Clive Unger

Computational Media Lab

Venmo Emoji Project

We started with a single json file with approximately 327 million Venmo transactions, the file size was about 400GB. Each line in the file was a json object that contained the transaction information returned from the API. Any computation on the raw dataset would take multiple hours to run because a program would have to go line by line through the massive file. Thus, we explored ways to improve the computation performance.

We started by examining the properties of the dataset we where given. Since we did not collect the data, and the Venmo API is no longer public, we had to reverse engineer which fields were important. An example of the json object is shown on the next page. Note, not every field is included in this example, but some of the common ones are. The json object has a hierarchical structure because some fields have different sizes per transactions. For example, the “comments” key has an array for its value, this array would be filled with any comments that is on the post. Next, the “actor” key has multiple subkeys that contain information about the person who created the transaction. We found that for this key in particular that older and newer transactions differed with their subkeys. In the example, “actor” has “firstname” and “lastname” subkeys. However, in newer transactions there is only a “name” subkey. We quickly realized that this json structure that the API provides was not useful for the research needs. Therefore, we decided to flatten the JSON object to store it in CSV files and eventually a MySQL database.

The flattening method that we used combines the parent keys and child keys so that there is only one level of hierarchy in the keyset. The “\_” underscore character is used as a delimiter between the parent and child keys. For example, “actor[lastname]” becomes “actor\_lastname”. However, since we need to be aware of all possible flattened keys, we had to write a script to go through each transaction, flatten they keys, and append new keys to a master list of all the possible keys. In Table 1.1, we show all the keys and their description.

**{**

"\_id"**:{**

"$oid"**:**"0123456789abcdef01234567"

**},**

"type"**:**"charge"**,**

"story\_id"**:**"0123456789abcdef01234567"**,**

"likes"**:{**

"data"**:[**

**],**

"count"**:**0

**},**

"audience"**:**"public"**,**

"payment\_id"**:**123456**,**

"message"**:**"for tea"**,**

"unix\_time"**:**1335918528**,**

"created\_time"**:**"2012-05-02T00:28:48Z"**,**

"permalink"**:**"/story/0123456789abcdef01234567"**,**

"comments"**:[**

**],**

"via"**:**""**,**

"mentions"**:[**

**],**

"transactions"**:[**

**{**

"target"**:{**

"lastname"**:**"John"**,**

"username"**:**"john\_smith"**,**

"date\_created"**:**"2010-10-17T22:34:03"**,**

"external\_id"**:**"**987654321001234567**"**,**

"is\_business"**:false,**

"name"**:**"John Smith"**,**

"picture"**:**"https://venmopics.appspot.com/u/v4/m/\*\*\*”

"id"**:**"33632"**,**

"firstname"**:**"John"**,**

"cancelled"**:false**

**}**

**}**

**],**

"actor"**:{**

"lastname"**:**"Example"**,**

"username"**:**"Alice123"**,**

"date\_created"**:**"2011-09-13T22:50:48"**,**

"external\_id"**:**"**012345678901234567**"**,**

"is\_business"**:false,**

"name"**:**"Alice Example"**,**

"picture"**:**"https://venmopics.appspot.com/u/v1/m/\*\*\*"**,**

"id"**:**"59741"**,**

"firstname"**:**"Alice"**,**

"cancelled"**:false**

**},**

"action\_links"**:{**

**},**

"updated\_time"**:**"2012-05-02T00:28:48Z"

**}**

Table 1.1: Dataset Keys after JSON flattening

|  |  |
| --- | --- |
| id | Unique identifier for transaction |
| message | Text of message for the transaction |
| created\_time | Time the transactions was created |
| updated\_time | ? |
| likes\_count | Number of likes the transaction has |
| actor\_firstname | First name of actor (initiator of transaction) |
| actor\_lastname | Last name of actor |
| actor\_name | Full name of actor |
| actor\_username | User name of actor |
| actor\_id | Unique id of actor |
| actor\_external\_id | ? |
| actor\_picture | Link to the actor profile picture |
| actor\_date\_created | Date of actor account creation? |
| actor\_cancelled | Boolean: True if cancelled, false otherwise |
| actor\_is\_business | Boolean: True if actor is business, false ow |
| actor\_friends | Boolean: True if actor is friends with target |
| actor\_num\_friends | Number of friends the actor has |
| actor\_about | Text about the actor? |
| actor\_email | Email address of actor |
| actor\_phone | Phone number of actor |
| payment\_id | Id of payment, unclear if unique or not |
| unix\_time | Time in unix format |
| permalink | Link to append to venmo url to see transaction |
| story\_id | Similar to permalink but without “/story/” |
| audience | Public/private |
| type | Charge/payment |
| via | ? this is always blank |
| transactions | Array of target information |
| comments | Array of all comments |
| likes\_data | Array of who liked the post |
| mentions | Array of mentions from comments? |
| action\_links\_iphone\_app\_store\_link\_text | ? |
| action\_links\_iphone\_app\_store\_id | ? |

MySQL Database

Instead of keeping the data in in the 400GB JSON format, we decided to move the dataset to a persistent SQL database on TACC. The schema is simply one table with all of the same fields as in Table 1.1, the “id” key is used as the primary key for the table. We also add 4 new fields to make writing queries a bit easier:

“emoji\_count”: number of emojis in message

“modifiable\_emoji\_count”: number of modifiable emojis in the message

“color\_of\_first\_modifiable\_emoji”: color of the first modifiable emoji in message

“comment\_count”: the number of comments a transaction has

The database will support any standard SQL query, however right now some queries are a bit slow, so we need to continue optimizing the performance of the database.

Overview:

Explored dataset:

Found broken json object

Found no transaction with multiple entry’s in transactions array

Found all keys after flattening

Set up SQL schema

Uploaded dataset to SQL

TODO:

Make new table in database that is smaller