COMP810 Data Warehousing and Big Data Assessment 2 Data Warehousing Project

Building and Analysing a DW for NatureFresh Stores in NZ

Stone Fang (Student ID: 19049045)

1 Project overview

The goal of this project is to create a Data Warehouse (DW) for the sales analysis of NatureFresh, one of the largest fresh food market chains in New Zealand. Analysis of sales and customer shopping behaviours can give NatureFresh in-depth insight of the market, so they can improve their selling strategies accordingly.

The original available data are customer transactions and product information. The transaction data contains records of customer buying, including who (customer) bought what (product), when (date), where(store) and how many was bought (quantity). The product data contains information for each product, including supplier and price.

However, the format of original data doesn't fit into the requirement of OLAP, so first we need transform the data into other formats for better querying.

The major content of this project contains:

- Design and implement the star-schema for sales DW, i.e. fact & dimension tables
- Fill DW by ETL process. Specifically, do Index Nested Loop Join (INLJ) on transactions and master data, transform and load data into fact & dimension tables.
- Execute queries on DW

All the operations above are implemented in SQL.

2 Schema for DW

According to the original data, the DW will consist of one fact table *Sales* and five dimension tables *Product*, *Supplier*, *Customer*, *Store*, and *Date*, as shown in Figure 1. The SQL code to create all tables are in file *createDW.sql*.

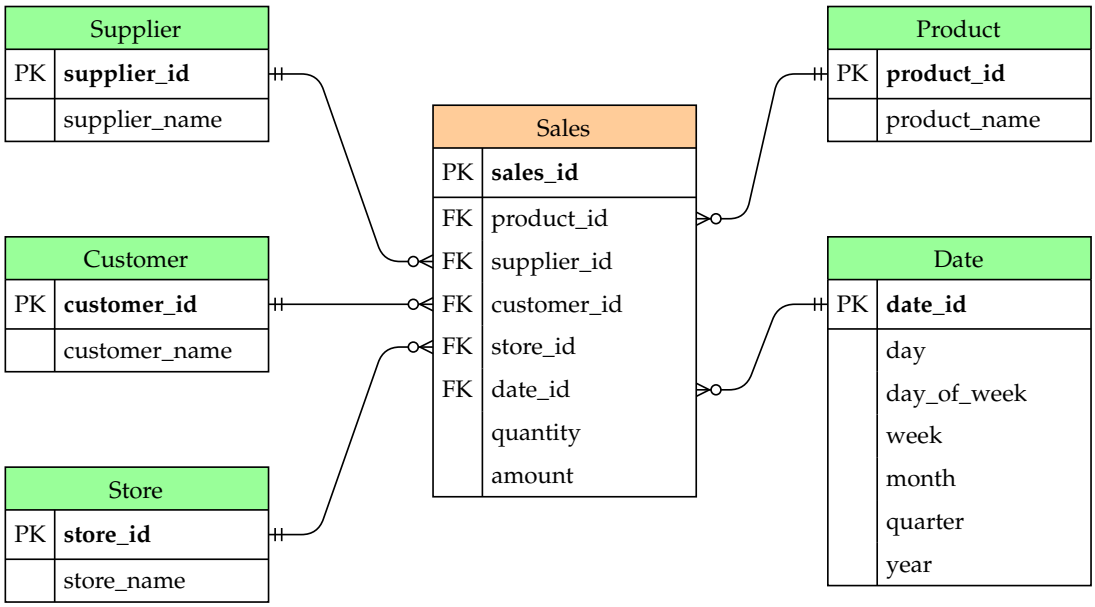

Figure 1: Star Schema of NatureFresh Sales

2.1 Fact Table

Apparently the fact table should have foreign keys corresponding to all five dimension tables, and the quantity of item sold. There are two decisions have been make for primary key and amount of money in sales.

