♣ CloudKeeper

A
Mobile-friendly, Cloud-based service
to give a powerful shopkeeping management tool and analytics to
low ender vendors.
For free!

ISDM Lab (COE-317)
Project by
Astha Arya (242/CO/12)
Divjot Singh (262/CO/12)

Contents

Problem Statement

Data is backbone of any successful business, no matter how big or small it is. While big corporate companies rely on mainframe computers for all their data entries and analyze it using complex systems, a shopkeeper on other hand maintains his register with all daily logs in it.

With the advent of Internet revolution and cheap smartphones, a low end shopkeeper can finally afford an efficient system than those registers, which not only are difficult to maintain, but also make analyzing the data nearly impossible.

We, here are maintaining a database for a shopkeeper and providing it as a web service, so that:

- Data entry and maintenance becomes less cumbersome.
- Searching or 'Querying' for particular entries or stock items becomes user friendly and faster.
- Analysis can be done for him by the web service, suggesting him better ways to sell, order, market or invest in his products.
- The shopkeeper can easily maintain his records of items available in his shops, items he sold and the profit or loss earned.
- He can easily analyze the market growth of the products and on the basis of demand and sales from his shops, he can calculate the money to be invested in his shop and many more benefits can be achieved.
- Eventually his business grows and even then the web service can scale up, thanks to presence of efficient database systems.

Technical Description

- Programming Languages Used :
    - **Backend :** PHP, MySQLi driver for MySQL database
    - **Frontend :** JavaScript, CSS3, HTML
- Libraries Used:
    - Highcharts for charts and graphs
    - Knob.js for pie wheels
    - jQuery for AJAX, DOM manipulations
- MySQL Features exploited :
    - **Triggers :**
        - After insert on invoice_details, update owner_items.quantity for each cart item.
        - After insert on invoice_items, update invoices.invoice_amount which is equal to the sum of all quantity*price for each cart item.
        - Before update in invoice_items, check if owner_items.quantity allows the intended transaction. If not, the trigger signals the user and doesn't let the transaction commit.
    - **Functions :**
        - *getOwnerId*(invoice_id) : It is the function which is used to find the owner_id for in whose shop the passed invoice was invoked.
        - *getName*(item_id) : It is a function used to get the name of the item of passed item_id.
        - *getCostPrice*(owner_id, item_id) : It is a function used to return the cost price of the item(item_id) owned by the owner(owner_id).

- **Integrity Constraints**
  - Foreign key constraints were used to ensure data remains intact and no unexpected id can enter the database.
  - Further validations were done using PHP and JS.
- **Some queries to highlight :**
  - Area graph for revenue & profit earned by the owner over a period of months. This is generated using SELECT queries nested to accumulate the data.
  - Least/Most Sold Product, Least/Most Profitable Product, Least/Most Profitable Shop and other such analysis helps a shopkeeper make key decisions like increasing/decreasing the order quantity, investing more on a shop, etc.

Entity Relationship Models

Relational Schema

- ***owner Table***
  - **Attributes :**
    - owner_id, INT(11)
    - first_name, VARCHAR(50)
    - last_name, VARCHAR(50)
    - username, VARCHAR(50)
    - password, VARCHAR(300)
      - **Note :** The reason password length is this huge is because we are hashing the password using BCRYPT algorithm, which renders a long hex string. This secures the account and makes it resilient to hacks.
  - **Primary Key :** owner_id
  - **Candidate Keys :**
    - owner_id
    - username
- ***shops Table***
  - **Attributes :**
    - owner_id, INT(11)
    - shop_id, INT(11)
    - name, VARCHAR(50)
    - address, VARCHAR(50)
    - state, VARCHAR(50)
    - country, VARCHAR(50)
    - pin_code, VARCHAR(20)
  - **Primary Key :** shop_id
  - **Candidate Keys :**
    - shop_id
    - COMPOSITE : (address, state, country)
  - **Foreign Keys :**
    - owner_id REFERENCES owner.owner_id

