WEEK 5 TERADATA PRACTICE EXERCISES GUIDE

The Teradata exercises for this week assume that you have completed all of the MySQL exercises, and know how to use subqueries, IF expressions, and CASE expressions. The quiz for this week will build off of the exercises below, so although I will give you hints to the answers, I will usually not give you the answers explicitly. If you can complete all the exercises listed below, you will do very well on the quiz.

Since the exercises for this week require implementing the query syntax you learned in previous weeks, there will not be a final quiz for this course, so this will be the final ungraded exercise and the Week 5 quiz will be the last assessment you need to complete. I strongly suggest that you save the queries you write during the exercises either in your Teradata account or in a text file on your computer so that you can use them for reference during the quiz. When noted, you might also want to save the output of the queries.

Please note that many of the exercises you complete this week (including those required to answer quiz questions) may take much longer to complete than those you completed last week, due to their complexity. You will want to plan accordingly. It may take you several work sessions to complete these exercises.

Specific Goals for this week's Teradata exercises

Use these exercises to:

- Practice using subqueries
- Practice using CASE statements and conditional logic
- Practice constructing longer queries in a step-wise fashion to address multiple computations or pieces of logic
- Practice translating real analysis questions into SQL queries
- Gain experience with how to handle "messy data" in SQL queries and analyses
- Gain an appreciation for how long it can take to design queries that address important analysis questions

Translating analysis questions into SQL queries

I truly think that the hardest aspect of becoming proficient at using SQL in your daily life is learning how to translate your analysis questions into, first, equations based on variables in your data, and second, into SQL queries that execute those calculations. In my opinion, that's why some people who take SQL courses never end up incorporating the skill into their job activities, even if they did very well in a SQL course. It takes practice to not only know how to turn your business problems into data variables, but to figure out how to use SQL syntax to put those variables together in the correct way. It takes even more practice to be able to use SQL to identify abnormalities in the data that can bias your interpretations of the queries you write. I am

providing the exercises below as a way for you to gain experience with the types of queries you might have to write to analyze real business data, which is messy and often poorly designed. **You should anticipate that it will take you a considerable amount of time to work through the exercises described below.** If the exercises take you longer than you expect to complete, that's ok! That means you are learning!

I found an interesting blog post that may serve as some inspiration to you as you work your way through this week's exercises. The author used the post to describe the process he used to design a complex query he needed for a company report. If you read through his example, you will be pleased to see that you now know enough SQL to understand his query and the strategy he used to write it (with one exception…he used a UNION operator, described here: http://www.techonthenet.com/sql/union.php). Nonetheless, at the end he wrote:

> *"At this point if you are not experienced in SQL report writing your head may be spinning. Yes, I made this look simple but it isn't. In all honesty this report took me about an hour to work through and what I have showed you above wasn't the first, second or even third attempt at the query."*

Source:
http://www.aranya.com/blog/Code_Elixir/The_Art_of_SQL_Report_Writing_Part_4.html.

(You might also find the strategy he describes in Part 3 useful:
http://www.aranya.com/blog/Code_Elixir/The_Art_of_SQL_Report_Writing__Part_3.html)

This blog post proves that it can take even very experienced SQL analysts an hour to write a query. So if you are struggling with an exercise below, don't worry! That's normal! The important thing if you truly want to start using SQL in your career is to persevere. Although knowing any SQL will help you as an analyst, the more SQL you know, the more job opportunities that will open up for you.

As you work your way through the exercises below, I encourage you to ask for help from your classmates or from online discussion forums. My hope isn't that you figure out everything by yourself, but rather, that you finish the course feeling confident that you know how to work through any query you need to write in the future. The business world isn't looking for analysts who know how to do everything (because such analysts don't exist!), they are looking for analysts who are persistent, eager to learn, and who find whatever resources they can to solve a problem. Use these exercises as a way to build that skillset and sense of enthusiasm for figuring things out.

