DATA-ANALYTICS-7-8-NOTEBOOKS (/github/juantorres-c1/DATA-ANALYTICS-7-8-NOTEBOOKS/tree/main)
 /
extended.sql_fellow (/github/juantorres-c1/DATA-ANALYTICS-7-8-NOTEBOOKS/tree/main/extended.sql_fellow)

Finding our best-performing salespeople and products

Total points: 24 points

Introduction

Business Context. You work for AdventureWorks, a company that sells outdoor sporting equipment. The company has many different locations and has been recording the sales of different locations on various products. You, their new data scientist, have been tasked with the question: *What are our best products and salespeople, and how can we use this information to improve our overall performance?*

You have been given access to the relevant data files with documentation from the IT department. Your job is to extract meaningful insights from these data files to help increase sales. First, you will look at the best products and try to see how different products perform in different categories. Second, you will analyze the best salespeople to see if the commission percentage motivates them to sell more.

Business Problem. Your task is to *write queries in SQL to carry out the requested analysis*.

Analytical Context. You are given the data as an SQLite database. The company has been pretty vague about how they expect you to extract insights, but you have come up with the following plan of attack:

1. Load the database and ensure you can run basic queries against it
2. Look at how product ratings and total sales are related
3. See how products sell in different subcategories (bikes, helmets, socks, etc.)
4. Calculate which salespeople performed the best in 2014
5. See if total sales are correlated with commission percentage

Of course, this is only your initial plan. As you explore the database, your strategy will likely change.

Overview of the data

The data for this case is contained in the `AdventureWorks.db` (AdventureWorks.db) SQLite database. We will be focusing on the tables that belong to the Sales and Product categories. Complete documentation, with schemas, for the original data (of which you have only a subset) can be found here (data/AdventureWorks.pdf).

Product Tables (Pg. 34 in the documentation):

- **Product**: one row per product that the company sells
- **ProductReview**: one row per rating and review left by customers
- **ProductModelProductDescriptionCulture**: a link between products and their longer descriptions also indicating a "culture" - which language and region the product is for
- **ProductDescription**: a longer description of each product, for a specific region
- **ProductCategory**: the broad categories that products fit into
- **ProductSubCategory**: the narrower subcategories that products fit into

Sales Tables (Pg. 71 in the documentation):

- **SalesPerson**: one row per salesperson, including information on their commission and performance
- **SalesOrderHeader**: one row per sale summarizing the sale
- **SalesOrderDetail**: many rows per sale, detailing each product that forms part of the sale
- **SalesTerritory**: the different territories where products are sold, including performance
- **CountryRegionCurrency**: the currency used by each region
- **CurrencyRate**: the average and closing exchange rates for each currency compared to the USD

Tip: Review the rest of the documentation carefully to learn more about the tables (like relevant columns in each) and the relationships between them. Note that not all columns may be available in the subset provided in this case, as they are not necessary for the following exercises.

Let's now load the database:

```
In [ ]: %FETCH https://amzn-dana.workspace-lite.correlation-one.com/extended.sql_jlite_fell
```

```
In [ ]: %LOAD AdventureWorks RW
```

Note: Do not round your results (i.e., leave them with as many decimal digits as they have). Also, be sure to name your columns *exactly* as they are in the sample tables in each exercise.

Finding our most popular products

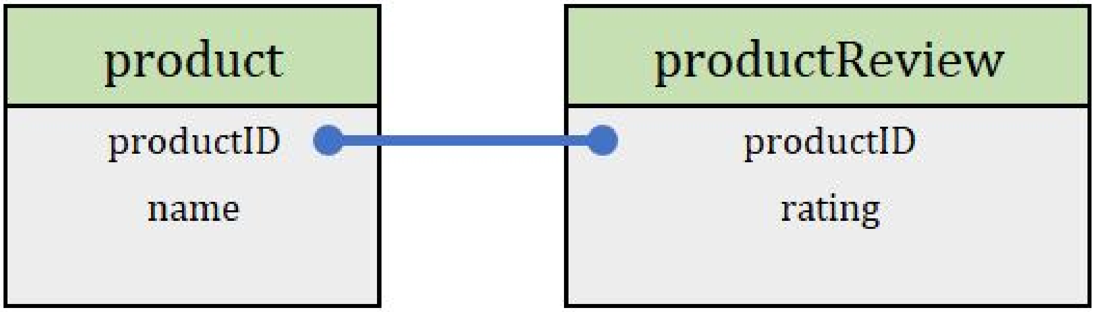
The company would like to know which of its products is the most popular among customers. You figure that the average rating given in reviews is correlated with the number of sales of a particular product (that products with higher reviews have more sales).

Exercise 1 (1 point)

Using the `product` and `productReview` tables, `INNER JOIN` them and rank the products according to their average review rating.

Below is a simplified visual of the `ERD (Entity Relationship Diagram)` , which outlines the table that you will be working with.

- The `Table Name` is the green header.
- The `Column Header` is listed under their respective Table Name.
- The `Blue Line` is there to show the `JOIN` column.
- Based on this information, write a query to retrieve product review details for each product.
- For more detailed information about the Tables, refer to the documentation (data/AdventureWorks.pdf).



Additional Hints:

- Each table has more columns than what is shown in the diagram.
- To view all of the columns, use one of the two methods. `SELECT everything` , or use `Pragma` .
- You will need to use `GROUP BY` to group multiple reviews together.
- `avgrating` is an aggregated column, it takes the average of multiple reviews.
- `num_ratings` is an aggregated column, it counts the number of reviews for a particular product.
- Be aware that SQL is syntax sensitive, meaning the order which you write your query matters. Check the `Sample Code` section for more detail.

