Overview

The following steps were performed to the data as hosted by Kaggle:

1. Download
2. Decompression
3. Data Interpretation
4. Population
5. Type refinements
6. Schema refinement

Download

The 5.6 meg zip file of training csv was downloaded from Kaggle

Decompression

The zip file expanded to 47.2 megs

Data Interpretation

Each row in the csv represents a “shopping point,” which includes demographic, geographic, and temporal information about a customer, insurance coverage options and price, and whether the insurance was purchased.

Population

Complying with RFC 4180, the csv file uses commas and line breaks to create store tabular information. This information is untyped and stored simply as strings. Relational databases including MySql allow for different types including textual and numeric to reduce memory footprint and improve computation. Initially a table with 25 columns, all strings, was created and populated with a `LOAD DATA INFILE` command.

Type Refinement

As stated, changing the types of the columns can improve storage, meaning, and retrieval. The 25 columns are discussed below using the description given by Kaggle in the order that they appear in the csv:

customer_ID

Though padded to 8 digits, 1XXXXXXX, the highest value is 10152724 with less than 100k unique values. Removing the padding could impact readability, so to maintain the initial values, this is stored using a 3-byte integral type.

shopping_pt

This contains values between 1 to 13 and is stored using a 1-byte integral type.

record_type

This indicates whether the insurance was purchased and is stored using a 1-byte integral type.

day

This stores which day the shopping point occurred on (0=Monday, 6=Sunday). This column is combined with time.

time

This stores 24 hour time of when the shopping event occured formatted as HH:MM. The time type in MySql stores times to the second for up to 838 hours. With a week only 168 hours, day of the week can be stored in the time as well.

state

This stores 2 letter state codes for 35 states and the District of Columbia. This is stored as an enum where the labels are stored once as a list and each record holds an index of the list. This column has a strong correlation with location.

location

Kaggle describes this as “Location ID where shopping point occurred.” There are 6219 unique locations but only 6906 unique state-location pairs [Appendix 1] The locations in 4 states all involve states that are adjacent or nearly adjacent that converge at major metropolitan areas [Appendix 2]. The remaining 628 locations have not been checked but location likely corresponds to major metropolitan areas that typically are contained in a single state.

group_size

This stores “[h]ow many people will be covered under the policy.” This is stored as a 1-byte integral type. There is a data quality issue surrounding this column along with age_oldest and age_youngest that will be discussed later.

homeowner

This stores “[w]hether the customer owns a home. This is stored as a 1-byte integral type with zero meaning false and all non-zero values meaning true.

car_age

This stores the “[a]ge of the … car” being insured. This is stored as a 1-byte integral type with the assumption that a car will be less than 255 years old. The oldest car is 85 years old

car_value

This stores the car’s value when it was new. The values range from ‘a’ to ‘g’ making the information non-numeric. This is stored as an enum like state. With 9 possible values, there are 765 possible age, value pairs. However only 314 pairs appear in the data.

risk_factor

This is described as “An ordinal assessment of how risky the customer is.” The values include the range 1 to 4 and ‘NA’. This is stored as a null-able 1-byte integral type with null meaning NA.

age_oldest

This stores the “[a]ge of the oldest person in … [the] group.” Assuming the same constraints as car_age, this is stored the same way.

age_youngest

This stores the “[a]ge of the youngest person in …[the] group.” This also using the format from car_age. In some cases, there are groups of size one where the oldest and youngest are do not match. This occurred both when the youngest is under 18 and over [3]. There are less than 3500 unique group_size, age_oldest, age_youngest tuples.

married_couple

This stores whether “the … group contains a married couple.” This is stored the same way as homeowner.

c_previous

This stores what C option the car is currently insured. If the car was not previously insured with any C option the value is zero. This is stored using the same type as the C option that will be discussed but nullable for the zero values.

duration_previous

This stores the number of years that the previous C coverage was in place. These values are similar to ages but may be null to represent a lack of previous C coverage.

