Week 2: SQL JOINs

Parch and Posey Database

Contents

Recap: In the Parch & Posey database there are five tables:

- web_events
- accounts
- orders
- sales_reps
- region

Figure 1 shows the ERD (entity relationship diagram) for Parch and Posey.

Reference: These notes are based on material from Mode's Analytics as well as w3schools.

Introduction

Last session we worked with one table at a time. But the real power of SQL comes from working with data across multiple tables at once. The term relational database refers to the fact that tables within it relate to one another. They contain common identifiers that allow information from multiple tables to be easily

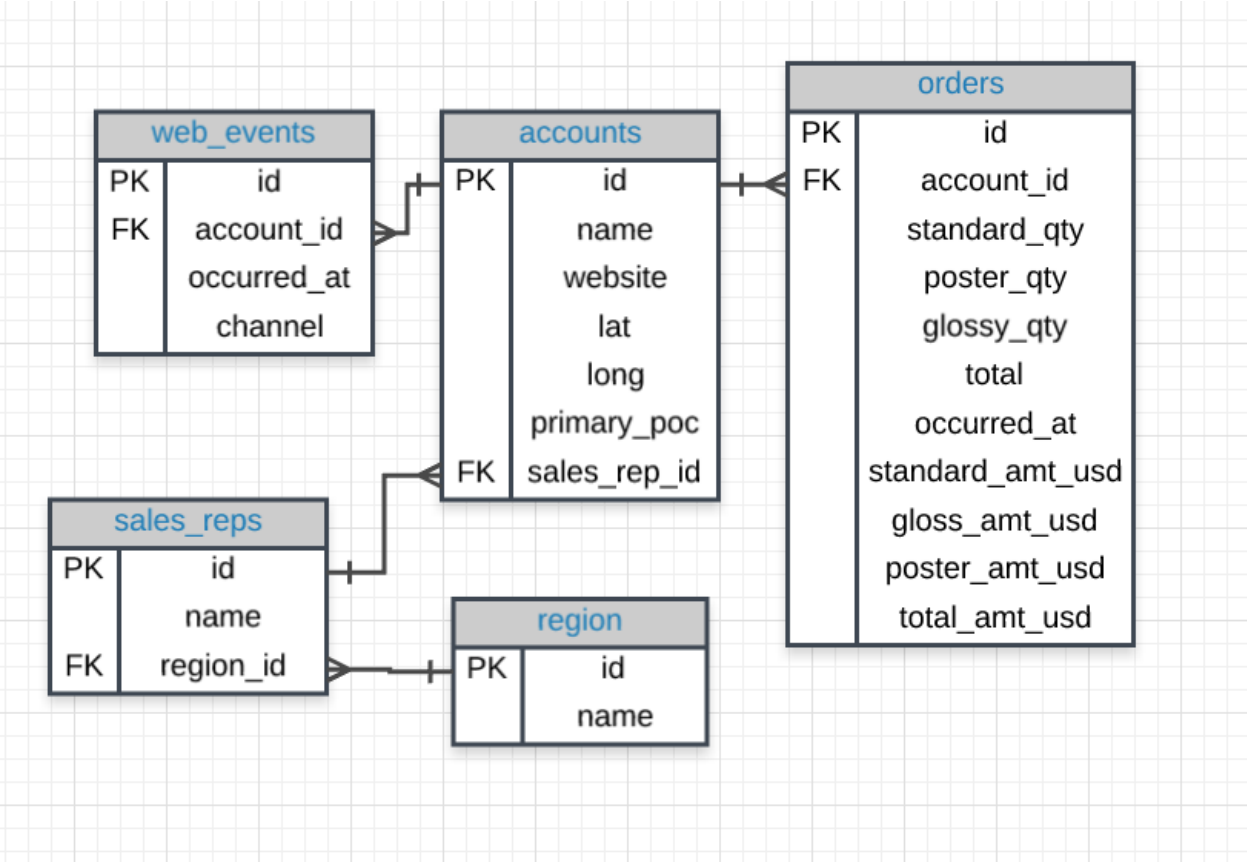

Figure 1: Parch and Posey ERD

combined. Keeping all of a company's data, from purchasing transactions, to employee job satisfaction, to inventory in a single Excel dataset doesn't make a ton of sense.

The table would hold a ton of information and it'd be hard to determine a row call and structure for so many different types of data. Databases give us the flexibility to keep things organized neatly in their own tables, so that they're easy to find and work with while also allowing us to combine tables as needed to solve problems that require several types of data.