General advice about how to work through complicated queries

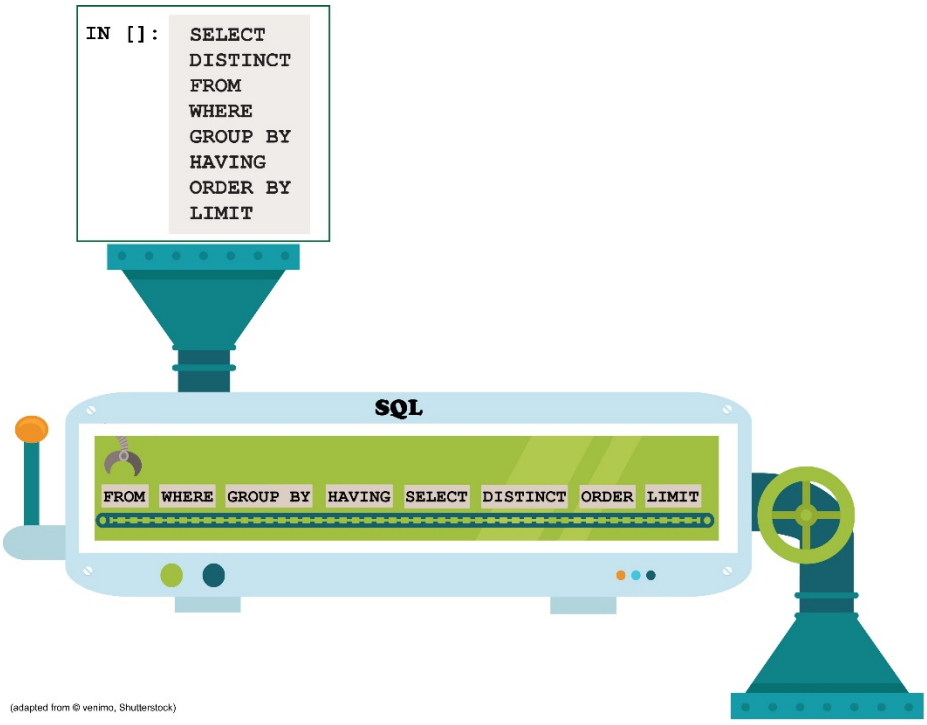
Overall, my advice for figuring out how to tackle a query is to first, use words to describe all the information you need to answer the question the query is meant to address. You might even want to write out an outline of the information and how it relates to each other on a piece of

paper. Once you can describe the information you need in words, identify the fields in your dataset that represent that information, and what tables they are kept in. Write down what tables you need to join in order to combine the fields you are interested in. After that, break the computations you need to combine into pieces, and work through each piece of a query step by step, never adding more to the query unless the steps you have already completed execute correctly.

One strategy some people use to work through a query is to start by writing all the join clauses first with SELECT * statements. When it is clear that the joins are outputting the information you intend, add your column names in the SELECT statement, and carefully check that they are aggregated appropriately. Next, add your GROUP BY statement (if needed), and finish with your WHERE/HAVING statements. As you work through the process, you will find where you need subqueries. Troubleshoot each subquery on its own, separately, before adding it to a main query.

A different strategy some people use is to write as much of a query as is possible with one table first, starting with the table from which you will retrieve the greatest number of columns. Once the query works with one table, add the next table and check the results to make sure they make sense. Keep adding one table at a time, checking the results as you progress to make sure the syntax is correct, until the query is complete.

Yet a third strategy you can try is writing the query according to the order the query will be processed. Recall this figure from MySQL Exercise 6:



When queries have a lot of nested clauses, it is easy to lose track of what gets executed first, so even if your query runs, it may give different results than you expect. If you write the clauses of your query in the same order as the clauses appear on the conveyer belt in the figure, it will help make sure the logic behind your query is reflected by the output of the query.

I suggest trying out many different query-writing strategies until you find out what works best for you. As you learn, I strongly encourage you to share advice with your peers about what you find most successful. Also, don't forget to use self-explanatory aliases, and format your queries with indentations and strategic capitalization so that they are easy to read.