Here is a sample output what your answer should look like:

productid	NAME	avgrating	num_ratings
709	Mountain Bike Socks, M	5.0	1
...	...	...	...

Sample Code:

```
-- This will output 1 row of information from the Table with all of the columns.
SELECT *
FROM TableName
LIMIT 1

-- This will show you a list of columns within a table.
pragma table_info( 'Insert a table name in here' )

-- This is the order of SQL needs to be written in.
SELECT
FROM
JOIN
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT
```

Additional Learning Resources:

- W3 School - JOIN (https://www.w3schools.com/sql/sql_join.asp)
- W3 School - GROUP BY (https://www.w3schools.com/sql/sql_groupby.asp)
- W3 School - Aggregate Functions (https://www.w3schools.com/sql/sql_count_avg_sum.asp)

Answer.

```
In [ ]: -- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

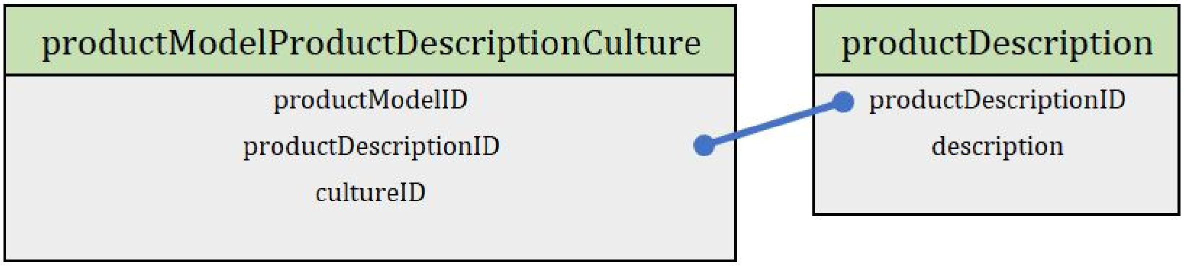
Exercise 2

Exercise 2.1 (1 point)

Much to your disappointment, there are only three products with ratings and only four reviews in total! This is nowhere near enough to perform an analysis of the correlation between `reviews` and `total sales`. Since we cannot infer the most popular products from the reviews, we will go with an alternative strategy. The database includes transactions in different `currencies` and `products` from different cultures. Since we don't have sufficient detail within the reviews, we turn our attention to extracting this insights from `sales` data.

Here is a visual of the tables and columns that you will be working with.

- There are two tables `productModelProductDescriptionCulture` and `productDescription`.
- `JOIN` the tables together by the column with the `Blue Line`.
- For this exercise, include only descriptions for which `productModelProductDescriptionCulture.cultureID = 'en'`.
- Get the `productModelID` and `description` for each product.



Additional Hints:

- Use `Alias` to shorten long table names.

Your output should look like this:

productmodelid	description
1	Light-weight, wind-resistant, packs to fit into a pocket.
2	Traditional style with a flip-up brim; one-size fits all.
3	Synthetic palm, flexible knuckles, breathable mesh upper. Worn by the AWC team riders.
...	...

Additional Learning Resources:

- W3 School - Alias (https://www.w3schools.com/sql/sql_alias.asp)

Answer.

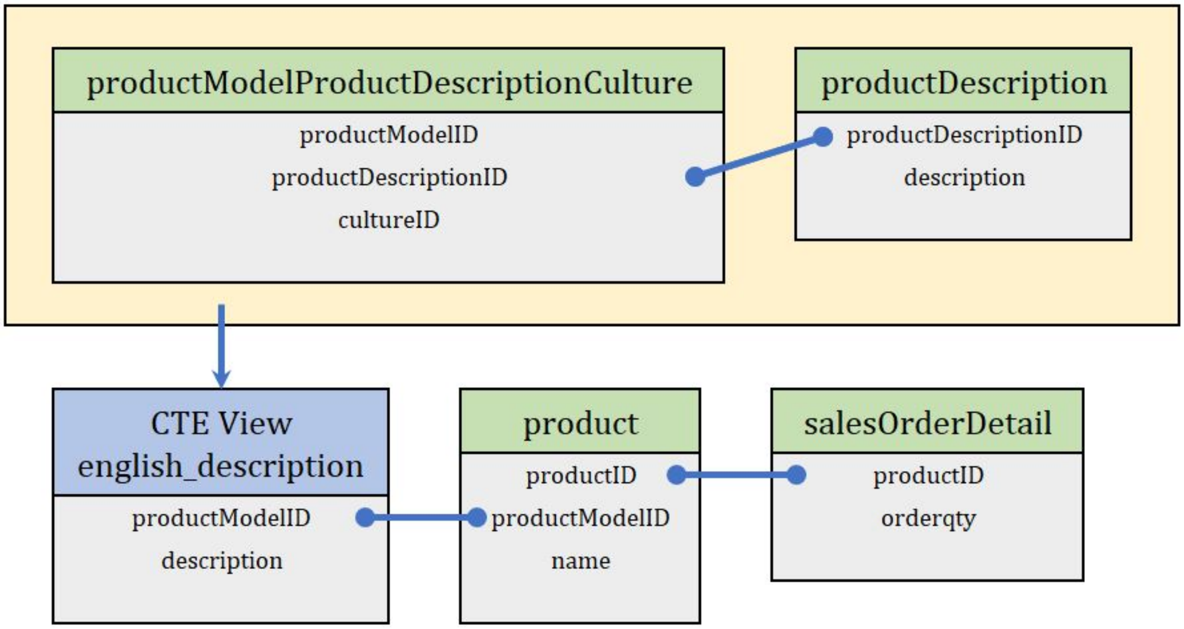
In [ ]:
```
-- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

Exercise 2.2 (2 points)

Now that we got the **`productModelID`** and its **`description`** . We can use the result in Exercise 2.1 to further expand our query to find out its **`name`** , and the **`quantity`** it sold. To do this, we will use **`CTE (Common Table Expression)`** to turn our Exercise 2.1 solution into a temporary table.