In this lesson, we're going to take a look at how to leverage SQL to link tables together. You're about to see why SQL is one of the most popular environments for working with data as we learn how to write what are known as Joints.

To understand what JOINS are and why they're helpful, let's think about Parch & Posey's orders table.

Looking back at this table, you'll notice that none of the orders say the name of the customer (accounts). Instead, the table refers to customers by numerical values in the `account_id` column. We'll need to join another table in order to connect this data to names. But first, why isn't the customer's name in this table in the first place? There are a few reasons that someone might have made the decision to separate orders from the information about the customers placing those orders.

There are several reasons why relational databases are like this. Let's focus on two of the most important ones.

First, orders and accounts are different types of objects and will be easier to organize if kept separate.

Second, this multi-table structure allows queries to execute more quickly.

Let's look at each of these points more closely. The account and orders tables store fundamentally different types of objects. Parch & Posey probably only wants one account per company, and they wanted to be up-to-date with the latest information. Orders, on the other hand, are likely to stay the same once they are entered, and certainly, once they're filled. A given customer might have multiple orders, and rather than change a past order, Parch & Posey might just add a new one.

Because these objects operate differently, it makes sense for them to live in different tables. Another reason accounts and orders might be stored separately has to do with the speed at which databases can modify data. When you write a query, its execution speed depends on the amount of data you're asking the database to read and the number and type of calculations you're asking it to make. Imagine a world where the account names and addresses are added into the orders table. This means, the table would have six additional columns: one for account name, and five that make up the street address, city, state, zip code, and country. Let's say, a customer changes their address. In this world, the five address columns need to be updated retroactively for every single order. This means five updates times the number of orders that they've made. By contrast, keeping account details in a separate table makes this only a total of five updates. The larger the data set, the more that this matters.

Primary Keys 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 Figure 2, 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.

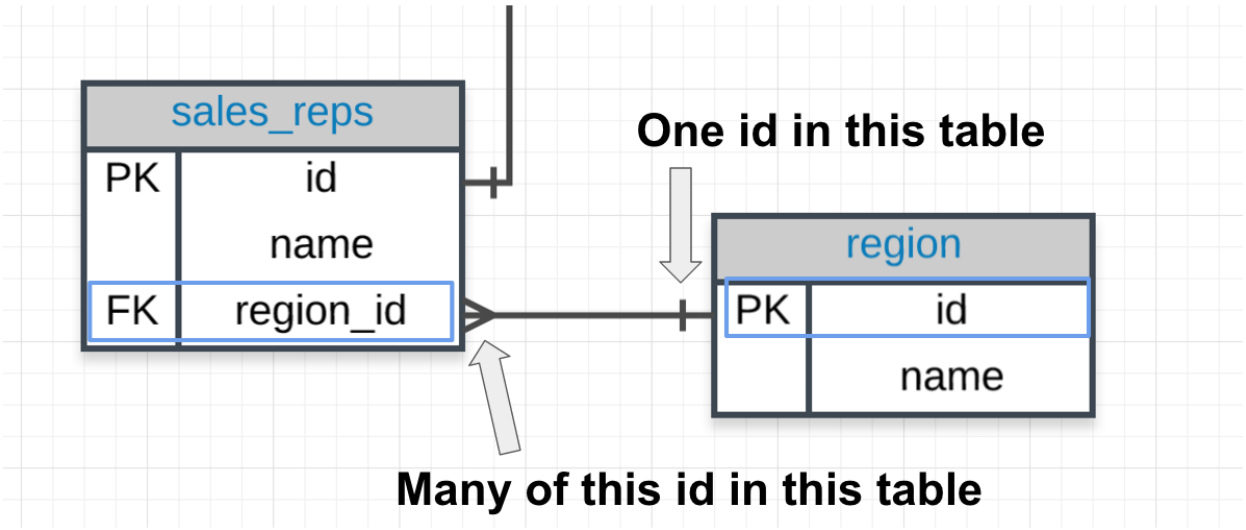

Figure 2: Primary and Foreign Keys

- While the single line is telling us that the PK shows that `id` appears only once per row in this table.

SQL JOINS

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

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

Figure 3 shows a Venn diagram explaining the different types of JOINS.

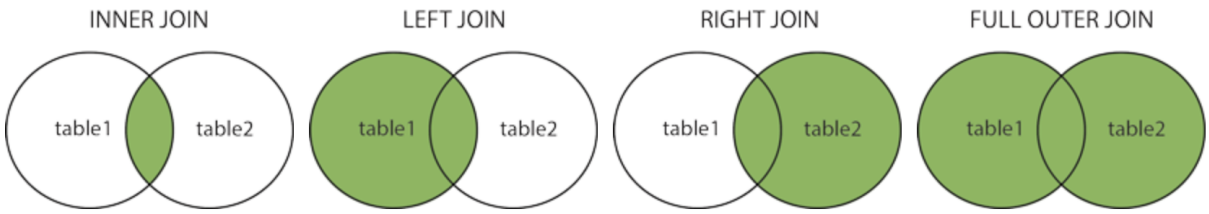

Figure 3: Joins

INNER JOINS (Two Tables)

The `INNER JOIN` keyword selects records that have matching values in both tables. The `INNER JOIN` selects all rows from both tables as long as there is a match between the columns. If there are records in one table that do not have matches in the other table, then these rows will not be shown.

1. 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.

In this query, we will join accounts to all orders. To do so, we start first by joining the `orders` table to `accounts` table using the `JOIN` clause. It identifies the table where the data that we want to join lives. In this case, account names live in the `accounts` table. Next, we need to specify the relationship between the two tables. This happens by writing a logical statement in the `ON` clause. You can think of the `ON` clause like a second `FROM` clause. Finally, we use the `WHERE` clause to limit our results to `Walmart`.

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

Notice that we don't have to use the word `INNER` for our inner joins.

```
SELECT orders.id, accounts.name, orders.total
FROM orders
JOIN accounts
ON accounts.id = orders.account_id
WHERE accounts.name = 'Walmart'
LIMIT 5
```

In addition, you 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 o.id, a.name, o.total
FROM orders as o
JOIN accounts as a
ON a.id = o.account_id
WHERE a.name = 'Walmart'
LIMIT 5
```

We can also get rid of the `AS`:

```
SELECT o.id, a.name, o.total
FROM orders o
JOIN accounts a
ON a.id = o.account_id
WHERE a.name = 'Walmart'
LIMIT 10
```

id	name	total
1	Walmart	169
2	Walmart	288
3	Walmart	132
4	Walmart	176
5	Walmart	165
6	Walmart	173
7	Walmart	226
8	Walmart	293
9	Walmart	129
10	Walmart	148

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 o.id, a.name, o.total
FROM orders o
JOIN accounts a
ON a.id = o.account_id and a.name = 'Walmart'
LIMIT 10
```

id	name	total
1	Walmart	169
2	Walmart	288
3	Walmart	132
4	Walmart	176
5	Walmart	165
6	Walmart	173
7	Walmart	226
8	Walmart	293
9	Walmart	129
10	Walmart	148

INNER JOINS (Three or More Tables)

This same logic can actually assist in joining more than two tables together. Look at the three tables in Figure 4.

