CS315: Group ID - G04
Database System with Temporal Warehousing

Abhay Kumar
Student ID: 70
Roll Number: 11008
abhayku@iitk.ac.in
Dept. of CSE

Adarsh Chauhan
Student Id: 47
Roll Number: 11034
adarshc@iitk.ac.in
Dept. of CSE

Aman Kumar
Student Id: 16
Roll Number:11070
amankr@iitk.ac.in
Dept. of CSE

Indian Institute of Technology, Kanpur

Final Report
April 18, 2014

Abstract

The project implements a model of a central storage of food products in the form of a database with temporal warehousing, having characteristics of validity of data in the temporal domain.

Introduction and Problem Statement

We are given a structure of a central storage, connected with various branches or retail stores. Food products are collected in bulk in the central storage from various sources. It is from the central storage, that the branches derive their replenishment to serve the customers. The aim is to develop a database management system which could model any hypothetical structure of a set of central storage and branches, and it should be possible to manage transfers of good between storage and any store, with consistency and preserving the validity of data in the form of shelf-life of food products involved.

Algorithm or Approach

The aim could be achieved by creating a database and populating it with the following tables:

1. $T_{central\_storage}$ = (ID, product_ID, quantity, batch_no, mfg_date, *valid*, t_stamp), the **central_storage** relation which stores the information about all the products that have been brought in the central storage from the sources.

2. T_{sales} = (ID, store_ID, product_ID, quantity, t_stamp), the **sales** relation which stores the details of any transaction made by the central storage to any branch store.

3. $T_{requests}$ = (ID, store_ID, product_name, quantity, *fulfilled*), the **requests** relation which stores the details of requests made by any store to the central storage according to its demand.

4. $T_{product\_details}$ = (product_ID, product_name, rate, period), the **product_details** relation which stores essentially the rates and correspoding shelf life periods of various products.

5. T_{store_i} = (ID, product_name, quantity, batch_no, expiry_date, received_date), the **store_i** relation which stores the information about all the products that have been received by the i^{th} store from the central_storage.

For the purpose of simplicity and illustration, we have taken **one** *central storage*, **four** *branch stores* and **four** different types of food products, i.e. *Milk, Eggs, Bread, Juice*.
It is assumed that the database is meant for control and management of the food resources for buying and selling purposes by an administrator who may be the owner of the central store and/or owns responsibility of maintaining his/her store.

Our approach to manage the transactions in the modeled warehouse consists of the following features (along with their descriptions):

- **Incoming**: It is assumed that the food warehouse has sources of itself from which it procures the food products. **Incoming** is used to enter the record of products, their quantities and manufacturing dates, which are brought into the central storage.
  Two facilities are provided: either you could manually enter the products from **Central Storage** option or by **Incoming**, you could enter a bulk of products randomly to populate the **central_storage** table.
  It uses the MySQL query INSERT to insert tuples into the table.

- **Central Storage**: It displays the present contents of the **central_storage** table, which are valid in the present time.
  It uses the conditional SELECT to select valid tuples from the table, i.e. tuples which have *valid* field as 1.

- **Sell**: It provides the facility to the administrator to sell products to branch stores by two ways: either arbitrarily according to the wish and decisions of the administrator, or by looking at the requests from the branch stores and fulfilling each request accordingly.
  For the latter purpose, it displays all the unfulfilled requests from the **requests** table i.e. tuples which have the *fulfilled* field as 0.
  Every selling transaction performs the following queries:

  - It uses SELECT the tuples of the particular products which are to be sold and then uses UPDATE to update their quantity in the **central_storage** table.
  - It uses INSERT to insert the record of the selling transaction performed as a log record in the **sales** table.
  - It uses INSERT to insert the tuples into corresponding table of the branch store to which the particular food product has been transferred.
  - If the particular demands by branch stores are fulfilled using the tuples from the **requests** table, then the corresponding request is also marked as fulfilled by setting its *fulfilled* field to 1.
    It uses UPDATE query to update the fulfillment of requests in the **requests** table.

- **Stats**: In this option, the statistics corresponding to the sales made are displayed:

  - *Temporal Query* - It provides the facility to display the sales made and profit/loss incurred between a given time period.
  - It displays the all the sales made till the current date or between the selected table by using SELECT from the **sales** table.
  - It also displays the product details like rate and shelf life by fetching information from **product_details** table by using SELECT query.
  - It also displays a bar graph representing all the profit/loss incurred for each product category by using SELECT query from **sales** for counting costs incurred as profits and from **central_storage** for counting non-zero quantities of products which expired and were not sold in the given duration.
    The graph is prepared and displayed using **Google API (jsapi)**, which is a JavaScript API to create graphs, given the data in a formatted manner.

- **Store_i**: For each store, we display the current contents of all the products in the stores whose *temporal validity* is ensured by their expiry dates being greater than the current date.
  It uses SELECT query from **store_i** table to get tuples of products currently valid in the corresponding branch store (available for the customers to buy).
  It also provides the facility to send requests to the central store depending on the current storage in the central store and the demand from the customers.
  For this, it uses the INSERT query to insert the request tuples into the **requests** table.

- Another important *temporal feature* of the database is the validity of the products in the **central_storage**. It is achieved by the following ways:

  - Whenever any user logs on to the **Central Storage**, depending on the *quantity* field, if the quantity of any tuple in the **central_storage** is zero, its *valid* field is updated to zero using the UPDATE query.
  - **Home** displays the list of the products which are going to be expired in the next one, two, three and four weeks, by using the SELECT query on the **central_storage** table and by looking at the shelf life of the product from the **product_details**. It calculates the expiry date of the products and compares it with the current date.
    For all those products where the current date exceeds the expiry date, it implies the exhaustion of the validity of the corresponding product and hence, by using the UPDATE query,

the *valid* field of the corresponding tuple in the **central_storage** table is updated to zero.