Primary key of fact table can be a combination of all foreign keys. However, there could be a concern to have more than one transactions for the same values on all five dimensions. A quick analysis shows that such situation does exists, though the possibility is low. In other words, a customer may buy one product multiple times at one store in one day. There are two options to solve this problem. One is summing up the quantities of multiple transactions, resulting in only one record for the same combination of dimension values. The other is keep multiple transactions while use a separated ID field as the primary key of *Sales* fact table. In this project, the latter solution is preferred because this approach can keep the original granularity of transactions, thus contains more information. Also, the possibility of multiple transactions for one combination of dimensions is low, so there would not be significant overhead in terms of memory and storage.

Price/Amount is another concerning field. In the original data, *price* is stored in master data table as a property of product, so it is natural to make it an attribute of product dimension. However, this design has a shortcoming when price changes as it always does. If the price of a product changes, we can't simply modify the value in *Product* dimension table otherwise the result on sales before that change will be incorrect. Therefore, in this project price information is kept in *Sales* table. Since the amount of

money in sales is a more frequent used number, we add to fact table an *amount* filed which is calculated by *quantity* $\times$ *price*. In section 5 further discussions will be provided on this issue.

2.2 Dimensions

Details of dimension tables can be referred to Figure 1. Most dimensions are as simple as "ID+name", while the *Date* dimension is relatively complicated. First of all, unlike other dimensions, there is no existing ID for *Date*. In this project, a string in format of "YYYYMMDD" is chosen as the ID for *Date*, rather than an auto-incremental column. The advantage is such ID is more readable and intuitive, and thus more convenient for partitioning if required in the future. On the other hand, it would need more storage space, which, however, is not a big issue providing the cost storage is quite low nowadays. Second, *Date* dimension contains more information other than names. In this project, common properties are calculated, including *year*, *quarter*, *month*, *week*, *day*, *day_of_week*. In fact, it can be extended to more fields such as *is_public_holiday*, if some analysis on holiday is in demand.

3 INLJ algorithm

Index Nested Loop Join (INLJ) is a table joining algorithm that can be used for stream data joining. Nested Loop Join takes an outer loop and an inner loop, each for one table, and output the rows that matches the conditions, so the time complexity is $O(NM)$ where N and M are the number of rows of two tables. . However, INLJ only keep the outer loop and replace the inner loop with an index-based loop up, thus greatly reduce the time complexity. For example, if the index is implemented by B-tree, then complexity of lookup is a logarithm of M instead of linear which is the case of the inner loop.

This algorithm is implemented in PL/SQL. First a bulk (50 rows as in this project) of transactions is read into memories. Then all rows in the bulk are read one after another, and retrieve the information for current row from master data by *product_id*. Then all properties corresponding to current row are transformed to fit the star schema and then load into the fact and dimension tables. Please refer to file *INLJ.sql* for the complete implementation.

4 OLAP Queries Results

This section summarise the results of required analysis. The SQL statements for these queries are referred to file *queriesDW.sql*.

Question 1

> *Determine the top 5 products in Dec 2019 in terms of total sales*

Result:

PRODUCT_NAME TOTAL_SALES RANK ———— ———— —-

Question 2

> *Determine which store produced highest sales in the whole year?*

Result:

STORE_NAME TOTAL_SALES RANK ———- ———————- —-

Question 3

> *Determine the top 3 products for a month (say, Dec 2019), and for the 2 months before that, in terms of total sales.*

Result:

PRODUCT_NAME SUM(TOTAL_SALES) RANK ————- ———————- —-

Question 4

> *Create a materialised view called "STOREANALYSIS" that presents the product-wise sales analysis for each store. The results should be ordered by StoreID and then ProductID.*

Result:

STOREID PRODUCTID SUM(STORE_TOTAL) —— ———- ———————-

Question 5

Think about what information can be retrieved from the materialised view created in Q4 using ROLLUP or CUBE concepts and provide some useful information of your choice for management.

5 Discussion

x

6 Summary of what was learnt

x