- ***phonenumbers Table***
    - **Attributes :**
        - owner_id, INT(11)
        - phone_number, VARCHAR(40)
            - **Note :** Phone numbers follow several standards so it's harder to validate them all. We are using basic technique in this project.
    - **Primary Key :** COMPOSITE : owner_id, phone_number
    - **Candidate Keys :** COMPOSITE : owner_id, phone_number
    - **Foreign Keys :**
        - owner_id REFERENCES owner.owner_id
- ***items Table***
    - **Attributes :**
        - item_id, INT(11)
        - name , VARCHAR(50)
        - description, VARCHAR(250)
        - mrp, DOUBLE
        - image, VARCHAR(300)
            - **Note :** Existing version doesn't make use of images.
    - **Primary Key :** item_id
    - **Candidate Keys :**
        - item_id
- ***owner_items Table***
    - **Attributes :**
        - owner_id, INT(11)
        - item_id, INT(11)
        - quantity, DOUBLE
        - cost_price, DOUBLE
        - sell_price, DOUBLE
    - **Primary Key :**
        - COMPOSITE : owner_id, item_id
    - **Candidate Keys :**
        - owner_id, item_id
    - **Foreign Keys :**
        - owner_id REFERENCES owner.owner_id
        - item_id REFERENCES items.item_id

- ***invoices Table***
    - **Attributes :**
        - invoice_id, INT(11)
        - invoice_time, INT(11)
        - shop_id, TIMESTAMP
        - invoice_amount, DOUBLE
    - **Primary Key :**
        - invoice_id
    - **Candidate Keys :**
        - invoice_id
    - **Foreign Keys :**
        - shop_id REFERENCES shops.shop_id
- ***invoice_items Table***
    - **Attributes :**
        - item_id, INT(11)
        - quantity, DOUBLE
        - price, DOUBLE
        - invoice_id, INT(11)
    - **Primary Key :**
        - COMPOSITE : invoice_id, item_id
    - **Candidate Keys :**
        - COMPOSITE : invoice_id, item_id
    - **Foreign Keys :**
        - invoice_id REFERENCES invoices.invoice_id
        - item_id REFERENCES owner_items.item_id
            - **Note** : In our earlier ERM there was a flaw: invoice_items were referencing to items entity, which was enabling any item, owned by owner or not, to sneak into the invoices. We fixed it and now it references to owner_items.

Normalisation Analysis

Functional Dependencies :

Apart from trivial functional dependencies our database originally had following extra FD : *{item_id, owner_id} → mrp*

1. First Normal Form (1NF)

A database is in first normal form if it satisfies the following conditions:

- Contains only atomic values
- There are no repeating groups

Analysis :

a. Phone Numbers are multivalued, and are needed to be splitted into a different table : phonenumbers, having owner_id as foreign key.
b. Name is non atomic attribute in owner. It has to be split into two attributes first_name and last_name in owner table.

2. Second Normal Form (2NF)

A database is in second normal form if it satisfies the following conditions:

- It is in first normal form
- All non-key attributes are fully functional dependent on the primary key

Analysis :

a. Originally there were two FDs associated with MRP.
item_id → mrp and item_id, owner_id → mrp
and MRP was an attribute of owner_items. However this became a classic 2NF problem. In order to fix it we had to move MRP to items table so that MRP could become fully functional dependent on the primary key.

3. Third Normal Form (3NF)

A database is in third normal form if it satisfies the following conditions:

- It is in second normal form
- There is no transitive functional dependency

Analysis :

a. Our database lacks such functional dependencies and after fixing the above mentioned issues we were able to make our database 3NF compliant.

Potential Issues

- Deleting an item would affect all the related invoices. Although deleting an item would imply that such item was accidentally added or is now discontinued, but the effect of it to database is pretty huge. Such an operation must be pondered thoroughly.
- Similarly, deleting a shop would affect all invoices through that shop. This again is a highly risky operation.

Future Scope

- Intelligently auto-filling entries based on previously filled details. This would save vendor's time and help him quickly log the invoices.
- Predictions like "This item is often times bought with this one". These are already popular in online domain, and might give nice insights to the local vendor too while organizing his/her products.
- Giving tips and suggestions to maximize profit using calculations and concepts of economics (e.g. [EOQ] Economic Order Quantity, Break even point analysis etc).
- Using the data from the vendors to provide a search engine to customers to find local shops and vendors, even household items**.** This completes the other end and joins customers to vendors. Ultimately a local vendor will be able to gain traction from online audience.