Syntax error checklist

If your query is crashing and you can't determine why, make sure:

- all keywords are spelled correctly
- all keywords are in the correct order
- aliases do not have keywords in them
- quotation marks are of the correct type
- semi-colons are at the end of your query, not in the middle
- all opening parentheses are matched with a closing parentheses
- there are commas between all the items in a list
- there is NOT a comma after the last item in a list
- all text strings are enclosed with the appropriate types of quotation marks
- each column name is linked with the correct table name
- all the necessary join conditions are included for each join
- aliases and title names do not contain white spaces (unless the full title is encased in quotation marks)

Three things to know before starting the exercises:

1) Teradata does not permit the use of IF statements, but it does permit the use of CASE statements. Subqueries are run in Teradata the same way they are run in MySQL.

2) The exercises below will be organized according to analysis questions and how you might work through a data analysis, rather than what query elements are or are not required to arrive at your answer. Don't be surprised if one question requires joining 3 tables with many subqueries, while the next question simply requires a COUNT DISTINCT statement from one table. I encourage you to explore other analysis questions I left out as well.

3) There may be multiple correct query strategies for successfully completing the exercises. Feel free to use different query strategies than the ones I propose if you feel you can arrive at the same analytical answer in a different way.

What metric should we use to assess sales trends?

One of the things I talk about when describing "Structured Pyramid Analysis Plans" in the Data Visualization and Communication with Tableau course is that there are many ways to compute and operationalize a business metric, even a metric as simple as "revenue" or "profit". It is very important to think carefully about which computation will best represent the concept or phenomenon you are interested in exploring. In past weeks, we've examined Dillard's sales primarily by adding up total amounts of revenue, or when possible, by calculating the profit obtained from an item by subtracting the costs of a sku from the saleprice of that sku. These are valid approaches for assessing some questions about sales trends. Yet, the same approaches can lead to incorrect conclusions when applied to other questions about sales trends. In the exercises that follow, we are going to look at sales trends across stores and months to determine if there are specific stores, departments, or times of year that are associated with better or worse sales performance. You will perform these types of analyses often as an analyst, and from a high level view, they may appear easy to implement. It seems like all we should have to do is use a GROUP BY statement to break up sums of values in the "amt" column of our transaction table by store, department, or any other variable we are interested in. However, this approach may be misleading if used in isolation. Let's see why.

Exercise 1. How many distinct dates are there in the saledate column of the transaction table for each month/year combination in the database?

EXTRACT is a Teradata function that allows you extract date parts out of a date:
http://www.info.teradata.com/htmlpubs/DB_TTU_14_00/index.html#page/SQL_Reference/B035_1145_111A/ch07.077.147.html#ww18265377

The syntax to retrieve the month out of a date (as a number; there is no function in Teradata that will return the name of the month) is:

```
EXTRACT(MONTH from [fill in column name here])
```

The syntax to retrieve the year out of a date is:

```
EXTRACT(YEAR from [fill in column here])
```

Use these functions to answer the question in Exercise 1. Don't forget that you can't use SQL keywords like "month" or "year" as aliases. Instead of writing:

```
EXTRACT(MONTH from [fill in column name here]) AS month
```

Write something like:

```
EXTRACT(MONTH from [fill in column name here]) AS month_num
```

The first thing your answer should show you is that there are 27 days recorded in the database during August, 2005, but 31 days recorded in the database during August, 2004. Due to the fact that the August, 2005 data is curtailed (the missing days in 2005 were likely due to how the dataset was prepared when it was donated to University of Arkansas) and August is the only month that is repeated in our dataset, we will restrict our analysis of August sales to those recorded in 2004.

The second thing your answer should do is remind you that there are different numbers of days in each month of the year. If we simply add up all the sales in each month to look at sales trends across the year, we will get more sales for the months that have more days in them, but those increased numbers, by themselves, will not reflect true buying trends.