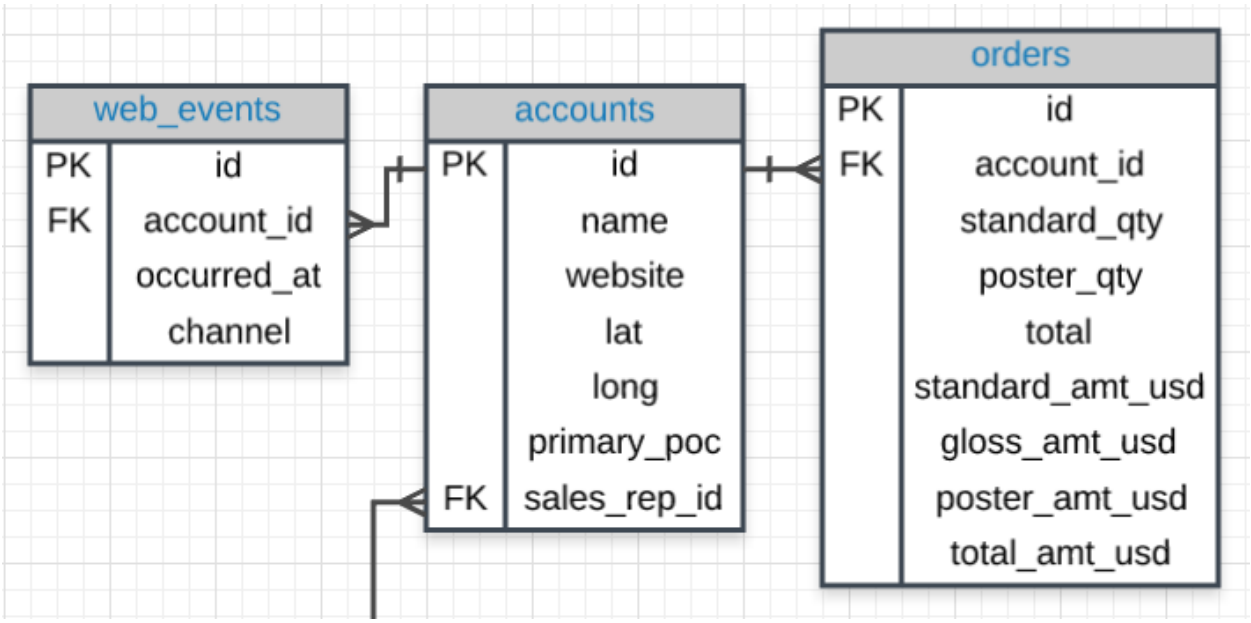

Figure 4: Three Tables JOIN

2. **YOUR TURN** Provide 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 (call it `account_name`)

3. Provide 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 I divided by (total + 0.01) to assure not dividing by zero.

```
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
LIMIT 10;
```

region	account	unit_price
Northeast	Walmart	5.759600
Northeast	Walmart	5.965175
Northeast	Walmart	5.879706
Northeast	Walmart	5.444236
Northeast	Walmart	5.960184
Northeast	Walmart	6.168719
Northeast	Walmart	6.628910
Northeast	Walmart	7.003857
Northeast	Walmart	5.833424
Northeast	Walmart	5.935815

OUTER JOINS

In the previous section, we talked about INNER JOINS which links two or more tables to find all the rows that match. The OUTER JOIN return not only the rows that match on the criteria you specify but also the unmatched rows from either one or both of the two tables you want to join.

There are three types of OUTER JOINS: (1) LEFT JOIN, (2) RIGHT JOIN, and (3) FULL JOIN.

LEFT AND RIGHT OUTER JOINS

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is `NULL` from the right side, if there is no match (refer to Figure 3, second venn diagram).

What's the difference between the LEFT and RIGHT JOIN? When you begin building a query user LEFT JOIN, the first table you name in the FROM clause is the table considered on the left, and the second table in the JOIN clause is considered on the right. For example, if you want all the rows from the first table and any matching rows from the second table, you will use a LEFT OUTER JOIN. Conversely, if you want all the rows from the second table and any matching rows from the first table, you will specify a RIGHT OUTER JOIN.

FULL OUTER JOIN

The last type of join is a full outer join. This will return the inner join result set, as well as any unmatched rows from either of the two tables being joined.

Again this returns rows that do not match one another from the two tables. The use cases for a full outer join are very rare.

You can see examples of outer joins at the link **here** and a description of the rare use cases **here**. We will not spend time on these given the few instances you might need to use them.

Similar to the above, you might see the language FULL OUTER JOIN, which is the same as OUTER JOIN.

More Exercises on Parch and Posey

4. Provide 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 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'
LIMIT 10;
```

region	rep	account
Midwest	Sherlene Wetherington	Rite Aid
Midwest	Chau Rowles	Tech Data
Midwest	Charles Bidwell	Qualcomm
Midwest	Cliff Meints	Sears Holdings
Midwest	Delilah Krum	Capital One Financial
Midwest	Kathleen Lalonde	EMC
Midwest	Julie Starr	USAA
Midwest	Cordell Rieder	Duke Energy
Midwest	Sherlene Wetherington	Time Warner Cable
Midwest	Chau Rowles	Halliburton

5. **YOUR TURN** are the different channels used by account id 1001? Your final table should have only 2 columns: account name and the different channels. You can try SELECT DISTINCT to narrow down the results to only the unique values.

6. **YOUR TURN** Find all the orders that occurred in 2015. Your final table should have 4 columns: occurred_at, account name, order total, and order total_amt_usd.

Aggregation Function

SQL is excellent at aggregating data the way you might in a pivot table in Excel. You will use aggregate functions all the time, so it's important to get comfortable with them. The functions themselves are the same ones you will find in Excel or any other analytics program. We'll cover them individually in the next few lessons. Here's a quick preview:

- **COUNT**: counts how many rows are in a particular column.
- **SUM**: adds together all the values in a particular column.
- **MIN** and **MAX** return the lowest and highest values in a particular column, respectively.
- **AVG** calculates the average of a group of selected values.

COUNT

COUNT is a SQL aggregate function for counting the number of rows in a particular column.

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

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

count
351

When using COUNT to count the number of rows in an individual column, then it will return the number of rows that are not NULL in this row.

SUM

SUM is a SQL aggregate function that totals the values in a given column. Unlike COUNT, you can only use SUM on columns containing numerical values.

8. 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

9. Find the total amount spent on standard and gloss orders in USD.

