Wedge Data Engineering Project

Applied Data Analytics

John Chandler

m: 406.544.8720

e: john.chandler@business.umt.edu

# Introduction

This project is a view into the world of data engineering. As a graduate of the MSBA, you are going to spend a ton of your time trying to shape up data sets for analyses. It’s literally 80% of the job. A deep understanding of data pipelines and how raw data is transformed into consumable data is critical to a successful career. As you work with data you will often be called upon to define the data sets you need for your work. This project hones those skills.

The Wedge Co-Op is the largest co-operative grocery store in the US and is in Minneapolis, MN. Through a partnership with the co-op, we have data dating back to January 1, 2010 from the point-of-sale (POS) system that the Wedge developed[[1]](#footnote-1). We have data through January 2017. This system logs every row of every receipt. While we do not have other sources of data such as ordering, inventory, and spoilage, this transaction-level view is one of the richest data sets available to us. The source of this richness is that the Wedge is a member-owned co-operative. The public is welcome to shop there, but membership is lightweight: $80 (which can be paid in installments) buys you shares which entitle you to $3.75 off per month, other Member discounts, and annual refunds based on business profitability.

About 75% of transactions are generated by owners. For those owners[[2]](#footnote-2), we have a full record of their shopping at the co-op and can understand the changes to their consumption over time and can add dimensions like products and departments shopped.

On the other hand, having raw POS records comes with its own challenges. For instance, the transaction records contain many records that are not item purchases like payment, tax, discounts, members rounding up for charity and change. So, the data is valuable, but we’re going to have to do some work.

The raw zip files can be found here: <https://tinyurl.com/ada-wedge-files>. The zip of zips is pretty large, around 2.6 GB. Unzipped I believe the data is around 15 GB.

# Data Format

The transaction files contain many columns. Many of these we won’t need to use for our purposes, but there is a comprehensive list in the first appendix.

The data are stored in delimited text files. Unfortunately, some files are delimited with commas, some are delimited with semicolons. Blank fields are typically given the value of “NULL” but the Wedge occasionally uses “\N” or “\\N”. The files cover one or three months. Early on the Wedge was splitting out inactive owner transactions, but in 2015 the switched to keeping those in the same file.

You will see both regular owner numbers and non-owners (card\_no==3) in these files.

# Task 1: Building a Reporting Database in Google Big Query

[Google Big Query](https://bigquery.cloud.google.com/) is a distributed data warehouse built on a serverless architecture[[3]](#footnote-3). We’ll discuss this framework in class. In this task you’ll upload all Wedge transaction records to Google Big Query. You’ll want to make sure that the column data types are correctly specified, and you’ve probably handled the null values.

As we get closer to the end of the assignment, I’ll give you a query to run against your data to verify accuracy. You’ll submit a screenshot with the results of the query.

Note: this assignment can be done manually or programmatically. Naturally I’d prefer it be done programmatically so that you get more practice, but that’s not required to get full credit.

# Task 2: A Sample of Owners

These files are not easy to use in their current chronological arrangement, though having them in a large system like GBQ will solve a lot of our problems. Nevertheless, it’ll be convenient to have a local sample of owners to do work again.

This task asks you to generate a file of owners where the file contains every record for each owner. There will be more than one owner in the file, and I do not want you to include card\_no==3, which is the code for non-owners. The size of the sample is up to you, but I’d recommend shooting for a sample that’s around 250 MB. That’s big enough to be rich, but small enough to be fast. Ish.

#### Deliverable

A python script that handles the following tasks:

1. Connects to your GBQ instance.
2. Builds a list of owners.
3. Takes a sample of the owners.
4. Extracts all records associated with those owners and writes them to a local text file.

You’ll submit your code carrying out the steps.

# Task 3: Building Summary Tables

It is useful to have summary files that allow you to quickly answer questions such as the following:

* How have our sales-by-day changed over the last few months?
* What is our most popular item in each department?
* Which owners spend the most per month in each department?

The classic way to structure data to answer these questions is in a relational database. In this task, you will build the summary text files that hold this data and populate a relational database with the data.

#### Input

You will process your owner records in GBQ to build the summary tables.

#### Output

For this task, you will build a single SQLite database via Python (in a .db file) containing three tables:

1. Sales by date by hour: By calendar date (YYYY-MM-DD) and hour of the day, determine the total spend in the store, the number of transactions, and a count of the number of items[[4]](#footnote-4).
2. Sales by owner by year by month: A file that has the following columns: card\_no, year, month, sales, transactions, and items.
3. Sales by product description by year by month: A file that has the following columns: upc, description, department number, department name, year, month, sales, transactions, and items.

You will submit your Python code that builds the database and the database itself.

You are welcome to generate these tables via queries in Google Big Query, export the text files, and store them locally on your machine. Then you will need to write a Python script that creates the database, creates the tables, and fills those tables. Obviously, it’d be great to do the whole thing in Python.

#### Deliverable

The Python code that creates the summary tables. The Python code that builds the database. The actual database file (.db).

# Due Date

This assignment is due October 25th at 8A Mountain Time. That feels like a long way away, but this project can take a lot of time. I’d urge you to make regular progress each week. I’ll ~~nag~~ remind you about this.