You will also notice that December has only 30 distinct sale dates (even though there are 31 days in December), November has only 29 distinct sale dates (even though there are 30 days in November), and March has only 30 sale dates (even though there are 31 days in March). That's because none of the stores have data for Nov. 25 (Thanksgiving), Dec. 25 (Christmas), or March 27 (I have no idea how to explain this one).

Exercise 2. Use a CASE statement within an aggregate function to determine which sku had the greatest total sales during the combined summer months of June, July, and August.

CASE statements can be used with aggregate functions. For example, if you wanted to find the greatest (or latest) saledate in December, you could write:

```
MAX(CASE WHEN EXTRACT(MONTH from saledate)=12
         THEN saledate
    END) AS last_day_in_Dec
```

To determine the sku that had the greatest combined total sales during the summer months, make one CASE statement that sums up all the sales in June, another that sums up all the sales in July, and a third that sums up all the sales in August. Then include another field in your SELECT statement that adds those 3 numbers together. Group your output by sku, and sort the output by the derived field that adds the revenue from the 3 months together.

Exercise 3. How many distinct dates are there in the saledate column of the transaction table <u>for each month/year/store combination</u> in the database? Sort your results by the number of days per combination in ascending order.

You will see that there are many month/year/store combinations that only have one day of transaction data stored in the database. In real life, this phenomenon could be a result of data being deleted, Dillard's stores closing during the year of 2004-2005, or data being entered in error. In our case, it may simply be the result of Dillard's removing some data before donating it. We will need to take these missing data into account in many of our future analyses, especially analyses that aim to compare sales trends within subsets of stores. How would you

examine the properties of month/year/store combinations that have a specific minimum number of days of data?

Exercise 4. What is the average daily revenue for each store/month/year combination in the database? Calculate this by dividing the total revenue for a group by the number of sales days available in the transaction table for that group.

If you write a query to examine how many distinct sale dates there are in the transaction table, and how many stores have transaction data for each of those dates, you will find that about 20% of stores have at least one day of transaction data missing. The fact that we are missing data from some, but not all, of the stores in the database will pose a challenge for us when we want to assess sales trends across different stores or geographic areas. We can't assess sales performance by simply adding up all the revenue (recall that we are also missing a lot of cost information, so we can't viably assess sales by analyzing profit), because like months that have more days in them, stores that have more data recorded from them will artificially appear to have better sales performance. We also don't want to just exclude all the information from stores that have some missing data, because 20% is a lot of data to throw away, and the stores with missing data may have some common features that would skew our results if they were excluded. Therefore, we would like to come up with a way to assess sales trends that takes the number of days of data we have into account. Somehow we need to standardize the data from each store so that, if we choose to include data from a store, it will not be affected by the number of days of data we have. For all of the exercises that follow, unless otherwise specified, we will assess sales by summing the total revenue for a given time period, and dividing by the total number of days that contributed to that time period. This will give us "average daily revenue".

Write a query that calculates the average daily revenue for each store/month/year combination. Make sure you divide by the number of days of data present **for each individual combination** rather than by how many days are usually present in a month (you can determine the number of days of data for a combination using a COUNT (DISTINCT salesdate) clause.) Check that your query excludes "return" transactions. Use the discussion boards to confirm if this first query is correct, because we are about to adapt it to take the missing data we learned about in Exercise 2 into account, and we will use the adapted query in subsequent exercises (and quiz questions).

To begin cleaning our query results, modify the query you wrote above to assess the average daily revenue for each store/month/year combination with a clause that removes all the data from August, 2005. There are many ways to do this, but I will remove the data from August, 2005 in the examples I give you using a CASE statement in the SELECT clause, and a filtering expression in the WHERE clause.

Next, we only want to include data from store/month/year combinations that have enough data to justify taking an average. Given the data we have available in our data set, I propose that we only examine store/month/year combinations that have at least 20 days of data within that month. To implement this, modify the query you wrote above to assess the average daily revenue for each store/month/year combination with a clause that excludes all store/month/year

that have less than 20 days of data. You can do this in at least two ways. The first is by including a HAVING clause to limit your data. The second is by using a subquery to say that you will only examine data that is included in the list of store/month/year combinations that have at least 20 days of data associated with them. The first method will be easier for you to implement now, but may make it more challenging for you to write subsequent queries. The second method requires you to learn another aspect of SQL syntax, but may be easier to track and implement in future exercises.