Here is a visual of the tables and columns that you will be working with.

- Using the answer in Exercise 2.1, wrap your solution into a **`CTE view`** called **`english_description`** , the **`Blue Header`** shows that the table is only there temporary.
- With CTE View created, you can **`JOIN`** the **`CTE View`** along with other tables.
- Get the **`productModelID`** , **`description`** , **`name`** , and **`total number of sales`** for each product and display the top-10 selling products.
- You can infer how often products have been sold by looking at the **`salesOrderDetail`** table (each row might indicate more than one sale, so take note of **`OrderQty`**).



Additional Hints:

- Make the query you wrote in Exercise 2.1 a temporary view with the **`WITH ... AS`** syntax. It will give you the English descriptions of the products as a starting point.
- Then **`JOIN`** the temporary table with other relevant tables.
- This exercise require multiple **`JOIN`** to link several tables together.
- You will need to use combination of **`ORDER BY`** and **`LIMIT`** to sort the table to get TOP 10 products.
- **`total_orders`** is an aggregated column that sums up the quantity of units ordered.

Your output should look like this:

productmodelid	description	NAME	total_orders
2	Traditional style with a flip-up brim; one-size fits all.	AWC Logo Cap	8311
111	AWC logo water bottle - holds 30 oz; leak-proof.	Water Bottle - 30 oz.	6815
33	Universal fit, well-vented, lightweight , snap-on visor.	Sport-100 Helmet, Blue	6743
...	...	...	...

Additional Learning Resources:

- What Is a Common Table Expression (CTE) in SQL? (https://learnsql.com/blog/what-is-common-table-expression/)
- How to use CTE using WITH...AS (https://www.youtube.com/watch?v=_SanZ41uTlw)

Answer.

In [ ]:

```
-- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

Exercise 3

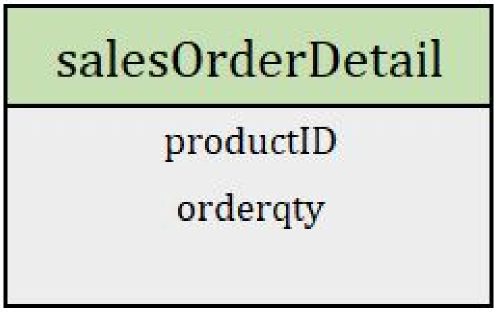
To get a better sense of the sales, let's look at the correlation between the **`quantity sold`** and the price for each **`subcategory`** . We will break this process into three steps.

1. Find out the **`quantity sold`** by their **`productID`** .
2. Next we will query the **`listing price`** by their **`productID`** and their respective **`category`** and **`subcategory`** .
3. Finally, we will **`JOIN`** both results together so we will know how many were sold by their **`category`** and **`subcategory`**

Exercise 3.1 (1 Point)

First we want to start by writing a query that shows how many items were ordered in total for every product in the database. Do *not* filter by culture.

Here is a visual of the tables and columns that you will be working with.



Additional Hints:

- Use the **`salesOrderDetail`** table.
- **`quantity`** is an aggregated column that sums up the quantity of unit sold and **`GROUP BY`** the product id.

Your output should look like this:

productid	quantity
707	6266
708	6532
709	1107
710	90
711	6743
712	8311
713	429
714	3636
...	...

Answer.

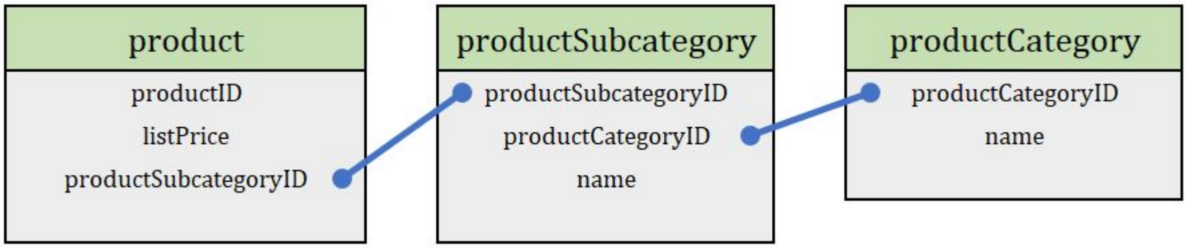
In [ ]:

```
-- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

3.2 (1 point)

Awesome! Next, we need to know the `list price` for each product alongside its `category` and `subcategory` .

Here is a visual of the tables and columns that you will be working with.



Additional Hints:

- You will find the product categories in the `productCategory` table, and the subcategories in the `productSubcategory` table.
- Use multiple JOIN to connect `product` , `productCategory` , and `productSubcategory` tables together.

Your output should look like this:

productid	category	subcategory	listprice
680	Components	Road Frames	1431.5
706	Components	Road Frames	1431.5
707	Accessories	Helmets	34.99
708	Accessories	Helmets	34.99
709	Clothing	Socks	9.5
710	Clothing	Socks	9.5
711	Accessories	Helmets	34.99
712	Clothing	Caps	8.99
713	Clothing	Jerseys	49.99
714	Clothing	Jerseys	49.99
715	Clothing	Jerseys	49.99
716	Clothing	Jerseys	49.99
717	Components	Road Frames	1431.5
718	Components	Road Frames	1431.5
719	Components	Road Frames	1431.5
...	...	...	...

Answer.

In [ ]:

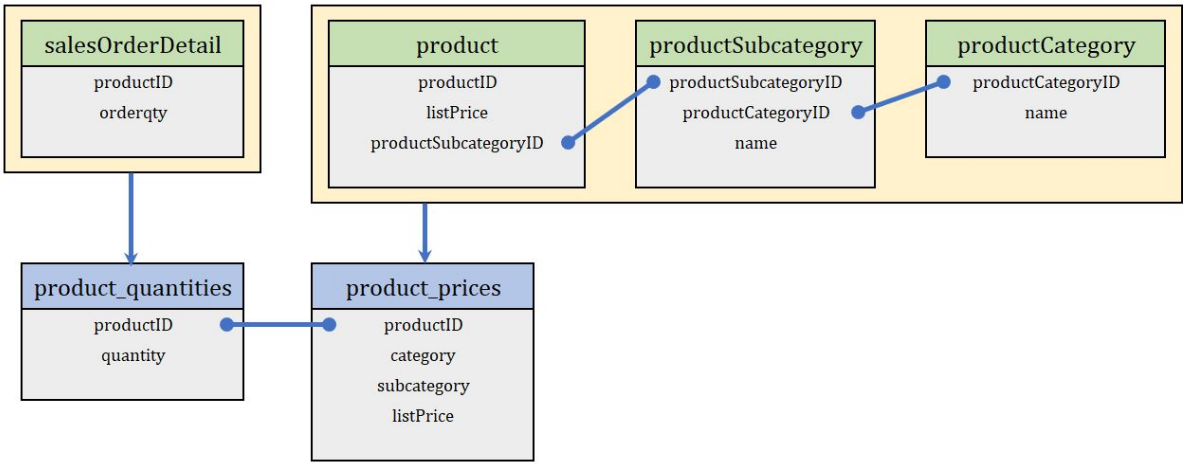
```
-- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

Exercise 3.3 (3 points)

Now that we have the productID and its quantity sold from Exercise 3.1, as well as the category, subcategory and list price from Exericse 3.2. We can now merge the two solutions together to obtain a table showing the `average list price` and the `total quantity of products sold` for each subcategory.

Here is a visual of the tables and columns that you will be working with.

- We need to put our answers from `Exercise 3.1` and `Exercise 3.2` into their respective CTE.
- Then use `JOIN` to join the 2 CTE tables together.



Additional Hints:

- To have two `WITH ... AS` statements in the same query, you separate the subqueries with a comma and don't write `WITH` again. Like the sample code below.
- If you used any `LIMIT CLAUSES` in previous Exercise, remove them.
- `average_price_in_subcategory` is an aggregated column calculated from the CTE.
- `total_items_sold_in_subcategory` is an aggregated column calculated from the CTE.

Your output should look like this:

category	subcategory	average_price_in_subcategory	total_items_sold_in_subcategory
Accessories	Bike Racks	120.0	3166

category	subcategory	average_price_in_subcategory	total_items_sold_in_subcategory
Accessories	Bike Stands	159.0	249
Accessories	Bottles and Cages	7.989999999999999	10552
Accessories	Cleaners	7.95	3319
Accessories	Fenders	21.98	2121
Accessories	Helmets	34.99	19541
Accessories	Hydration Packs	54.99	2761
Accessories	Locks	25.0	1087
Accessories	Pumps	19.99	1130
Accessories	Tires and Tubes	19.482727272727274	18006
Bikes	Mountain Bikes	1683.3649999999982	28321
Bikes	Road Bikes	1597.45	47196
Bikes	Touring Bikes	1425.2481818181814	14751
Clothing	Bib-Shorts	89.99	3125
...	...	...	...

Sample Code:

```
WITH first_query_alias AS
(
    SELECT ...
),
second_query_alias AS -- Notice we didn't include a second WITH here
(
    SELECT...
)
SELECT ...
```

Answer.

```
In [ ]: -- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

Turns out there is a positive correlation between average price and items sold ($\rho = 0.68$). This is somewhat unexpected since common sense tells us that the more expensive an item is, the lower the demand for it. It is possible that we are witnessing an instance of Simpson's Paradox here. To verify if that is indeed the case, we could instead compute the correlation coefficient for each subcategory, possibly evidencing a negative correlation coefficient in some subcategories. We will not do that right now however, since it would make us deviate too much from our business problem. (You can use Excel for tasks like this.)

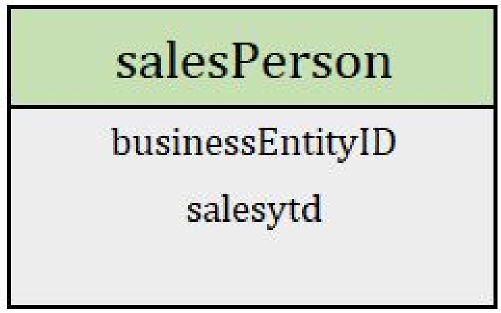
Finding our top salespeople

As mentioned earlier, we want to find our best salespeople and see whether or not we can incentivize them in an appropriate manner. Namely, we want to determine if the commission percentage we give them motivates them to make more and bigger sales.

Exercise 4 (1 point)

Let's start by finding our top five performing salespeople by using the `salesytd` (Sales, year-to-date) column.

Here is a visual of the tables and columns that you will be working with.



Additional Hints:

- We only need to know the `businessEntityID` for each salesperson as this uniquely identifies each salesperson.
- Your query should, therefore, only have two columns: `businessEntityID` and `salesytd` .
- Do not round the numbers.

Your output should look like this:

businessentityid	salesytd
276	4251368.5497
289	4116871.2277
275	3763178.1787
...	...

Answer.

