Project Report

Course: INFSCI 2711 Advanced Topics in Database Management

Project Name: Data Warehousing Strategies for an online book retailer

Group Members: Chen Song (chs222)

1. Introduction

In this project, we will design and implement database and website for an online book retailer. With our implementation, customers will be able to browse different kinds of books we provided, purchase their favorite books and check their transaction records through our website. Salespersons will be able to operate MySQL operating database to create, delete and update the data in the database. Meanwhile, for business managers who desire to analyze the sales and make prediction by querying from the database, we provide data analytic operations to them by building corresponding data warehouse through different platforms, including MongoDB, Neo4j and MySQL.

2. Assumptions

3. Descriptions of Data Maintained in the System

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 5 dimensions: Transaction, Customer, Product, Store and Time. In another word, the Fact table shows that one Customer buy one Product in one Store at one Time slot in one Transaction.

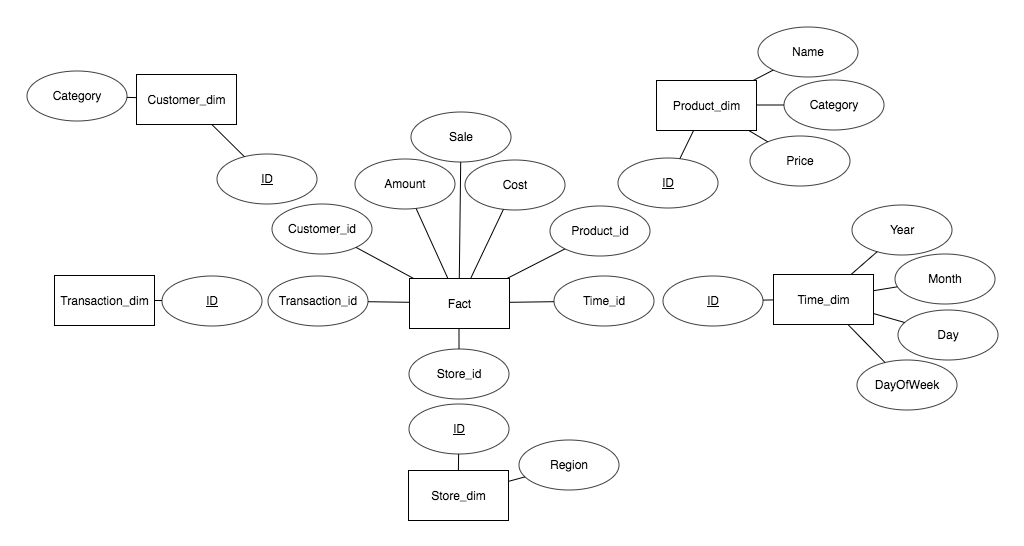


Table 1 Fact:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Transaction | Customer | Product | Store | Time | Amount | Sale | Cost | discount |

DDL:

Table 2 Product\_dim:

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Category | Price |

DDL:

Table 3 Store\_dim

|  |  |
| --- | --- |
| Id | Region |

DDL:

Table 4: Time\_dim

|  |  |  |  |
| --- | --- | --- | --- |
| Year | Month | Day | Week |

DDL:

Table 5: Customer \_dim

|  |  |
| --- | --- |
| Id | Category |

DDL:

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

8.1 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 the data warehouse we learned under relational database 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 should 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 follow a formet call DBrefs. By using DBrefs, we had our first version of MongoDB implementation: we build Fact and dimensions collections similar with the relational database, and connect them through DBrefs. DBrefs has the advantage that we do not need to specify the collection we need to reference in the query, but it is already in the database specified by DBrefs. A document in Fact collection will look like this:

{

\_id: …

Customer: {

$ref: Customer

$id: ObjectId(“…”)

}

Product: {

$ref: Product

$id: ObjectId(“…”)

}

Store: {

$ref: Store

$id: ObjectId(“…”)

}

…

Amount: …

Sale: …

Cost: …

}

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:

{

\_id: …

Customer: {

\_id: …

category: …

}

Product: {

\_id: …

name: …

category: …

price: …

}

Store: {

\_id: …

region: ...

}

…

Amount: …

Sale: …

Cost: …

}

After building the Fact collection, we also build some pre-aggregation collections from Fact for the data warehouse.

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 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.

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

9.1 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 actually build one materialized view named “Type”, which just including two keys, “business” and “customer”. Each key could contain many customers, like shown below:

Type materialized views:

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

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.