## Appendix 1: Wedge Transaction Data Columns

1. datetime: timestamp of the transaction-row creation
2. register\_no: register for transaction
3. emp\_no: employee number for cashier
4. trans\_no: transaction number. This number counts up by day and is only unique when combined with date, register and employee.
5. Upc: Universal Product Code for the item. 0 for non-items.
6. description: product description. Includes things like Tax, Tender type, etc.
7. trans\_type: One of five values (D, G, A, T, and I). These correspond to the following types of transactions:
   * D: Departmental rings, when the cashier just selects a department for the item.
   * G: Green patch donations. This is the donation made for shoppers who bring their own bag.
   * A: Tax
   * T: Tender, the payment row.
   * I: Items, but also includes discounts.
8. trans\_subtype: There are a lot of these. Key ones include methods of payment (CK for Check, CA for Cash, CP for coupon, EF for EBT Food Stamps[[5]](#footnote-5), WC for WIC). These are often blank for other trans\_type values.
9. trans\_status: *An important field*.The field trans\_status tells us more about the types transactions. Here are the possible values:
   * Blank: The typical value.
   * M: Member discounts.
   * V: Voids
   * C: Coupons
   * 0: Honestly, I think these are supposed to be blanks but they changed from 0s at some point in February 2010.
   * R: Returns.
   * J: Juice club cards
10. department: The *number* of the department. See the next appendix for a department lookup table.
11. quantity: The purchased quantity. Beware, some items such as flowers and bulk vegetables are priced per cent and then sold in very large quantities (like 1000 for a $10 bouquet.)
12. Scale: The reading on the scale. Note that the capital here is not a typo. This is one field that weirdly has a capital first letter.
13. cost: the per-unit cost of an item *to the Wedge*. This is not uniformly populated.
14. unitPrice: the per-unit cost of an item *to an owner.* Negative for things like returns and discounts.
15. total: price times quantity. The cost of the line item. Note that this can be negative because unitPrice can be negative.
16. regPrice: The regular price of an item. May be different from unitPrice but unitPrice plus discount should be regPrice.
17. altPrice
18. tax: an indicator of whether or not the item is taxable.
19. taxexempt: mostly zero.
20. foodstamp: can the item be purchased with food stamps?
21. wicable: can the item be purchased with WIC?
22. discount: a marker of any discounts.
23. memDiscount: the member discounts on items.
24. discountable: beats me.
25. discounttype: there’s probably information in here, but I haven’t decoded it.
26. voided: I think it’s used if an item is a void or if an item was run up and subsequently voided.
27. percentDiscount: I don’t use it.
28. ItemQtty: I’m not sure what this is.
29. volDiscType: Ditto
30. volume: Ditto
31. VolSpecial: Ditto
32. mixMatch: Ditto
33. matched: Ditto
34. memType: Mostly NULL or 1, but I’m not sure what it signifies. Maybe institutional memberships?
35. staff: indicative of staff transactions perhaps?
36. numflag: A complicated bitflag that encodes a bunch of other information. I’ll add the communication on this topic to an appendix below, but it’s not critical for our purposes.
37. Itemstatus: Don’t know
38. tenderstatus: Ditto
39. charflag: Ditto
40. varflag: Ditto
41. batchHeaderID: Ditto
42. local: is the item local?
43. organic: is the item organic?
44. display: Don’t know.
45. receipt: Ditto
46. card\_no: ***This one is important*.** This is the masked owner number for the transaction. It is an integer. If the value is 3, then the transaction is for a non-owner. You’ll find some owners (like 11572) that have a huge number of transactions. These are likely other co-ops. If you are a member of, say, the Seward Co-op you can receive discounts at the Wedge. The cashier selects your co-op and the receipt is flagged as being from that co-op.
47. store: 1 for the main store and 512 for catering.
48. branch: 0 for the main store and 3 for the Wedge Table, a grab-and-go bodega they opened in January 2015.
49. match\_id: don’t know
50. trans\_id: a counter that increments the line items of a receipt.

# Appendix 2: Department Lookups

|  |  |
| --- | --- |
| **department** | **dept\_name** |
| 1 | PACKAGED GROCERY |
| 2 | PRODUCE |
| 3 | BULK |
| 4 | REF GROCERY |
| 5 | CHEESE |
| 6 | FROZEN |
| 7 | BREAD |
| 8 | DELI |
| 9 | GEN MERCH |
| 10 | SUPPLEMENTS |
| 11 | PERSONAL CARE |
| 12 | HERBS&SPICES |
| 13 | MEAT |
| 14 | JUICE BAR |
| 15 | MISC P/I |
| 16 | FISH&SEAFOOD |
| 17 | BAKEHOUSE |
| 18 | FLOWERS |
| 19 | WEDGEWORLDWIDE |
| 20 | MISC P/I - WWW |
| 21 | CATERING |
| 22 | BEER & WINE |

1. Interestingly, the Wedge has given this away to other grocery stores across North America. One of our long-term program goals is extending our co-op footprint to more partners. In 2016 we added the Missoula Community Food Co-op. Then they went out of business in 2017. ☹ [↑](#footnote-ref-1)
2. Owners are typically individuals, households, or buying clubs. [↑](#footnote-ref-2)
3. Amazon has a competitive product called Redshift. A comparison can be found [here](https://blog.panoply.io/a-full-comparison-of-redshift-and-bigquery). Microsoft’s Azure has some offerings in this space (Data Lake), but they seem a bit behind the curve at this point. [↑](#footnote-ref-3)
4. We can identify the number of items by looking for trans\_status of ‘’ or ‘ ‘ (blank string or a single space). To be completely correct you would want to remove Returns (R) and Voids (V) as well. I’ll give you a query that helps with this. [↑](#footnote-ref-4)
5. I think… [↑](#footnote-ref-5)