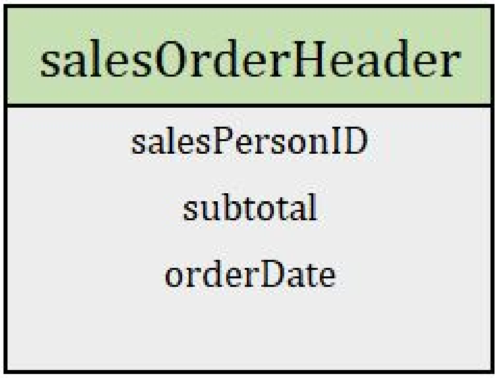
```
In [ ]: -- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

Exercise 5 (2 points)

The sales numbers from the previous query are hard-coded into the **`salesPerson`** table instead of dynamically calculated from each sales record. Currently, we don't know how this number is updated or much about it at all, so it's good to remain skeptical.

Here is a visual of the tables and columns that you will be working with.

- Using the **`salesOrderHeader`** table, find the top 5 salespeople who made the most sales *in the most recent year available* (2014) (there is a column called **`subtotal`** - use that.)
- Sales that do not have an associated salesperson should be excluded from your calculations and final output.
- All orders that were made within the 2014 calendar year should be included.



Additional Hints:

- Use **`WHERE CLAUSE`** to filter the dates.
- Put into consideration that some records don't have dates, filter it using **`IS NOT NULL`** keywords, and use **`<> comparison operator`** .
- Do not worry about rounding the numbers.

You can use the syntax `WHERE column >= '1970-01-01'` to generate an arbitrary date in SQLite and compare this to specific dates in the tables (in this example, dates equal to or later than Jan 1, 1970). Additionally, when you want to make sure that columns with empty or null values are excluded from a query in SQLite, you have to add a line like this one to your `WHERE` statement: `my_column IS NOT NULL AND my_column <> ""` . The `<>` operator is the opposite of `=` ; that is, it checks that two values are different from each other.

Your output should look like this:

salespersonid	totalsales
289	1382996.5839000002
276	1271088.5216
...	...

Additional Learning Resources:

- W3 - Working with Dates (https://www.w3schools.com/sql/sql_dates.asp)
- W3 - Working with NULLS (https://www.w3schools.com/sql/sql_null_values.asp)
- W3 - Using AND, OR, NOT (https://www.w3schools.com/sql/sql_and_or.asp)

Answer.

```
In [ ]: -- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

You should see right away that there are discrepancies between the two sales totals. This makes sense because we used filters in one table and not the other. Nonetheless, for the remainder of this case, use this dynamically-calculated total as the authoritative answer.

Exercise 6

Since there are discrepancies between Salesperson table and SalesOrderHeader table, let's double check this by investigating our sales record by manually adding all of the sales amount together.

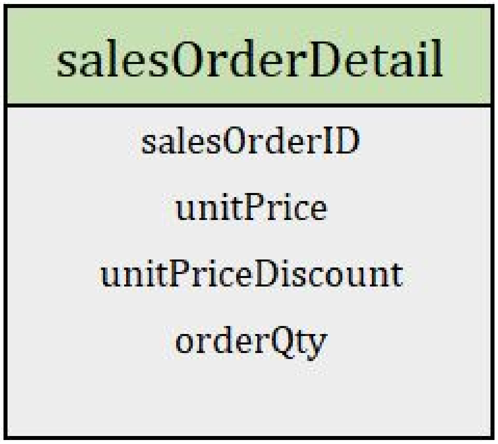
Looking at the documentation, you will see that `subtotal` in the `salesOrderHeader` table is calculated from other tables in the database. To validate this figure (instead of trusting it blindly), it might be a good idea to calculate the `subtotal` manually. Using the `salesOrderDetail` and `salesOrderHeader` tables, let's calculate the sales for each salesperson for **the year 2014** and display the results for the top 5 salespeople.

6.1 (1 point)

Write a query that shows the total amount of money paid by their `salesOrderID` (find this column in the `salesOrderDetail` table).

Note: It is good practice to always use a limit statement (i.e., LIMIT 10) when you do not know how large the output might be. Otherwise, the size of some of these tables may crash your browser and you will need to reload it and rewrite your queries.

Here is a visual of the tables and columns that you will be working with.



salesOrderDetail

salesOrderID

unitPrice

unitPriceDiscount

orderQty

Additional Hints

- The `ordertotal` column in the output sample is an aggregated column
- Remember to subtract `unitPriceDiscount` from each item's price (`unitPriceDiscount` is a percentage).
- You can check out this link (https://www.geeksforgeeks.org/sql-arithmetic-operators/) to get a refresher on SQL arithmetic
- Do not worry about rounding the numbers

Your output should look like this:

salesorderid	ordertotal
43659	20565.6206
43660	1294.2529
43661	32726.4786
43662	28832.5289
43663	419.4589
43664	24432.608799999995
43665	14352.7713
43666	5056.4896
43667	6107.081999999999
43668	35944.156200000005
43669	714.7043
...	...

Additional Learning Resources

- How to Calculate Discount (https://www.omnicalculator.com/finance/percentage-discount#:~:text=To%20determine%20the%20discount%20percentage,price%20from%20th

Answer.