In order to implement the subquery approach to removing bad data, the data that will be removed will always be removed at the level of a year/month/store combination. That means all 3 of those variables will need to be included in the SELECT and GROUP BY clauses of the subquery. Long, complicated logical statements would be needed to exclude only specific pairs or combinations of year/month/store combinations. You can reduce the amount of logical statements needed if you combine the year and month values into a single text string. The syntax for combining strings is described here:

http://blog.extrobe.co.uk/blog/2015/02/13/concatenating-strings-in-teradata-and-what-to-watch-out-for-with-fixed-length-fields/

You could make a single label to describe the year and month information of a year/month/store combination by writing:

```
EXTRACT(YEAR from saledate)||EXTRACT(MONTH from saledate)
```

Knowing this, try adapting the query you wrote to calculate the average daily revenue for each store/month/year combination so that it has a clause that says, in essence, "only include this data in my calculation if the date-month label of this group of data in my store is present in the list of date-month labels that have at least 20 days of data in this store." Achieve this using a subquery to say that you will only examine data that is included in a list of store/month/year combinations that have at least 20 days of data associated with them. Use the discussion boards to help each other if you have trouble, because the answers I provide with the quiz (which you will see only after answering a quiz question correctly) will show you some examples that use this strategy to remove "bad data".

Save your final queries that remove "bad data" for use in subsequent exercises. From now on (and in the graded quiz), **when I ask for average daily revenue:**

- Only examine purchases (not returns)
- Exclude all stores with less than 20 days of data
- Exclude all data from August, 2005

One situation where we will NOT exclude data is when we are looking at the sales performance associated with individual merchandise items (which each have their own sku number). Since different stores will sell different numbers of items at different times anyway, we have no reason for assuming the missing data will disproportionately affect any one sku. If we were forced to analyze a data set with this much missing data in real life, I would recommend assessing the sales performance of individual sku numbers by summing the total revenue

associated with each sku, but also including a disclaimer in your results summary that informs your audience about the data that are missing. We will do the same in our analyses for this course.

How do the population statistics of the geographical location surrounding a store relate to sales performance?

One set of questions it would be beneficial to know the answers to is whether the characteristics of the geographic location in which a store resides correlates with the sales performance of the store. If it does, you can design strategies that will allow Dillard's to take advantage of the geographic trends, or make decisions about how to handle geographic locations that consistently have poor sales performance.

Exercise 5. What is the average daily revenue brought in by Dillard's stores in areas of high, medium, or low levels of high school education?

Define areas of "low" education as those that have high school graduation rates between 50-60%, areas of "medium" education as those that have high school graduation rates between 60-70%, and areas of "high" education as those that have high school graduation rates of above 70%.

To answer the question, start by incorporating the store_msa table into your average daily revenue query from Exercise 4. Then write a CASE statement using just the store_msa table that creates the education groups described above. Once you are sure both of those pieces work independently, combine the CASE statement and the average daily revenue query using a subquery strategy. Importantly, make sure you compute the average daily revenue for an education group by summing together all the revenue from all the stores in that education group, and then dividing that summed total by the total number of sales days that contributed to that total. Calculate the total number of sales days by summing together the results of a COUNT(DISTINCT salesdate) clause in your average daily revenue query.

