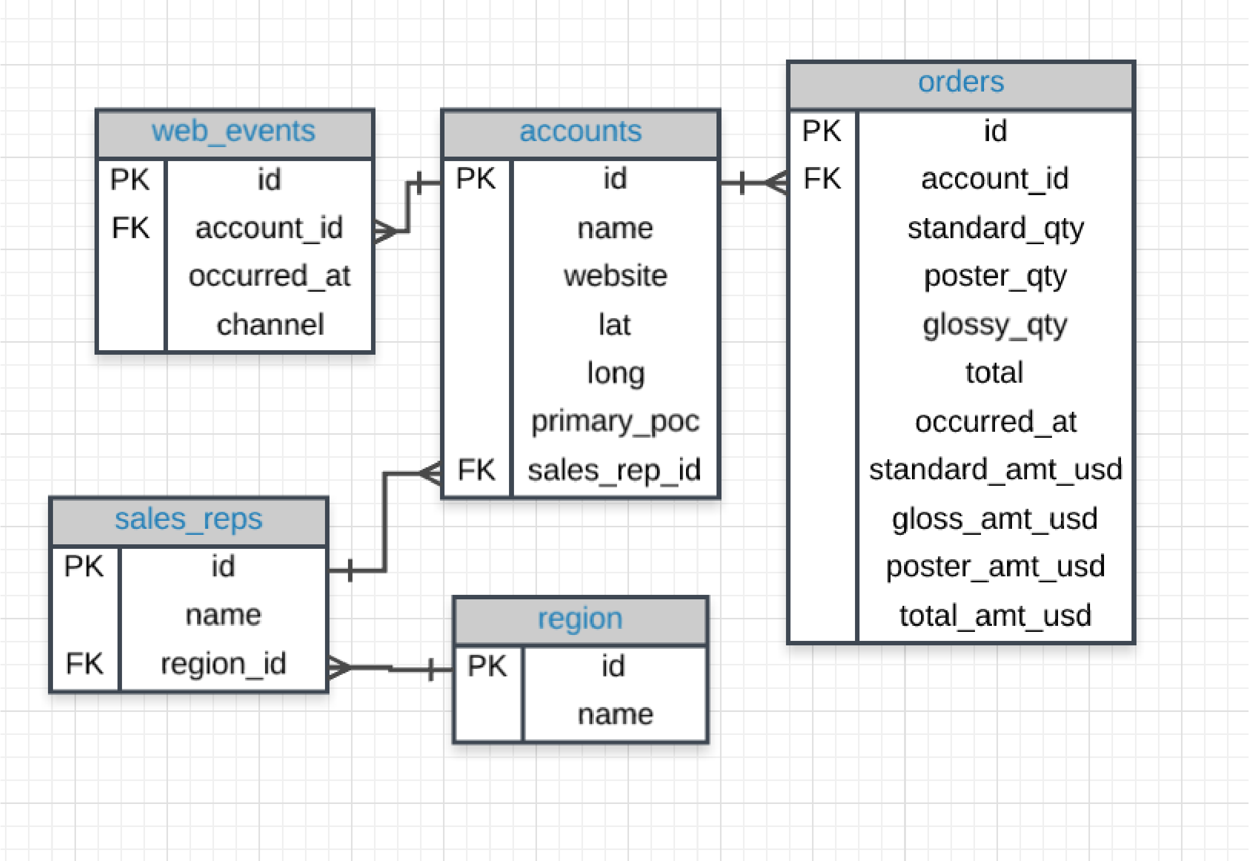
```
select concat(e.first_name, ' ', e.last_name) as employee,
      concat(m.first_name,' ', m.last_name) as manager
from employee as e
join employee as m
on e.employee_id = m.manager_id;
```

Table 10: Self Join Employee Manager Example

employee	manager
Windy Hays	Ava Christensen
Windy Hays	Hassan Conner
Ava Christensen	Anna Reeves
Ava Christensen	Sau Norman
Hassan Conner	Kelsie Hays
Hassan Conner	Tory Goff
Hassan Conner	Salley Lester

Parch and Posey Database

This example is based on Mode Analytics.



['h']

7. Write a query to return all orders by Walmart. Return the order id, account name, as well as the total paper quantity for each order.

```sql
SELECT accounts.name, orders.id, orders.total
FROM orders
INNER JOIN accounts
ON orders.account_id = accounts.id
WHERE accounts.name = 'Walmart'
limit 5;
```

name	id	total
Walmart	1	169
Walmart	2	288
Walmart	3	132
Walmart	4	176
Walmart	5	165

Notice that we don't have to use the word INNER for inner joins as follows:

```sql
SELECT accounts.name, orders.id, orders.total
FROM orders
JOIN accounts
```

```
ON orders.account_id = accounts.id
WHERE accounts.name = 'Walmart'
limit 5;
```

In addition, we can give each table in our query an alias or a nickname. Frequently an alias is just the first letter of the table name:

```
SELECT a.name, o.id, o.total
FROM orders as o
JOIN accounts as a
ON o.account_id = a.id
WHERE a.name = 'Walmart'
limit 5;
```

We can also get rid of the key word `AS` when we give an alias:

```
SELECT a.name, o.id, o.total
FROM orders o
JOIN accounts a
ON o.account_id = a.id
WHERE a.name = 'Walmart'
limit 5;
```

name	id	total
Walmart	1	169
Walmart	2	288
Walmart	3	132
Walmart	4	176
Walmart	5	165