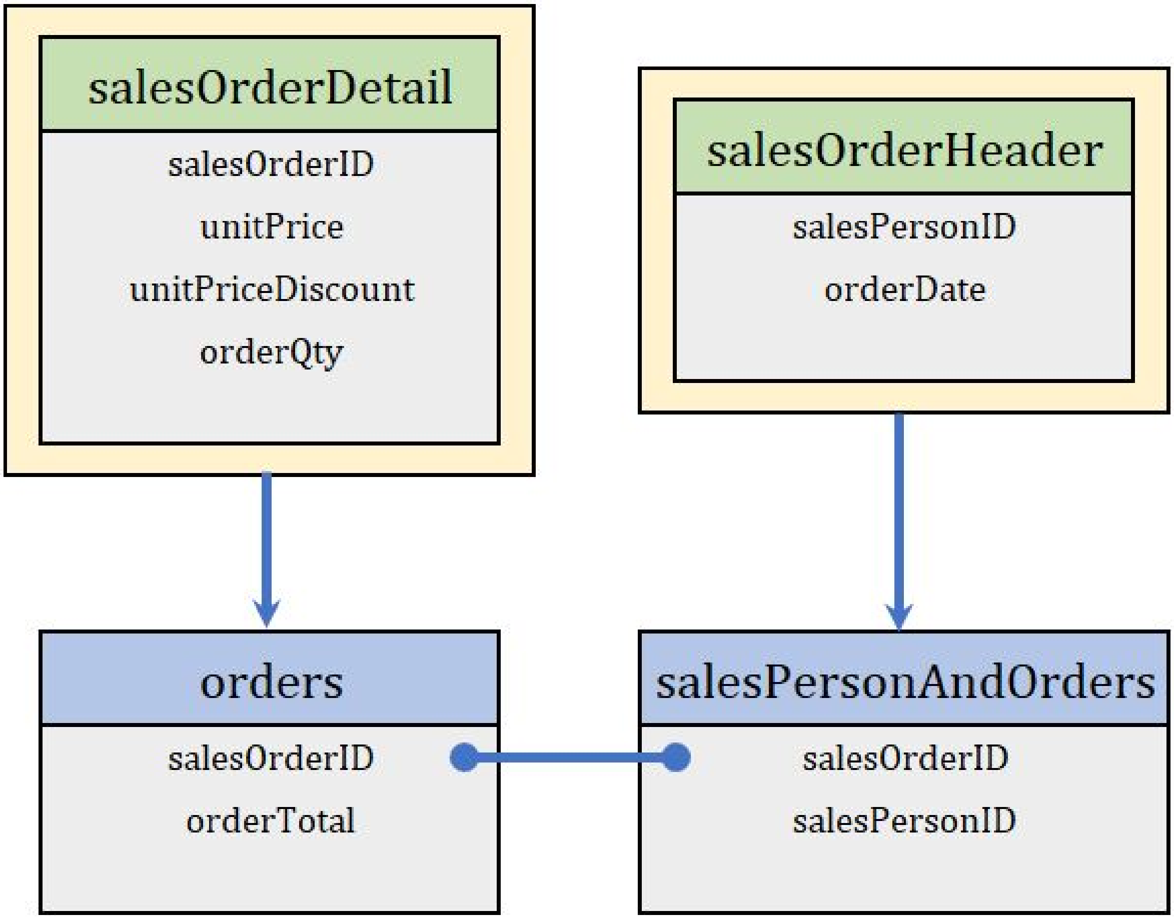
In [ ]:
```
-- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

6.2 (2 points)

Using the previous query as a subquery (CTE), find the sales for each salesperson for the year 2014 and display results for the top 5 salespeople.
Remember to exclude sales that are not associated with a salesperson.

Here is a visual of the tables and columns that you will be working with.

- The CTE on the left is created from **`Exercise 6.1`** .
- The CTE on the right is modified from **`Exercise 5`** , you will need to add the **`salesOrderID`** .



Additional Hints:

- You can get the **`salesOrderID`** and **`salesPersonID`** pairs from the **`salesOrderHeader`** table.
- It is recommended to use additional CTE to help simplify your query.
- Use **`Exercise 6.1`** solution as CTE.
- Modify **`Exercise 5`** and use it as CTE.

Your output should look like this:

salespersonid	ordertotalsum
289	1382996.5839100003
276	1271088.5214610002
...	...

Answer.

In [ ]:

```
-- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

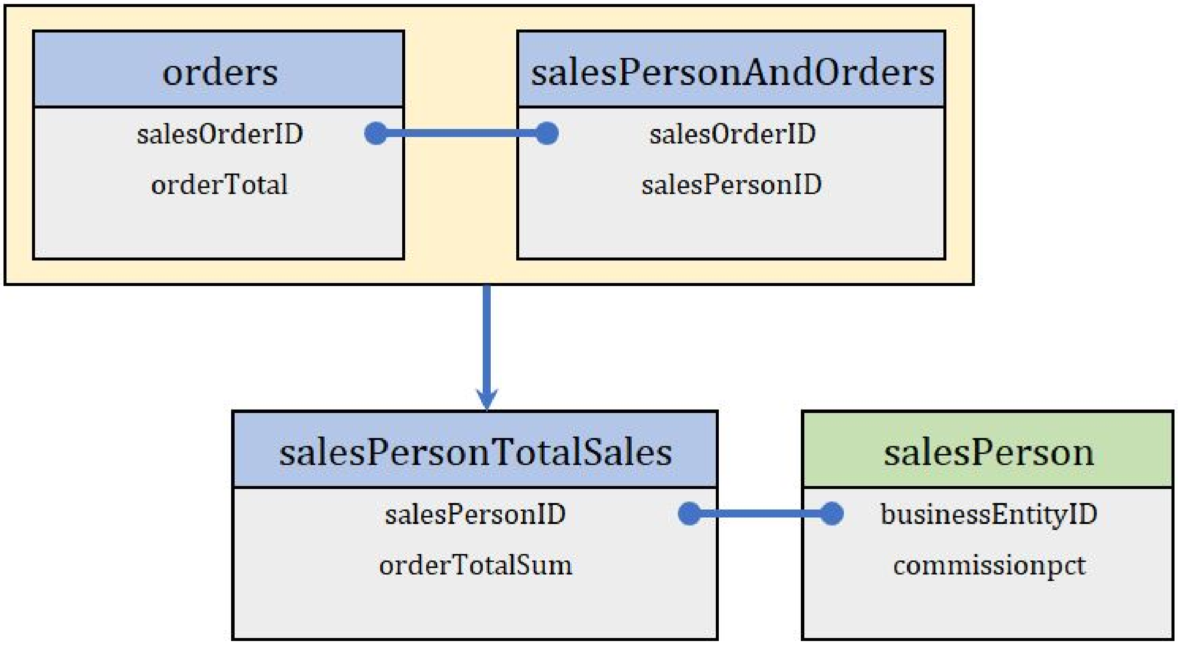
The results are the same as Exercise 5. We still prefer this query though because it is generated from granular data instead of relying on hard-coded figures.