A, B, C, D, E, F, G

For each of the options, there are 2 to 4 choices. With this each of the seven options is stored as a 1-byte integral type.

cost

This stores the cost of the policy considered or purchased. This is stored as a 2-byte integral type.

Schema Refinement

After converting each column into new types that better store the data spatially and semantically, it is important to determine primary keys, eliminate redundant information and divide information thematically.

To allow for better meaning and faster indexing, a table is made for each option of insurance with a value added for each possible value (table A has three values 0,1,2). The respective columns in the shopping points table become foreign keys.

A shopping point is uniquely determined by the pair (customer_ID, shopping_pt), which is the primary key of the table. Although Kaggle says that “Some customer characteristics may change over time,” in every case for the training data, location, group, and car information do not change between any shopping points. This means that the customer information is functionally dependent on the customer_ID without the shopping_pt. A second table is created to store the 12 columns of customer information. The primary key of this customers table is customer_ID which is a foreign key for the shopping points table.

For each customer, there is exactly one shopping point where the customer actually bought insurance. The policy information for this shopping point (including the foreign keys into the option tables) is also unique to each customer and can be moved to the customers table. This eliminated the need for the record_type table from both tables because every shopping point in the shopping points table was not a purchase point and every shopping point in the customers table is.

For marital status and homeownership, instead of booleans for each customer, additional tables could be used to simply store for which customers the properties hold. The Homeowner and Married tables both contain only one column, customer_ID, and store only the customers that meet the respective condition.

A similar approach is taken for the previous insurance coverage. Since not all customers had previously insured their vehicles, records are only stored for customers who did.

In the type refinement section, there was discussion of how certain sets of columns had far fewer combinations than there would be if the values were independent (geography, car, and family). Tables were created to store all of the unique combinations and the Customers table stores which of the combinations pertain to that customer.

Conclusion

The resulting 15 tables store the same information. An export of the database is 20% smaller than the csv, but the zip is only 5% smaller than the zip downloaded from Kaggle. Since zip and other forms of lossless compression exploit repeated information, we can confirm how repetitive the csv was. The database can now be efficiently queried to obtain statistics about the data to prepare for regression.

Appendix

Queries

Listed are some non-trivial queries run on the database whose results justify some of the design choices. Queries are written to run on the final schema presented in this report

1

Partitions the locations by how many unique states they are paired with and then finds the size of each partition

```
select count, count(location)
from (
      select location, count(distinct state) as count
      from Geography
      group by location
) as statesPerLocation
group by count
```

```
1     5588
2      578
3       50
4        3
```

2

Finds the locations with 4 states.

```
select location, GROUP_CONCAT(distinct state)
from (
      select location
      from (
            select location, count(distinct state) as stateCount
            from Geography
```

```
            group by location
      ) as statesPerLocation
      where stateCount = 4
) as locationsWith4States
join Geography using(location)
group by location
```

```
11271       PA,DC,DE,MD       (DC)
11515       TN,AR,MO,MS       (Memphis)
11836       AR,KS,MO,OK       (Fayetteville, AR)
```