```
SELECT SUM(standard_amt_usd + gloss_amt_usd) AS total_standard_gloss
FROM orders
```

total_standard_gloss
17265506

MIN and MAX

MIN and MAX are SQL aggregation functions that return the lowest and highest values in a particular column.

They're similar to COUNT in that they can be used on non-numerical columns. Depending on the column type, MIN will return the lowest number, earliest date, or non-numerical value as close alphabetically to "A" as possible. As you might suspect, MAX does the opposite—it returns the highest number, the latest date, or the non-numerical value closest alphabetically to "Z".

10. What is the min and max order quantity for each poster papers in the database? Provide two columns in the report - `poster_min` and `poster_max`.

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

poster_min	poster_max
0	28262

11. When was the earliest order ever placed? Name the result `earliest_order_date`.

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

min
2013-12-04 04:22:44

12. Try performing the same query as in the previous without using an aggregation function.

```
SELECT occurred_at
FROM orders
ORDER BY occurred_at
LIMIT 1;
```

occurred_at
2013-12-04 04:22:44

AVERAGE

AVG is a SQL aggregate function that calculates the average of a selected group of values. It's very useful, but has some limitations. First, it can only be used on numerical columns. Second, it ignores nulls completely.

13. 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

14. **YOUR TURN** What is the average `standard_qty` spent on by our Ford Motor account?

Sales Orders Database

This database is an order entry database for a store that sells bicyles and accessories. The following figure (Figure 5) shows the ERD for the database.

15. List Customers and the dates they placed an order- provide the first name, last name of the customer and the date they ordered.

```
SELECT c.custFirstName, c.custLastName, o.orderDate
FROM customers c
JOIN orders o
ON c.customerid = o.customerid
LIMIT 10;
```

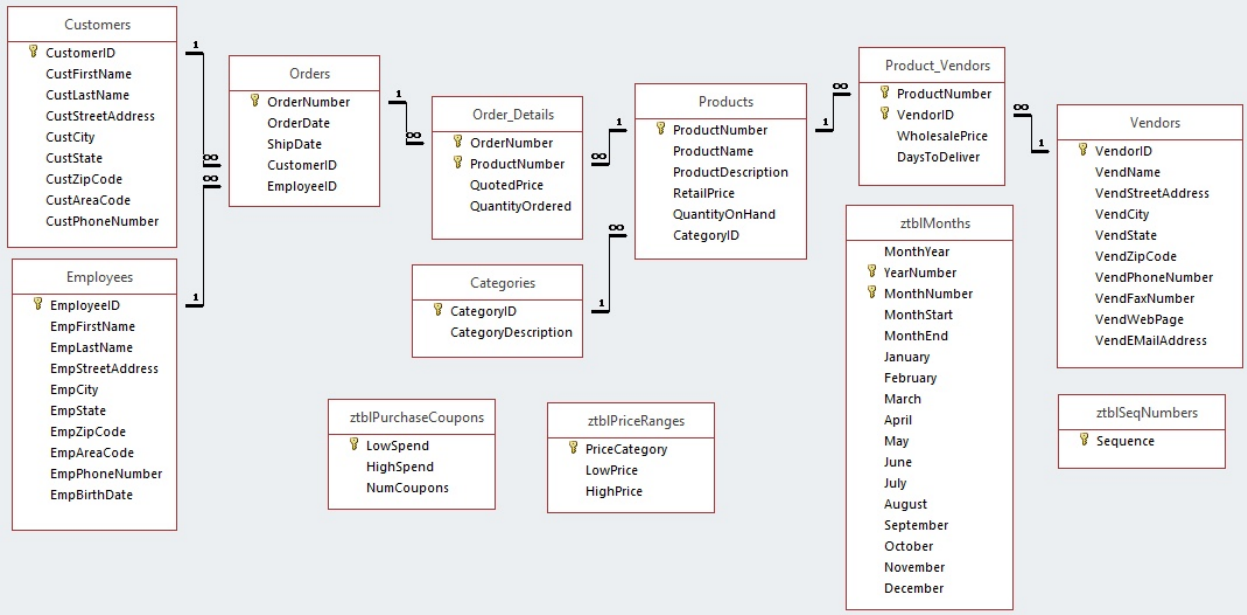

Figure 5: Sales Order Database Schema

custfirstname	custlastname	orderdate
Suzanne	Viescas	2017-09-02
Suzanne	Viescas	2017-09-02
Suzanne	Viescas	2017-09-03
Suzanne	Viescas	2017-09-10
Suzanne	Viescas	2017-09-10
Suzanne	Viescas	2017-09-17
Suzanne	Viescas	2017-09-24
Suzanne	Viescas	2017-09-25
Suzanne	Viescas	2017-09-28
Suzanne	Viescas	2017-09-30

16. Display all products and their categories.

