Livingsocial Challenge for Software Engineer

Documentation

Design

With the given specification of the project, the problem has already been broken down into the following steps: parse file, normalize data, and store information in relational database. With PHP as a popular choice for web-based programming, I chose to use its easy file parsing and connectivity methods with my MySQL relational database.

After writing a loop to parse the text file line-by-line, I formulated the theory required to normalize the data presented in the text files by first examining the input file given, which has been reproduced below:

purchaser name	item description	item price	purchase count	merchant address	merchant name
Snake Plissken	$10 off $20 of food	10.0	2	987 Fake St	Bob's Pizza
Amy Pond	$30 of awesome for $10	10.0	5	456 Unreal Rd	Tom's Awesome Shop
Marty McFly	$20 Sneakers for $5	5.0	1	123 Fake St	Sneaker Store Emporium
Snake Plissken	$20 Sneakers for $5	5.0	4	123 Fake St	Sneaker Store Emporium

As I understand, with database normalization, the first normal form, 1NF, requires that all columns are atomic and that there are no repeating groups. These requirements would be filled by creating a separate column for each of the columns in the text file, assuming a slightly modified version of “atomic” based on our case. For example, we do not need to break the first and last names in to different columns, or break the merchant address # from its street name.

In 2NF, all non-key attributes should be dependent on only the whole key and not just one attribute. In 3NF, no attribute must be dependent on a non-key attribute. Therefore, I create a separate table for purchases, items, and merchants. If I had kept items in the purchases table, the

columns related only to items would not be dependent on the purchase keys. In order to relate purchases to items, and items to merchants, I keep a separate table for each of them, relating them by their primary keys, which are unique and auto increment in their respective tables.

Future Improvements

In this project, I assume that the item description uniquely identifies an item, whereas, in reality, this may not be the case. A different shoe store could have a coupon of "$20 Sneakers for $5," but this has not been accounted for in the current code. In the future, this could be implemented with a slight change in the implementation of MySQL tables.

Currently, I am in the process of implementing OpenID Authentication using the OpenID Enabled libraries by JanRain (http://www.janrain.com/openid-enabled). However, the code thus far requires HTTP authentication. This allows the user to enter a unique username and password, and also allows the program to gain information useful for connecting to the MySQL database. A version with no authentication required has also been provided.

Testing

The program has been tested with various forms of data that are different versions of the example input. Inside the mysql command line interface, I type "use livingsocial;" to set the current database, and use "`select * from purchases; select * from items; select * from merchants; select * from purchase_item; select * from item_merchant;`" to display the different tables that have been created. For the case of the example input, the output is as shown below:

```
mysql> select * from purchases; select * from items; select *
ase_item; select * from item_merchant;
+-------------+----------------+----------------+
| purchase_id | purchaser      | purchase_count |
+-------------+----------------+----------------+
|           1 | Snake Plissken |              2 |
|           2 | Amy Pond       |              5 |
|           3 | Marty McFly    |              1 |
|           4 | Snake Plissken |              4 |
+-------------+----------------+----------------+
4 rows in set (0.00 sec)

+---------+-----------------------+------------+
| item_id | item_description      | item_price |
+---------+-----------------------+------------+
|       1 | $10 off $20 of food   |      10.00 |
|       2 | $30 of awesome for $10 |     10.00 |
|       3 | $20 Sneakers for $5   |       5.00 |
+---------+-----------------------+------------+
3 rows in set (0.00 sec)

+-------------+-----------------------+------------------+
| merchant_id | merchant_name         | merchant_address |
+-------------+-----------------------+------------------+
|           1 | Bob's Pizza           | 987 Fake St      |
|           2 | Tom's Awesome Shop    | 456 Unreal Rd    |
|           3 | Sneaker Store Emporium | 123 Fake St     |
+-------------+-----------------------+------------------+
3 rows in set (0.00 sec)

+-------------+---------+
| purchase_id | item_id |
+-------------+---------+
|           1 |       1 |
|           2 |       2 |
|           3 |       3 |
|           4 |       3 |
+-------------+---------+
4 rows in set (0.00 sec)
```

The end product on the web browser looks like the following image, with revenue from the example input file: