ISYS90086 Data Warehousing

Assignment 1

Fantastic Fireworks

883717 Yuming Lin

935666 Xian Dong

1. Executive Summary

In the business data mining and analysis, the traditional database that pays more attention to daily transaction requests cannot meet the user's needs well. Therefore, companies need to utilize the data warehouse for decision analysis. Data warehousing is a data storage technology. It processes the raw data in the traditional database and then integrates it into a comprehensive and transparent data organization. Data analysis can be used to develop a process for a company's future scenarios and to provide a collection of policies. It can also showcase intelligent decision analysis and guide business process improvement, monitoring time, cost, quality, and control.

Compared to traditional databases, data warehousing does not have such excellent transaction processing capabilities. It contains many data, which may come from the traditional database, or some external data. Therefore, the data warehouse has more data, better readability, and higher query efficiency. However, its data redundancy is high, and its scalability is poor. In a data warehouse, there are fewer connections between tables. Additionally, some intermediate results are stored in the data warehouse to represent the data content we need efficiently.

This paper will introduce the definition of the data warehouse firstly. Then it will gradually analyze the data warehouse design of the Fantastic Fireworks case study. Next, the paper will give detailed decision analysis and usage recommendations based on business issues that may arise in different situations.

2. Data Warehousing Overview

The traditional data service cannot satisfy the information need of this era. Therefore, data warehouse is the key to solve such problems.

2.1 The reason for using Data Warehouse

When an organization is running a business, it is indubitably necessary to record the data that generated by the business; plus, analyze historical data would lead the manager to a clear understanding of the business and making better decisions. With the dynamic development of electronic devices and the internet, it is impractical to process and archive the data in the traditional way for the analytic purpose. Therefore, the data warehousing was introduced to this world to solve the problem, and it has overgrown in the last few decades.

2.2 The definition of Data Warehouse

Jarke (2003) mentioned that data warehouse caches selected strategic data from multiple sources gathered over a long period of time. A data warehouse is a database composed of organizational data to produce business information through business analysis (Maynard & Naseer, 2019). It was discussed by Mukherjee and Kar (2017) that the extract-transform-load (ETL) process is the key to data warehousing. Firstly, the data warehouse would extract current and historical data from multiple internal or external sources on many operational systems, but these raw data might be poor in quality, it may not compatible with the database, or some wrong data would result in unsatisfied analysis. Then, it is important to transform the data so that it would be able to match the required form of the data warehouse and ensure the correctness of the business analysis. Finally, those data will be load into the data warehouse and the analysis can be performed upon the data in the warehouse.

2.3 The Benefit of Data Warehouse

Nagabhushana (2006) illustrated that the data warehouse would provide a stable way for users to frequently and rapidly access the data, especially from a historical perspective. The warehouse should also allow users to acquire database on specific demand, and it could receive a different form of query from user input and search through the warehouse, to make sure the returned results are integrated and accurate. In this way, the users can view facts of the business which are customized for business analytics requirement, and they can also scrutinize the warehouse data from the various perspective in the different level of details.

The data warehouse would also offer a service to business organizations to record their operation data accurately, the data in a warehouse will be updated frequently to ensure the essential data are not only updated to the latest version but also archived the old data. This kind of feature would maintain the accuracy in all data, also improve the data quality of the warehouse.

The data warehouse is a database designed for analytic tasks, and this is also a significant benefit for organizations. The warehouse data can be processed by experienced business analysts to produce business knowledge and strategic information, which would be helpful for organizations to make decisions about their business.

