**Project Report**

|  |  |
| --- | --- |
| **Course:** | INFSCI 2711 Advanced Topics in Database Management |
| **Project Name:** | Data Warehousing Strategies for an Online Book Retailer |
| **Group Members:** | Qi Fei (feq5), Yu Qiu (yuq6), Weiqi Li (wel106), Xinying Sun (xis53), Hongya Ma (hom19), Chen Song (chs222) |

1. Introduction

In this project, we will design and implement a database as well as website for an online book retailer. This system will consist of the whole work flow from the front-end to the back-end, and have generally 2 types of users: for the customers using our website, they will be able to:

1. Register, log in and purchase;
2. Customers must register and log in before they can do any purchase;
3. Search the books we provided;
4. Select books and corresponding quantities they want to buy;
5. Add multiple products into the shopping cart and pay for them together;
6. After submitting a purchase, view the process of shipping;
7. Review their order histories.

For the salespersons in our company, they will be able to create, delete and update the data in the MySQL operating database. For the managers who care about the sales and profits, we also provide data analytic operations by building a data warehouse to do analysis and predictions through querying the data warehouse. We will build the data warehouse in 3 different platforms: MySQL, MongoDB and Neo4j.

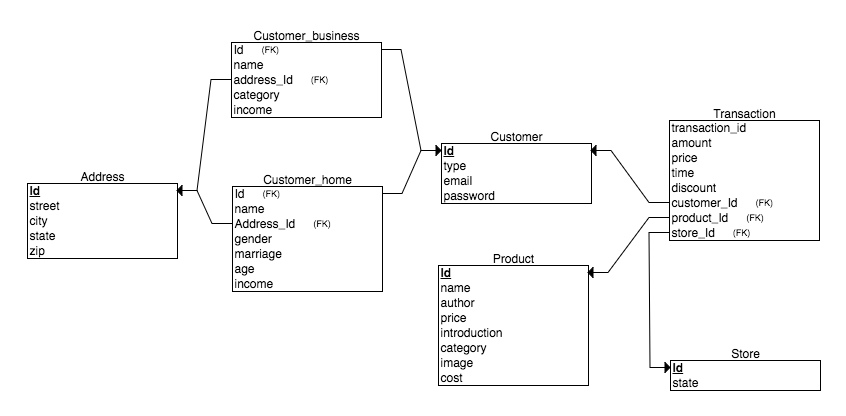
2. Assumptions

For our operating database, we have several assumptions:

1. We only sell products to registered customers. New customers need to register and log in first to do any purchase;
2. All stores will use the same inventory;
3. Customers never return any product.

3. Descriptions of Data Maintained in the System

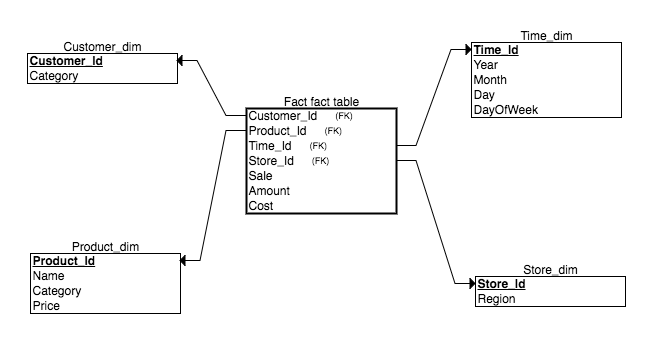
We choose MySQL to implement our operating database. There will be 7 tables in the database: customer, home\_customer, business\_customer, product, transaction, store and address. Then we build the data warehouse from the MySQL operating database. In another word, NoSQL data warehouse implements also are based on the relational operating database.



4. Star Schema Design

Our managers, as any manager in any company, care about sales and profits most. Therefore, there are 3 measurements in our Fact table: Amount, Sale and Cost. And there are 4 dimensions: Customer, Product, Store and Time. Therefore, the Fact table shows that one Customer buy one Product in one Store at one Time slot.

To load the newest data from the operating database into the data warehouse, every time when the user requests to update the data warehouse, we drop the old DW and reload the data from the operating database. We only load the schemas we care about into DW. The star schema design and DDL shows as follow:



DDL:

##Create Data Warehouse

DROP TABLE IF EXISTS `fact`;

CREATE TABLE `fact` (

`product\_id` varchar(45) NOT NULL,

`amount` int(11) NOT NULL,

`store\_id` varchar(45) NOT NULL,

`customer\_id` varchar(45) NOT NULL,

`time` varchar(45),

`sale` decimal(10,2) NOT NULL,

`price` decimal(10,2) NOT NULL,

`week` int(11)NOT NULL);

Insert into fact

select product\_id, amount, store\_id, customer\_id, time, sum(sale), price\*discount, week from transaction

group by product\_id, customer\_id, store\_id, time;

DROP TABLE IF EXISTS `product\_dim`;

CREATE TABLE `product\_dim` (

`product\_id` int(11) NOT NULL,

`name` varchar(45) NOT NULL,

`cost` decimal(10,2),

`category` varchar(45) NOT NULL);

Insert into product\_dim

select id, name, cost, category

from product;

DROP TABLE IF EXISTS `store\_dim`;

CREATE TABLE `store\_dim` (

`store\_id` varchar(45) NOT NULL,

`region` varchar(45) NOT NULL);

Insert into store\_dim

select id, state

from store;

DROP TABLE IF EXISTS `customer\_dim`;

CREATE TABLE `customer\_dim` (

`customer\_id` int(11) NOT NULL,

`type` varchar(45) NOT NULL);

Insert into customer\_dim

select id, type

from customer

5. Pre-aggregated Summary Tables

(including corresponding DDL statements). The SQL statements for creation and populating of the summary tables. Specification of nightly  scheduled batch job to summarize data.