Exercise 7 (3 points)

Next, let's now see whether there is a positive relationship between the **`total sales of the salespeople`** and their **`commission percentages`** .

Here is a visual of the tables and columns that you will be working with.

- Using your previous query from **`Exercise 6.2`** , use it as CTE.
- Join the new CTE query (remove the **`LIMIT`** clause) with the **`salesPerson`** table



Additional Hints:

- Remember that the **`businessEntityID`** column from the **`salesPerson`** is compatible with the **`salesPersonID`** column in the query of exercise 6 (they both represent the salesperson ID).
- Use **`Nested CTE`** by encasing the entire query in Exercise 6.2 into its own CTE.
- **`JOIN`** the new **`Nested CTE`** with **`salesPerson`** table.
- Once you get your SQL output, paste the results into an Excel table and use the `=CORREL()` (https://support.microsoft.com/en-au/office/correl-function-995dcef7-0c0a-4bed-a3fb-239d7b68ca92) formula to calculate the correlation coefficient. If it is positive, the relationship is positive. If it is negative, the relationship is negative. You can view this visually by creating a scatterplot.

You should get a table like this one:

salespersonid	ordertotalsum	commissionpct
274	178584.36250800002	0.0
275	1057247.378572	0.012
276	1271088.5214610002	0.015
277	1040093.406901	0.015
...	...	...

Answer.

We externally calculated the correlation coefficient between `ordertotalsum` and `commissionpct` , which turned out to be $\rho = 0.73$. This suggests that the salespeople who earn a high commission are also those who close the bigger deals.

Exercise 8

Remember how we mentioned that products were sold in many regions? This is why you had to work with the `culture` value before getting the English language descriptions. The problem is you are now told the sales are recorded in *local* currency, so your previous analyses are flawed. Technically, you must convert all amounts to USD if you wish to compare the different salespeople fairly! Instead, let's group the salespeople orders by the currency used for each order (you will have to consider `tocurrencyrate` for this task in the `CurrencyRate` table).

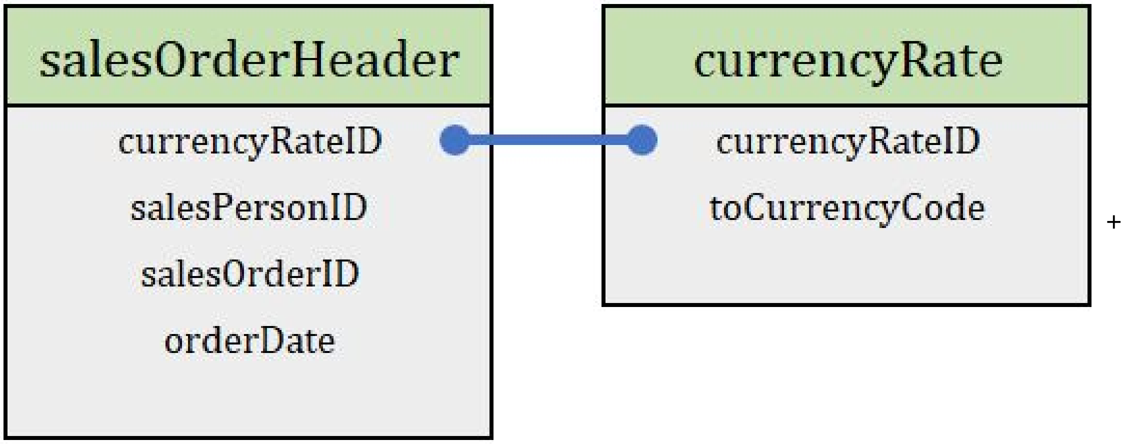
8.1 (1 point)

Let's explore the currencies in different sales. But first, here are some things to understand about the currency columns:

- The **`FromCurrencyCode`** is all USD, so focus on **`toCurrencyRate`**
- If the sale was paid in USD, the **`currencyRateID`** was left blank (since there was no need to make a conversion)

Create a table with the **`salesPersonID`** , **`salesOrderID`** , **`CurrencyRateID`** and **`toCurrencyCode`** to see the connection. Remember to exclude sales that are not associated with a salesperson and only consider sales in 2014. Order by the salesperson ID and show only 10 rows.

Here is a visual of the tables and columns that you will be working with.



Additional Hints

- Since **`USD`** would not show up in the **`CurrencyRate`** table, you will have to do a **`LEFT JOIN`** to avoid losing information.
- Be sure that you understand how the content of the **`CurrencyRate`** table is read and what each column means.
- Use additional cells to experiment if needed.
- The **`None`** in the above example takes the place of **`NULL`** values, which contextually means that the sale was in USD.

Your table should look like this:

salespersonid	salesorderid	currencyrateid	tocurrencycode
274	65294	None	None
274	65298	None	None
274	67277	None	None
274	67286	11427	CAD
274	69528	None	None
...	...	...	...