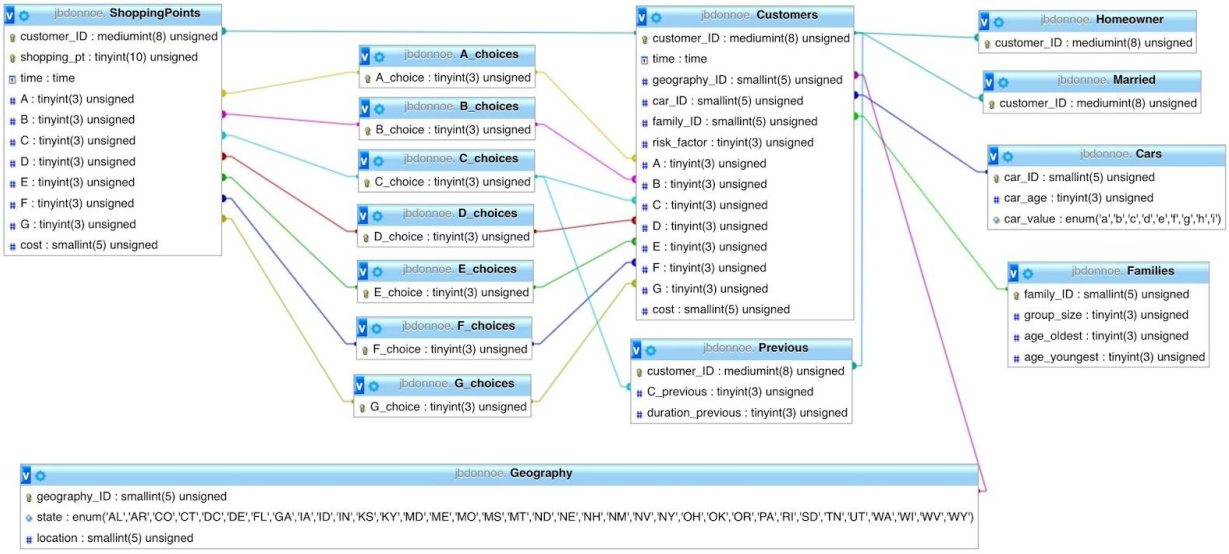
3

Finds groups of size one with distinct oldest and youngest members

```
select flag, count(0)
from (
      select age_youngest < 18 as minor
      from Families
      where
            group_size = 1 and
            age_oldest <> age_youngest
) as anomalies
group by minor
```

```
0     3858
1     282
```

Schema


jbdonnoe.ShoppingPoints
customer_ID : mediumint(8) unsigned
shopping_pt : tinyint(10) unsigned
time : time
A : tinyint(3) unsigned
B : tinyint(3) unsigned
C : tinyint(3) unsigned
D : tinyint(3) unsigned
E : tinyint(3) unsigned
F : tinyint(3) unsigned
G : tinyint(3) unsigned
cost : smallint(5) unsigned
jbdonnoe.A_choices
A_choice : tinyint(3) unsigned
jbdonnoe.B_choices
B_choice : tinyint(3) unsigned
jbdonnoe.C_choices
C_choice : tinyint(3) unsigned
jbdonnoe.D_choices
D_choice : tinyint(3) unsigned
jbdonnoe.E_choices
E_choice : tinyint(3) unsigned
jbdonnoe.F_choices
F_choice : tinyint(3) unsigned
jbdonnoe.G_choices
G_choice : tinyint(3) unsigned
jbdonnoe.Customers
customer_ID : mediumint(8) unsigned
time : time
geography_ID : smallint(5) unsigned
car_ID : smallint(5) unsigned
family_ID : smallint(5) unsigned
risk_factor : tinyint(3) unsigned
A : tinyint(3) unsigned
B : tinyint(3) unsigned
C : tinyint(3) unsigned
D : tinyint(3) unsigned
E : tinyint(3) unsigned
F : tinyint(3) unsigned
G : tinyint(3) unsigned
cost : smallint(5) unsigned
jbdonnoe.Previous
customer_ID : mediumint(8) unsigned
C_previous : tinyint(3) unsigned
duration_previous : tinyint(3) unsigned
jbdonnoe.Homeowner
customer_ID : mediumint(8) unsigned
jbdonnoe.Married
customer_ID : mediumint(8) unsigned
jbdonnoe.Cars
car_ID : smallint(5) unsigned
car_age : tinyint(3) unsigned
car_value : enum('a','b','c','d','e','f','g','h','i')
jbdonnoe.Families
family_ID : smallint(5) unsigned
group_size : tinyint(3) unsigned
age_oldest : tinyint(3) unsigned
age_youngest : tinyint(3) unsigned
jbdonnoe.Geography
geography_ID : smallint(5) unsigned
state : enum('AL','AR','CO','CT','DC','DE','FL','GA','IA','ID','IN','KS','KY','MD','ME','MO','MS','MT','ND','NE','NH','NM','NV','NY','OH','OK','OR','PA','RI','SD','TN','UT','WA','WI','WV','WY')
location : smallint(5) unsigned