Assignment 4

Instructions

1. Some parts of this assignment require you to use `SQL_Wedge.db`. Other parts ask you to use tables within the `umt-msba` project on GBQ and the `wedge_example` or `transactions` data sets.
2. Complete each of the queries described below. You will write your queries in DB Browser. When you've completed a query and feel confident it's returning the correct results, paste it into a file called `module-4-assignment.sql`. Feel free to use comments (`--` or `/* */`) to add commentary, or ask me questions.
3. Additionally, take a screenshot of the visible portion of the results set in DB browser. Paste these screenshots into a document called `module-4-results.pdf`.
4. Submit both `module-4-assignment.sql` and `module-4-results.pdf` into the assignment on Moodle.
5. For feedback you will receive a copy of your `.sql` file. Each query will be marked "complete" or "incorrect" and the `.sql` file will be returned to you. Part of the learning process is trying to figure out *why* your query was marked "incorrect", but feel free to ask in class or office hours.

Assignment

4.1 Creating a table

In the SQLite database, write a statement which creates an empty table with three columns called "year", "description", and "sales" with types integer, text, and numeric respectively. Include a line before the create statement that drops the table if it exists. Call your table `product_summary`. Paste a screenshot of the successful query log.

4.2 Filling the table

Write a statement or set of statements that inserts the following rows in your table:

year	description	sales
2014	BANANA Organic	176818.73
2015	BANANA Organic	258541.96
2014	AVOCADO Hass Organic	146480.34
2014	ChickenBreastBoneless/Skinless	204630.90

Select all rows from your table for your "results" file.

4.3 Updating the table

Write a statement that updates the year for the row containing the avocado sales to 2022.

Select all rows from your table for your "results" file.

4.4 Deleting from the table

Write a statement that deletes the oldest banana row from the table.

Select all rows from your table for your "results" file.

The following questions use GBQ

4.5 Top Departments in 2015

Using the table `umt-msba.wedge_examples.department_date` in GBQ, write a query that returns `department` and a column called `dept_spend`, which should be the sum of spending in the department in 2015. Join the department name from `departments`, so that your query returns three columns. Sort the results from highest spend to lowest.

(This is a repeat of 2.2, translated to GBQ.)

4.6 Owner, year, month query

Using the table `umt-msba.wedge_example.owner_spend_date`, write a query that returns the following columns: `card_no`, `year`, `month`, `spend`, and `items`. Those last two columns should be the sum of the columns of the same name in the original table. Filter your results to owner-year-month combinations between $750 and $1250, order by spend in descending order, and only return the top 10 rows.

(This is a variation of 3.1, translated to GBQ. We don't make our own tables here because this server is a shared resource, so we don't want to be littering it with lots of tables.)

4.7 Exploring Transactions

As mentioned in the lecture, GBQ allows you to query a series of tables using the wildcard operator. In this query we explore a single one of these tables, which are in the `transactions` dataset. I've included a data dictionary for these types of tables in the assignment.

For this query, use the table `umt-msba.transactions.transArchive_201001_201003`. Write a query against this table with the following columns:

- The total number of rows, which you can get with `COUNT(*)`
- The number of unique card numbers
- The total "spend". This value is in a field called `total`
- The number of unique product descriptions (`description`)

One of these return values is going to look pretty weird. See if you can figure out why it's so weird. Type the answer in a comment in your `.sql` file.

4.8 Querying across transaction tables

Repeat **4.7** but now query across all transactions and report the results at the year level. Your query should return 8 rows.

4.9 The Big Transactions Query

Note: this question is the toughest of this assignment, but you'll need it to complete the next query.

You may have noticed that we've been working with two different types of data while we've been doing this Wedge work. The first kind of data, which we saw in our `SQL_Wedge.db` file, is summary data, giving us information like spend, transaction counts, and items.

The other kind of data are the raw transactions, which are stored in tables like `umt-msba.transactions.transArchive_201001_201003`. As we saw above, naive querying of these transaction tables can lead to weird results. Again, I've included a data dictionary for these types of tables in the assignment.