In this query, we really don't need the WHERE clause to filter the results to have the account name to be "Walmart". We can do that by filtering in the ON clause. By changing WHERE to AND, we're moving this logical statement to become part of the ON clause. This effectively pre-filters the right table to include only rows with account name "Walmart" before the join is executed. In other words, it's like a WHERE clause that applies before the join rather than after.

```
SELECT a.name, o.id, o.total
FROM orders o
JOIN accounts a
ON o.account_id = a.id AND a.name = 'Walmart'
limit 5;
```

name	id	total
Walmart	1	169
Walmart	2	288
Walmart	3	132
Walmart	4	176
Walmart	5	165

8. **YOUR TURN** Create a table that has the different channels used by account id 1001. Your final table should have only 2 columns: account name and the different channels.

```
SELECT distinct a.name as Account_Name, w.channel as Channel
FROM accounts as a
JOIN web_events as w
ON a.id = w.account_id AND a.id = 1001
limit 5
```

account_name	channel
Walmart	adwords
Walmart	banner
Walmart	direct
Walmart	facebook
Walmart	organic

9. **YOUR TURN** Create a table that has the orders that occurred in 2015 (sorted by date). Your final table should have 4 columns: occurred_at, account name, order total, and order total_amt_usd.

```
SELECT a.name as Account_Name, o.total AS Total, o.total_amt_usd AS Total_Amount, o.occurred_at AS Date
FROM orders as o
JOIN accounts as a
ON o.account_id = a.id
WHERE occurred_at between '01-01-2015' and '01-01-2016'
ORDER BY occurred_at
limit 5
```

account_name	total	total_amount	date
FirstEnergy	38	284.62	2015-01-01 05:33:43
FirstEnergy	529	2737.29	2015-01-01 05:53:44
New York Life Insurance	517	2644.89	2015-01-01 11:17:47
Travelers Cos.	1320	8770.30	2015-01-01 14:40:53
Travelers Cos.	195	1233.37	2015-01-01 14:42:53

10. Create a table that has the `region` for each `sales_rep` along with their associated accounts. Your final table should include three columns: the region name, the sales rep name, and the account name.

```
SELECT r.name Region, s.name Sales_Rep, a.name Account_Name
FROM sales_reps AS s
JOIN region AS r
ON   r.id = s.region_id
JOIN accounts AS a
ON  s.id= a.sales_rep_id
limit 5;
```

region	sales_rep	account_name
Northeast	Samuel Racine	Walmart
Northeast	Samuel Racine	Express Scripts Holding
Northeast	Samuel Racine	Freddie Mac
Northeast	Samuel Racine	Ingram Micro
Northeast	Samuel Racine	American Airlines Group

11. **YOUR TURN** Create a table that provides the region for each sales_rep along with their associated accounts. This time only for the Midwest region. Your final table should include three columns: the region name, the sales rep name, and the account name.

```
SELECT r.name Region, s.name Sales_Rep, a.name Account_Name
FROM sales_reps AS s
JOIN region AS r
ON   r.id = s.region_id
JOIN accounts AS a
ON  s.id= a.sales_rep_id
WHERE r.name = 'Midwest'
limit 5;
```

region	sales_rep	account_name
Midwest	Sherlene Wetherington	Progressive
Midwest	Sherlene Wetherington	U.S. Bancorp
Midwest	Sherlene Wetherington	Community Health Systems
Midwest	Sherlene Wetherington	Time Warner Cable
Midwest	Sherlene Wetherington	Rite Aid

12. **YOUR TURN** Who was the primary contact associated with the earliest web_event?

```
SELECT a.primary_poc
FROM web_events w
JOIN accounts a
ON a.id = w.account_id
ORDER BY w.occurred_at
LIMIT 1;
```

primary_poc
Leana Hawker

13. **YOUR TURN** Create a table that has the name for each region for every order, as well as the account name and the unit price they paid (`total_amt_usd/total`) for the order. Your final table should have 3 columns: region name, account name, and unit price. A few accounts have 0 for total, so you might have to divide by (total + 0.0001) to assure not dividing by zero.

```
SELECT r.name Region, a.name Account_Name, round(o.total_amt_usd/(o.total + 0.0001), 2) Unit_Price
FROM region AS r
JOIN sales_reps AS s
ON r.id = s.region_id
JOIN accounts AS a
ON s.id = a.sales_rep_id
JOIN orders AS o
ON a.id = o.account_id
limit 5
```

region	account_name	unit_price
Northeast	Walmart	5.76
Northeast	Walmart	5.97
Northeast	Walmart	5.88
Northeast	Walmart	5.44
Northeast	Walmart	5.96

Data Aggregation

14. Count the number of rows in the `accounts` table.

```
SELECT COUNT(*)
FROM accounts
```

count
351

15. Find the total amount of `poster_qty` paper ordered in the `orders` table.

```
SELECT SUM(poster_qty) AS total_poster_sales
FROM orders;
```

total_poster_sales
723646

16. What is the min and max order quantity for each poster papers in the database?

```
SELECT MIN(poster_qty) as poster_min, MAX(poster_qty) poster_max
FROM orders;
```

poster_min	poster_max
0	28262

17. When was the earliest order ever placed?

```
SELECT MIN(occurred_at)
FROM orders;
```

min
2013-12-04 04:22:44

18. Find the mean (AVERAGE) amount spent per order on each paper type.

```
SELECT AVG(standard_amt_usd) mean_standard, AVG(gloss_amt_usd) mean_gloss,
           AVG(poster_amt_usd) mean_poster
FROM orders;
```

mean_standard	mean_gloss	mean_poster
1399.356	1098.547	850.1165

19. What is the MEDIAN `total_usd` spent on all orders?

```
SELECT *
FROM (SELECT total_amt_usd
      FROM orders
      ORDER BY total_amt_usd
      LIMIT 3457) AS Table1
ORDER BY total_amt_usd DESC
LIMIT 2;
```