Results

- The database management system works accurately and models a hypothetical food product retail warehouse as according to the expectations.

- It provides the temporal warehousing of data, mainly through the **central_storage** table, where the validity of data is decided mainly by the manufacturing date and the shelf life of any batch of a food product. This validity is represented by the *valid* field in each tuple of **central_storage**.

- The most computationally expensive queries in our implementation are: (1) the population of **central_storage** by generating random values and (2) the updation of valid field in the **Home** field where for each *valid* tuple, if the current date exceeds its expiry date, then it is marked invalid.

- The rest of the SELECT, INSERT and UPDATE queries are linear in time complexity and thus, less computationally expensive.

Conclusions

- Thus, the model of a food product retail warehouse can be modeled using a temporal warehouse database management system, where the *temporal feature* of the data is based on the expiry dates of the food products.

- Future scope includes extending the incoming of data into the central storage from its sources in a transaction-manner implementation, rather than just simply presenting it as a log-record management.

- The system of branch stores can also be further developed to modeled a branch retail store where the food products can be available for purchase by the customers being served by that particular branch.

- The entire database management system can be modeled as an online platform with membership by the administrator and local branch owners and a login-based system where these individuals can login and access certain features based on their privileges.

- Finally, the aim of the project was to focus on the temporal aspect of certain data and the implementation was targeted accordingly.
  In case of an extension to the online platform, we can focus on the concurrency of the essential tables which are accessed frequently like **central_storage** and **requests**.
  Concurrency can be implemented using standard methods like exclusive and shared locks.

References

1. Wikipedia Article on *Temporal database*: `https://en.wikipedia.org/wiki/Temporal_database`

2. A Ph.D Thesis on *Temporal Databases and Data Warehousing*: `http://www.dcs.ed.ac.uk/home/tz/phd/thesis/node11.htm`

Remarks by the Instructor

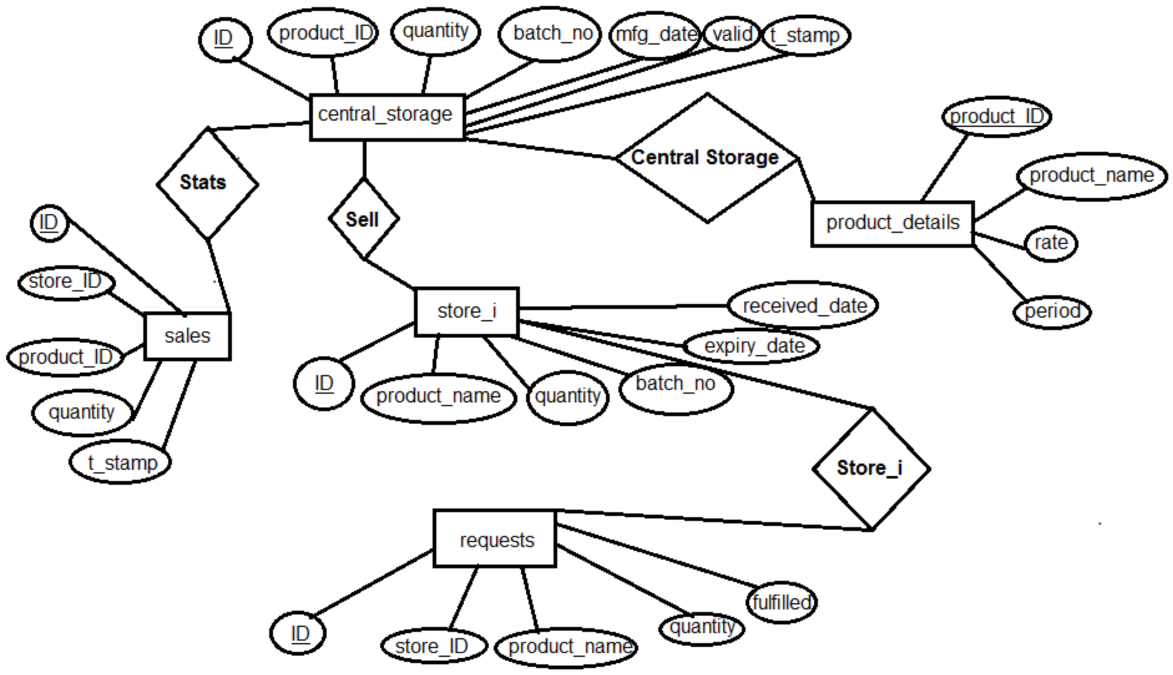
- It includes the implementation of basic features of transactions involved regarding buying and selling of food products.

- The temporal characteristic of data in the form of validity of food products has been implemented as required.

- The database also has characteristics of data consistency and no bugs were observed during inspection.

- The implementation of database system is satisfactory and complete as was desired according to the aim.

Entity Relationship Diagram of the Implemented Database System


ID
product_ID
quantity
batch_no
mfg_date
valid
t_stamp
central_storage
Central Storage
product_ID
product_name
product_details
rate
period
Stats
Sell
ID
store_ID
sales
product_ID
quantity
t_stamp
store_i
received_date
expiry_date
ID
product_name
quantity
batch_no
Store_i
requests
fulfilled
quantity
ID
store_ID
product_name