You're now ready to start writing queries that produce the summary results we see in the first kind of data. In this query you'll write a query that returns the spend, transactions, and items by year. I'll start by telling you how to form the columns in the `SELECT` statement:

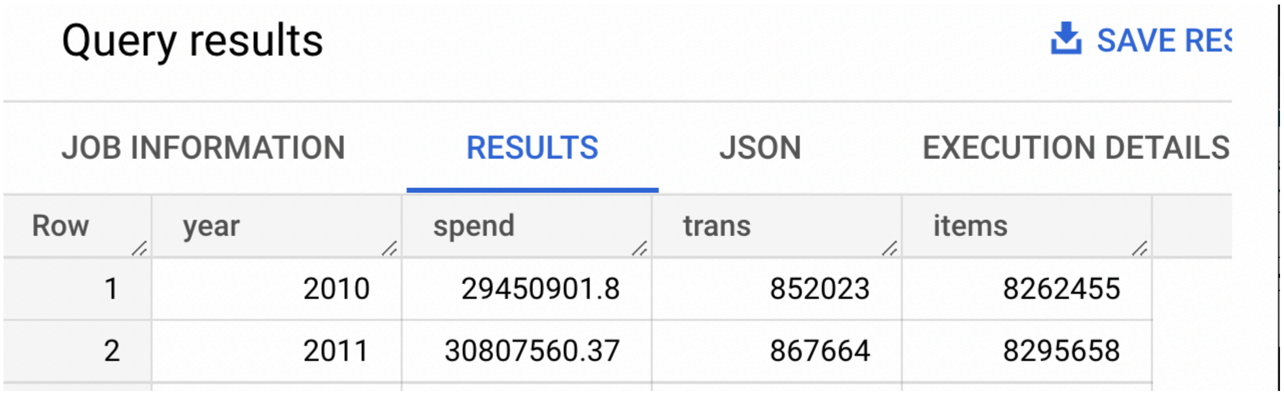
- `year` : you can extract this from `datetime` , as you may have figured out above.
- `spend` : for this you can just sum the `total` column. We'll need to use the `WHERE` clause to make this column more accurate.
- `trans` : unfortunately, there is no unique receipt ID. If there were, we could just `COUNT(DISTINCT receipt_id)` and be done. Instead, you'll need to make a unique identifier and *then* count the distinct values of that. You can make a unique identifier by concatenating (using `CONCAT`) the following fields. Each of these fields will need to be `CAST` to a string:
    - The date, found by extracting `DATE` from `datetime`
    - `register_no`
    - `emp_no`
    - `trans_no`
- `items` : we will use the `WHERE` clause to restrict the rows to "actual items", but we do have returns and voids in the data set, so we can't just count item rows. Instead, use the trick from *Become a SELECT Star* and `SUM` a `CASE` statement. When `trans_status` is `V` or `R` , yield -1. For all other `trans_status` values yield 1. Summing these values will give you a correct item count.

As I mentioned above, the `WHERE` clause is going to do a lot of the work here. We need to cut down the reciept to just rows that correspond to items. (In other words, we want to avoid rows like the tax, the payment, the change given, member discounts, etc.) Here are some things your `WHERE` clause should include:

- Exclusion of department numbers 0 and 15
- Exclusive inclusion of `trans_status` values of `NULL` or the values `" ", "V", "R"` . (That first one is a string made up of a single space; the next two are for voids and returns respectively.)

Good luck!

P.S. To help you along the way, the following images holds the first row of the solution (unless I've made a mistake).


Query results

Row	year	spend	trans	items
1	2010	29450901.8	852023	8262455
2	2011	30807560.37	867664	8295658

Note that these results might not match what's in the DB, since some of those tables were built with less accurate versions of the query described here.

4.10 Date Hour

That previous query built summary information at the year level. In this one we mimic the `date_hour` table in the database. Repeat the previous query tallying the spend, transactions, and items by date and by hour.