Writing queries so that they only take the average of a population of numbers once reduces the influence of outliers, a phenomenon I discussed more generally in MySQL Exercise 11. This blog post explains why taking averages of averages often leads to misleading results:

http://geekswithblogs.net/darrengosbell/archive/2014/07/28/the-perils-of-calculating-an-average-of-averages.aspx

Stephanie Coontz provided an amusing example of how averages are sensitive to outliers in her article "When Numbers Mislead":

> *In 2001, the average income of the 7,878 households in Steubenville, Ohio was $46,341. But if just two people, Warren Buffet and Oprah Winfrey, relocated to*

> *that city, the average household income in Steubenville would rise 62 percent overnight, to $75,263 per household.*
> Source: http://www.nytimes.com/2013/05/26/opinion/sunday/when-numbers-mislead.html?_r=0

Since calculating medians in Teradata or MySQL is outside the scope of this class, our best strategy for reducing the effects of outliers on our average calculations is to calculate averages as few times as possible when calculating summary values. Thus, whenever I ask you to calculate the average daily revenue for a group of stores in either these exercises or the quiz, do so by summing together all the revenue from all the entries in that group, and then dividing that summed total by the total number of sale days that contributed to the total. Do not compute averages of averages.

Exercise 6. Compare the average daily revenues of the stores with the highest median msa_income and the lowest median msa_income. In what city and state were these stores, and which store had a higher average daily revenue?

You can answer this question by adapting the query from Exercise 5 to include a third table and a subquery with an IN clause. Recall that median msa_income is a column in the store_msa table.

Subquery Nuances

One aspect of subqueries I didn't emphasize in the MySQL exercises is that you cannot use either LIMIT (MySQL only), TOP (Teradata only), or ORDER BY operations in subqueries positioned withn SELECT, WHERE, or HAVING clauses. The only situation in which you can use LIMIT, TOP, or ORDER BY operations in a subquery is when the subquery is used in a FROM clause that makes a derived table. So, for example, the following query:

```
SELECT s.city, s.state
FROM trnsact t JOIN strinfo s
                 ON t.store = s.store
WHERE t.sku IN (SELECT TOP 3 DISTINCT sku
                FROM skstinfo
                ORDER BY sprice DESC)
```

Would give one of these errors in SQL Scratchpad:

> Syntax error: ORDER BY is not allowed in subqueries.
> TOP N Syntax error: Top N option is not supported in subquery.

However, you could retrieve the skus of the entries with the highest sprice in a derived table:

```
SELECT DISTINCT top3.sku, top3.city, top3.state
FROM (SELECT TOP 3 t.sku, s.city, s.state
```

```
FROM trnsact t JOIN strinfo s
                    ON t.store = s.store
ORDER BY t.sprice DESC) top3
```

Exercise 7: What is the brand of the sku with the greatest standard deviation in sprice? Only examine skus that have been part of over 100 transactions.

I introduced the concept of standard deviation in MySQL Exercise 11. The standard deviation is a measure of dispersion (https://en.wikipedia.org/wiki/Standard_deviation). It will give you a sense of how much a value varies. Financial analysts, for example, use standard deviations to assess how variable the price of an asset, such as a stock, is, which in turn gives them an idea of how risky it would be to invest in that asset. Similarly, calculating the standard deviation of the price of a sku will give you an idea of the variability of the sale prices of that item. A high standard deviation could indicate that the item often needs to be put on sale (the price lowered from the normal selling price) for customers to buy it (which could indicate that the original price set by the manufacturer or retailer is too high), or that it has been discontinued. It could also indicate that stores in different parts of the country have priced the item very differently, either intentionally due to marketing strategies tailored for specific geographic locations, or due to error. Of course, high standard deviations could also simply indicate that there are many mistakes in the database for the sku you are examining. When this is the case, examining items with high standard deviations can be an effective way to quickly identify parts of your dataset that need to be cleaned. Overall, it's often fruitful to investigate values with high standard deviations in your data to learn more about sales phenomena you might be able to improve, or to help you efficiently clean your data.

Teradata provides the STDDEV_SAMP and STDDEV_SAMP functions (instead of the STDDEV function in MySQL). The documentation for the functions can be found here:

http://www.info.teradata.com/htmlpubs/DB_TTU_14_00/index.html#page/SQL_Reference/B035_1145_111A/Aggregate_Functions.082.220.html

http://www.info.teradata.com/htmlpubs/DB_TTU_14_00/index.html#page/SQL_Reference/B035_1145_111A/Aggregate_Functions.082.232.html#ww594250

To address the question in this exercise, you will need to use a subquery. The fact that you need to retrieve the "maximum" or "top" value gives you a hint of what part of the outer query you will likely want to place the subquery.