6. Description of Data Warehouse Queries and front-ends required for the warehouse

7. example scenarios of how various types of users will interact with the system

8. description of alternative implementation of your DW system using two NoSQL platforms

**MongoDB**

The very first and the most import problem we faced when we implement the data warehouse in MongoDB is that MongoDB does not support JOIN, but building and querying the data warehouse requires a lot of JOIN between Fact table and dimension tables to do slice, dice, roll-up or drill-down. If we want to do a JOIN like operation in MongoDB, we need to query the first collection and save the results in some variables, and use the results to query the second collection, which is hard to implement, especially for some complex queries.

After research, we found that MongoDB has some forms of relations: embedded relations and reference relations. Reference relations works very similar as relational database: each object in MongoDB has an ObjectId, and we can reference any object through it ObjectId. There are 2 forms of reference relations: directly using the ObjectId or following a formet call DBrefs. By using DBrefs, we had our first version of

MongoDB implementation: we build Fact and dimensions collections which are very similar with them in the relational database, and connect them through DBrefs. DBrefs has the advantage than directly using ObjectId that we do not need to specify the collection we are referencing in the query, but it is already in the database specified by DBrefs. A document in Fact collection will look like this (left):

{

\_id: …

Customer: {

$ref: Customer

$id: ObjectId(“…”)

}

Product: {

$ref: Product

$id: ObjectId(“…”)

}

Store: {

$ref: Store

$id: ObjectId(“…”)

}

…

Amount: …

Sale: …

Cost: …

}

{

\_id: …

Customer: {

\_id: …

category: …

}

Product: {

\_id: …

name: …

category: …

price: …

}

Store: {

\_id: …

region: ...

}

…

Amount: …

Sale: …

Cost: …

}

Figure: MongoDB document in design 1

Figure: MongoDB document in design 2

This design could fulfill the functions of the data warehouse, but it still requires a lot of JOIN like operations, and ObjectId is hard to read or do any error check. Therefore, we had our second version of implementation using embedded relations: we embedded all data in dimension collections into the Fact collection. In another word, we only have the Fact collection in our database. A document will look like this (right):

This design avoided any JOIN because all the data we need is in the same collection. But it also introduces a lot of redundancy since the information of customer, product or store is repeated again and again.

After learning about XML, we had the third idea that we can make a XML style database and keep all data in the same collection but remove the redundancies. It could look like this:

|  |  |
| --- | --- |
| {  customer: {  \_id: 1  name: …  category: …  }  customer: {  \_id: 2  name: …  category: …  }  product: {  \_id: 3  name:  category:  price:  } | product: {  \_id: 4  name:  category:  price:  }  …  {  customer: 1  product: 3  …  Amount: …  Sale: …  Cost: …  }  } |

We have no time to research and implement this design, and do not know whether it can be done in MongoDB or not. But in a document oriented database, this design is possible and might have some advantages. At least it could remove the redundancy in our previous design.

After building the Fact collection using the second design, we also build some pre-aggregation collections (materialized views) from Fact for the data warehouse. For example, we build a view named “Type” to see the ration between home and business customers like this:

|  |  |
| --- | --- |
| home | Cusotmer1, Cusotmer2, Cusotmer6 |
| business | Cusotmer3, Cusotmer4, Cusotmer5 |

9. comparison of relational and NoSQL implementations with explanation of advantages and disadvantages of each approach

**MongoDB**

**Disadvantages:**

As we mentioned above, MongoDB does not support JOIN. This must not be a disadvantage for a NoSQL database platform, but data warehouse is a model developed from relational database, so it is hard to fit the data warehouse into MongoDB. In relational database, Fact table could store the primary key, like Id of every dimension table. When some queries need the join of Fact table and Dimension table, relation database could easily execute join operation through the primary key. However, in MongoDB, users need to execute more than one query operation to implement join operations. For example, if the user wants to know which region sell most products, it requires Fact table joining Store table by Store Id, and then grouping by store region in a relational database. But in MongoDB, firstly users have to aggregate total sales of each store in Fact table and record the results. Secondly, users have to union this result and Store Dimension table to get the total sales of each regions. This could be very complex and painful.

Therefore, we use embedded relations and nested documents to solve the problem, which means dimension tables could be added to the Fact table as documents. But as we mentioned above, this will introduce redundancy and blow the size of the database.

**Advantages:**

The first advantage of MongoDB comes from its flexible schema design: it allows one key has several corresponding values. Therefore, when we build the pre-aggregation views, we can do some jobs more easily. For example, when we want to query the ratio of home customer to business customer, we build a materialized view named “Type” as mentioned before, which just including two keys, “business” and “customer”. Each key could contain many customers. With this view, it will be very easy to query the ratio of different type of customers.

Another important advantage of MongoDB is its pipeline system. By using the pipeline, we can add anything into one query, no matter it is a projection, grouping or renaming. With this great function, we can process the data much easier than in MySQL: it will require complex nested query or several queries to fulfill the same function in MySQL.