Answer.

In [ ]:

```
-- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

As expected, we can see that different salespeople have sales in different currencies.

Note: The `None` in the above example takes the place of `NULL` values, which contextually means that the sale was in USD.

8.2 (2 points)

Looking good! We can now see which sales order are from a different currency, but there is a big issue!
The **`None`** in the above query can be confusing to someone who doesn't understand the database. In this case, it's best to replace them with useful information.

Redo the previous exercise with the following changes:

- Leave out the **`currencyRateID`** column
- Replace **`None`** with 'USD' in the **`toCurrencyCode`** column
- One way of completing this task is to use the **`CASE`** expression, which can be incorporated as outlined in the **`Sample Code`** below.

The expected output should look like this, where **None** has been replaced by **USD**, while other values remain as is...

salespersonid	salesorderid	tocurrencycode
274	65294	USD
274	65298	USD
274	67277	USD
274	67286	CAD
274	69528	USD
...	...	...

Additional Learning Resource

- W3 - CASE Expression (https://www.w3schools.com/sql/sql_case.asp)

Sample Code

```
SELECT column1, column2,
CASE
    WHEN condition1 THEN result1
    ELSE result2
END AS column3
FROM TableName
```

Answer.

In [ ]:

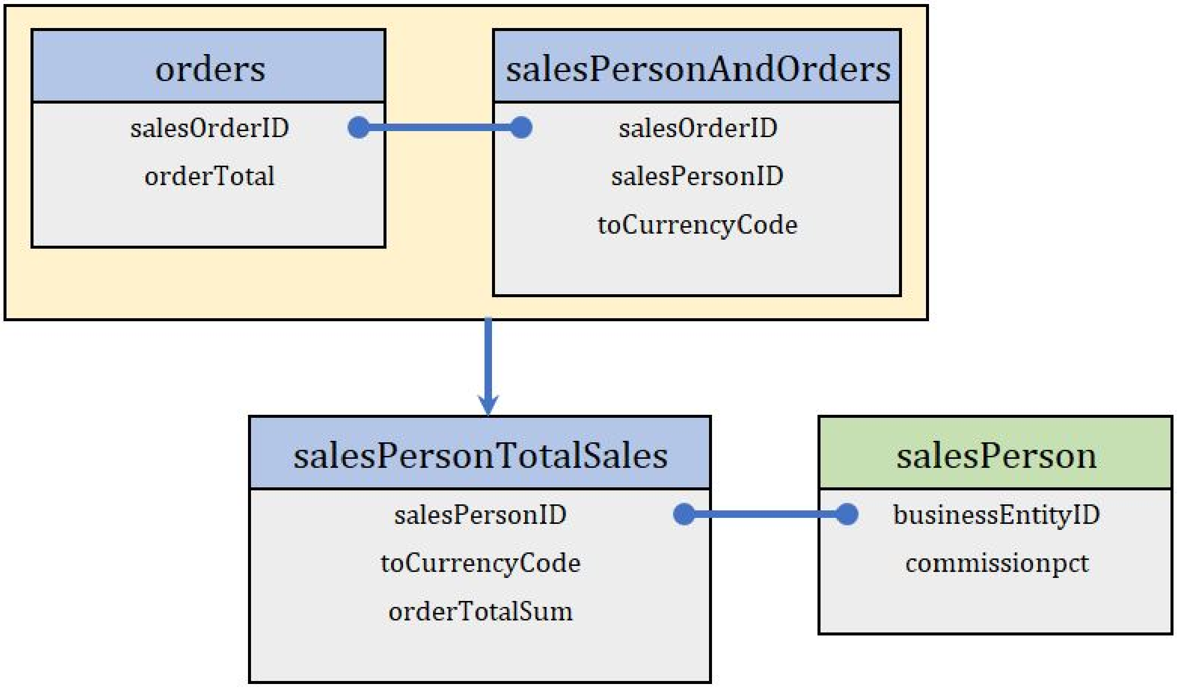
```
-- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

Exercise 9 (3 points)

Much better! Now that we have the currency codes associated with each salesperson ID, redo Exercise 7, adding in the `toCurrencyCode`. Order the results by currency (ascending) and total sales (descending) to make it easier to see who the best salespeople are for each currency.

Here is a visual of the tables and columns that you will be working with.

- The ERD comes from Exercise 7 with slight modification
- `salesPersonAndOrder` has been modified to include `toCurrencyCode`
- Remember to integrate the `CASE expression` to get USD values in, refer to Exercise 8.2
- The resulting CTE `salesPersonTotalSales` is to include `toCurrencyCode`



This is what the expect output will look like...

salespersonid	tocurrencycode	ordertotalsum	commissionpct
286	AUD	585755.800528	0.018
285	AUD	21267.336	0.0
289	CAD	1382996.5839100003	0.02
...	...	...	

Answer.

```
In [ ]: -- DELETE THIS COMMENT AND WRITE YOUR ANSWER HERE
```

CONGRATULATIONS!!!

You did it! You have completed ALL Extended Cases! I hope that you all feel more confident in your abilities to use the technologes taught in this program!

To keep your skillset sharp, we highly encourage you to further explore and utilize what you've learned by building your own portfolio. Use public datasets or collect your own data through webscraping and continue building your style of analytics.

It has been an amazing journey with you all and we wish you all the best!

Pat yourself on the back, because you did it!

Attribution

"AdventureWorks database", Nov 7, 2017, Microsoft Corporation, MIT License (https://docs.microsoft.com/en-us/sql/samples/sql-samples-where-are?view=sql-server-ver15), https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/adventure-works (https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/adventure-works)