Exercise 8: Examine all the transactions for the sku with the greatest standard deviation in sprice, but only consider skus that are part of more than 100 transactions.

Do you think the original sale price was set to be too high, or was just right?

Analyzing monthly (or seasonal) sales effects

It is common knowledge in the retail industry that sales are consistently higher or consistently lower during specific months of the year. It is important to know when these months are, first, so that a company can design their marketing and inventory strategies appropriately, and second, so that they can be taken into account when a company assesses its sales performance. It may make more sense to compare the sales during a given month to the sales in the same month of a previous year, rather than to the sales of a different month of the same year, for example. We will use the next set of exercises to assess the monthly sales trends in our Dillard's data set.

Exercise 9: What was the average daily revenue Dillard's brought in during each month of the year?

Adapt the query you wrote in Exercise 4 to answer this question.

Comparing monthly sales effects in our data set is tricky because of the data we are missing from about 20% of our stores. I suggest you try calculating sales in several different ways for this exercise (by excluding more or less data, taking different types of averages, etc.), in addition to using the approach we designed in Exercise 4. No matter what sales metric you use (as long as you exclude data from August, 2005 from your calculation), you should find that December consistently has the best sales, September consistently has the worst or close to the worst sales, and July has very good sales, although less than December. Just as you would if you were trying to make recommendations about seasonal marketing strategies or what inventory to have at different times of the year, we will ask questions about these consistently high- and low-performing months in the exercises that follow.

Exercise 10: Which department, in which city and state of what store, had the greatest % increase in average daily sales revenue from November to December?

Percent change, rather than raw change in revenue or profit, is a common method used to operationalize and examine changes in sales trends. Percent change takes into account how much an item cost in the first place.

Adapt the strategy you learned in Exercise 2 and the query you wrote in Exercise 4 to answer this question. You will need to join 4 tables to arrive at your answer. Use two CASE statements within an aggregate function to sum all the revenue November and December, separately. Use two CASE statements within an aggregate function to count the number of sale days that contributed to the revenue in November and December, separately. Use these 4 fields to calculate the average daily revenue for November and December. You can then calculate the change in these values using the following % change formula: *(X-Y)/Y)*100. Don't forget to exclude "bad data" and to exclude return transactions.

Exercise 11: What is the city and state of the store that had the greatest decrease in average daily revenue from August to September?

Adapt your query from exercise 10 to answer this question. You will only need to join 2 tables to arrive at your answer this time. Begin by removing one table at a time, with its accompanying field names that are no longer needed. Make sure each reduced query works before making other changes. Then change your CASE statements to refer to August and September instead of November and December. Then change your derived field to calculate average daily revenue instead of % change in average daily revenue. Make sure your bad data remain excluded, and that you are only looking at purchase transactions.

Exercise 12: Determine the month of maximum total revenue for each store. Count the number of stores whose month of maximum total revenue was in each of the twelve months. Then determine the month of maximum average daily revenue. Count the number of stores whose month of maximum average daily revenue was in each of the twelve months. How do they compare?

An important question to ask when you are examining any kind of sales trend is "how reliable is this trend?" As I said in MySQL Exercise 12, whenever you see a big effect in your data that inspires you to make or recommend a business decision, it's a good idea to see if that effect replicates in different situations or samples of data. One way to do that, which we implemented in our Dognition analysis, is to see whether the effect replicates across different years. Since we only have 1 year of Dillard's data, we will have to use a different strategy to determine whether monthly sales trends are consistent. Here, we will determine whether monthly sales trends are consistent across stores.

To address this question, you will need to make use of a Teradata function called ROW_NUMBER. Refer to these webpages to learn how to use this function:

http://forgetcode.com/Teradata/1797-Difference-ROW-NUMBER-Vs-RANK-functions
http://www.teradatawiki.net/2013/10/Teradata-Ranking.html.