```
SELECT p.productname, c.categorydescription
FROM products p
JOIN categories c
ON p.categoryid = c.categoryid
LIMIT 10;
```

productname	categorydescription
Trek 9000 Mountain Bike	Bikes
Eagle FS-3 Mountain Bike	Bikes
Dog Ear Cyclecomputer	Accessories
Victoria Pro All Weather Tires	Components
Dog Ear Helmet Mount Mirrors	Accessories
Viscount Mountain Bike	Bikes
Viscount C-500 Wireless Bike Computer	Accessories
Kryptonite Advanced 2000 U-Lock	Accessories
Nikoma Lok-Tight U-Lock	Accessories
Viscount Microshell Helmet	Accessories

17. Find all the customers who have ever ordered a bicycle helmet from our company, but have not ordered a mountain bike. Provide two columns - the customers' first name, and the customers' last name.

```
SELECT DISTINCT c.custfirstname, c.custlastname
FROM customers c
    JOIN orders o
      ON c.customerid = o.customerid
      JOIN order_details od
      ON od.ordernumber = o.ordernumber
      JOIN products p
      ON p.productnumber = od.productnumber
WHERE (p.productname LIKE '%helmet%' OR p.productname LIKE '%Helmet%') AND (NOT p.productname LIKE '%Bi
LIMIT 10;
```

custfirstname	custlastname
Andrew	Cencini
Angel	Kennedy
Caleb	Viescas
Darren	Gehring
David	Smith
Dean	McCrae
Estella	Pundt
Gary	Hallmark
Jim	Wilson
John	Viescas

18. List employees and the customers for whom they booked an order for. Provide the concatenated employee name and concatenated customer name.

```
SELECT DISTINCT concat(e.empfirstname, ' ', e.emplastname)as EmployeeName,
                concat(c.custfirstname, ' ', c.custlastname) as CustomerName
FROM customers c
JOIN orders o
ON c.customerid = o.customerid
JOIN employees e
ON o.employeeid = e.employeeid
ORDER BY employeename, customername
LIMIT 10;
```

employeename	customername
Ann Patterson	Alaina Hallmark
Ann Patterson	Andrew Cencini
Ann Patterson	Angel Kennedy
Ann Patterson	Caleb Viescas
Ann Patterson	Darren Gehring
Ann Patterson	David Smith
Ann Patterson	Dean McCrae
Ann Patterson	Estella Pundt
Ann Patterson	Gary Hallmark
Ann Patterson	Jim Wilson

19. Count the number of orders who are associated with the employee *Kathryn Patterson*. Return only the count.

```
SELECT COUNT(ordernumber)
FROM orders o
JOIN employees e ON
 o.employeeid = e.employeeid
WHERE e.employeeid = '707'
```

count
140

20. **YOUR TURN** Display all order details that have a total quoted value greater than 100, the products in each order, and the amount owed for each product. Provide the order number, product name, and amount owed.
21. **YOUR TURN** Show me the vendors and the products they supply to us for products that cost less than $100 (use `wholesaleprice`) and take less than 1 week to deliver. Just provide the product name and vendor names.
22. **YOUR TURN** Show me customers and employees who have the same last name. Provide a report with the customer ID, customer last name, and employee ID.
23. **YOUR TURN** Display customers who have no sales rep (employees) in the same ZIP code. Provide the customer ID, zip code, and customer name. Have the customer name be in one column, first and last name concatenated.
24. **YOUR TURN** Which products have more than 10 units on hand but have never been ordered? Output the product number, product name, and order number.
25. **YOUR TURN** How many customers are based in Texas and have made at least one order?