(The latter reference is more comprehensive, but describes the RANK function; you can substitute the ROW_NUMBER function for the RANK function in all the examples they provide). ROW_NUMBER allows you to sort a set of rows and assign a ranked number to each of those sorted rows. Thus a strategy we could use in this exercise is:

1) calculate the average daily revenue for each store, for each month (for each year, but there will only be one year associated with each month)
2) order the rows within a store according to average daily revenue
3) assign a rank to each of the ordered rows
4) retrieve all of the rows that have the rank you want
5) count all of your retrieved rows

This will tell you how many stores have their maximum (or minimum, or whatever rank you want to examine) daily sales rates in a month.

To build this query, once again start with your query from Exercise 4 that already excludes all the data you should exclude. Adjust whatever version of the query you saved so that it outputs average daily revenue for every store/month combination (as long as you excluded data from August, 2005, each month will only be associated with one year in our dataset). Make sure you are only looking at purchase transactions. Then add in a ROW_NUMBER clause to retrieve a column with a number representing the rank of each row within a store according to its average daily revenue. Make sure you "partition" by store in your ROW_NUMBER clause. Then check your output to make sure it makes sense. When you have confirmed that the output is reasonable, introduce a QUALIFY clause (described in the references above) into your query in order to restrict the output of the query to rows that represent the month with the minimum average daily revenue for each store. You can use a subquery strategy to count how many of these rows report that the minimum average daily revenue for a store was in a given month.

You might find it helpful to rename the month numbers outputted by the EXTRACT function to a label you can more easily interpret. The best way to do that in Teradata (since there is no function like MySQL's MONTHNAME function) is to use a CASE statement.

I will be asking you more than one question in the quiz that require you to take this general query approach, so make sure you understand each aspect of your queries, and save at least one version of the successful queries you write.

A note about missing data

If you encounter a data set in your professional life that is missing as much data as we are missing in our Dillard's data set, I would recommend taking steps we do not have an opportunity to take here. In particular, do everything you can to find out why those data are missing, and to fill in missing data with other sources of information. You could also try to take advantage of other data sets to help you make statistical guesses about what the sales would likely have been, or to standardize your calculations more accurately (you could calculate revenue on an hourly basis if you knew how long each store was open on each day, or on a sqare-footage basis if you knew how large each store was, for example). The most important things to take away from the strategies we used in these exercises are that: (1) real data sets are messy and it is your job as an analyst to find that mess so that you can take it into account, (2) you can design your SQL queries to accommodate data anomalies, and (3) avoid using averages of averages to summarize the performance of a group.

Syntax error checklist

If your query is crashing and you can't figure out why, make sure:

- all keywords are spelled correctly
- all keywords are in the correct order
- aliases do not have keywords in them
- quotation marks are of the correct type (note that when you paste code you worked on in a text document outside of the Teradata interface into SQL Scratchpad, sometimes the quotation marks do not translate appropriately and need to be corrected)
- all text strings are enclosed with the appropriate types of quotation marks
- semi-colons are at the end of your query, not in the middle
- all opening parentheses are matched with a closing parentheses
- there are commas between all the items in a list
- there is NOT a comma after the last item in a list
- each column name is linked with the correct table name
- all the necessary join conditions are included for each join
- aliases and title names do not contain white spaces (unless the full title is encased in quotation marks)

Test your Understanding

Take as much time as you need to understand how to write the queries described in the exercises above. Ask questions, and help each other. Save your queries as you go so that you can refer to them later. Test your understanding of the queries and the motivation behind them using this week's graded quiz once you feel you fully understand how to do the following:

- Make "on the fly" calculations and test membership in a group using subqueries
- Calculate or retrieve data at a different level of aggregation than a main query using derived tables
- Implement if/then logic using CASE statements
- Calculate average daily sales revenue using CASE statements within aggregate functions
- Assess the consistency of sales trends across stores using the ROW_NUMBER function
- Calculate average daily revenue without using "averages of averages"