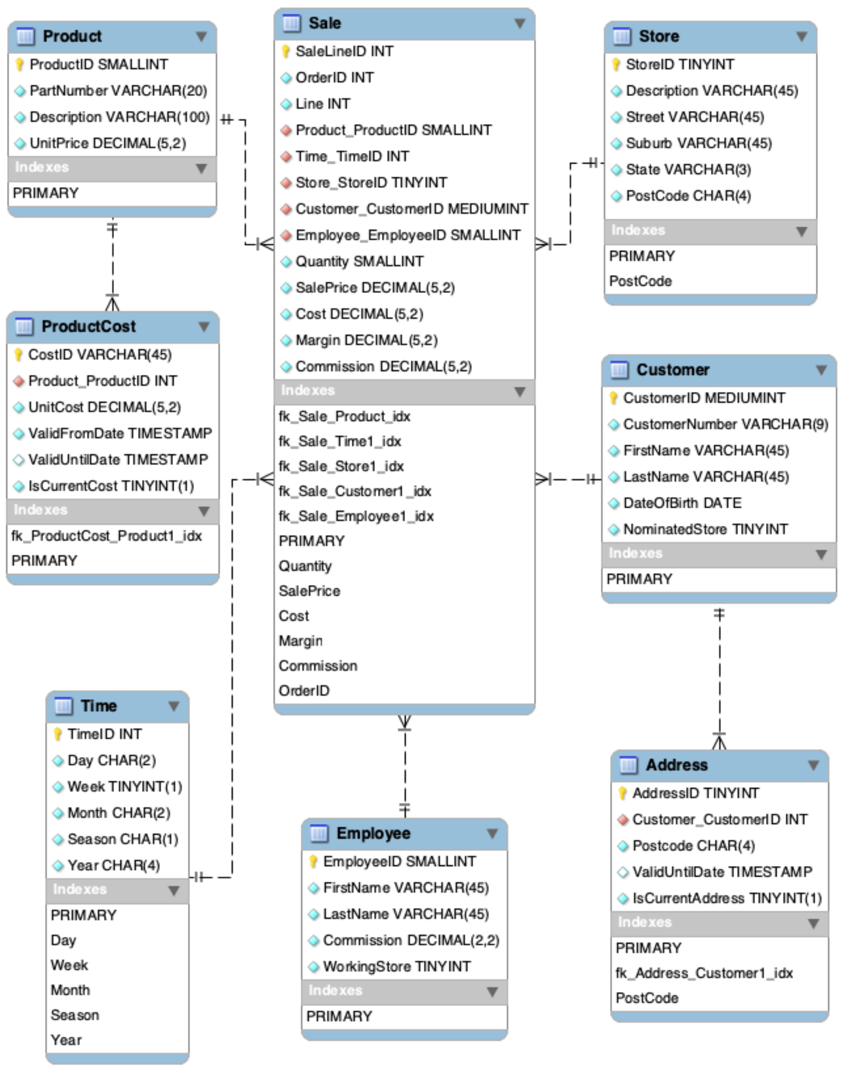

Figure 1 E-R diagram design

3. Design of the Data Warehouse

Since the data in most existing database systems is decentralized rather than integrated. Therefore, we need to extract, filter, clean, integrate the source data scattered according to the decision analysis to adapt to the needs of the data warehouse environment. The E-R diagram design is shown in Figure 1.

3.1 Identify Topics and Identify Business Process

The theme of the case determines the design content of the data warehouse. Based on the case study provided by the company, the data warehouse designer collects the existing data and documents and actively exchanges experiences with the business owner. To fully understand the architecture of the current database system and to identify existing data content and data sources. Then we need to determine the business process of the case. For example, in Fantastic Firework case, the theme of the project is the fireworks sales inventory management system. Companies need to analyse "what fireworks products have the best profit." Then, Sale could be a business process. The business process needs to be able to demonstrate a multifaceted dimension and the relationship between the statistics, that is, to show all the essential information. When all the necessary information is stored in dimensions, a simple fact table is a display of the required data information.

Therefore, at the beginning of data warehouse design, we decided to use a star schema to design a data warehouse and draw the E-R diagram. Statistical numerical data is stored as a fact table in the center of the star schema. Different dimensions are located in different corners of the star schema. In the data warehouse, we take into account the data in the fact table by using a combination of dimensions. If there are multiple fact tables from the data warehouse, it is called data marts. Data marts embody information about a particular aspect of a data warehouse, and there may be multiple data marts in a data warehouse.

3.2 Measured Value

In cube datasets, measured values are determined based on a column in the fact table, which is usually numeric data. Additionally, the measured value is the central value of the dataset being analyzed. After the theme is finalized, we need to think about which measures are required. For example, the profit of a store, the cost of the purchase, and so on. Before entering the data warehouse, we summarize the data, average the data, or take the maximum and minimum values. Measures are the attributes we want to count, which can intuitively provide statistical results to our customers to help them make business decisions.

In the Firework case, we have measures such as Margin, Commission, SalePrice, Cost and so on.

3.3 Declare the Grain

Grain refers to the level of refinement or integration of data stored in a data unit of a data warehouse (Maynard & Naseer， 2019). After determining measures, we will base on the aggregation of measures under different dimensions. The size of the grain will directly affect the level of detail that the data warehouse provides for decision making. For example, in the case of fireworks sales, we recorded the current time data to 'Day'. Then in the analysis, We can use "Day" as the minimum unit to calculate the recorded data. We can also set the granularity to 'Year', 'Month' or even 'Second'. The size of the grain needs to be determined by the cases requirements. Different fact tables can have different grain divisions. In the firework case, we divide the time grain into 'Day'.

3.4 Determine the Dimension

A dimension is a type of attribute used to reflect a business (Naseer, 2019). A collection of such attributes constitutes a dimension. For example, in the case of fireworks, we need to analyze data by time, store, customer, product, and employee. Then the time, store, customer, product, and employee are different dimensions. These dimensions allow not only cross-analysis but also a summary of the various measures.

In the design of firework case, we determine the hierarchy and the level of the attributes in the dimension firstly. In the time dimension, we form a hierarchy according to 'Day, Month, Year, Season, Week'. In the hierarchy, 'Day', 'Month', 'Year', 'Season' and 'Week' are the five levels. In the customer and store dimensions, we divided 'Street, Suburb, State, Postcode' into a hierarchy. It contains four levels of 'Street', 'Suburb', 'State' and 'Postcode'. In the design of the data warehouse table, we record all of these levels in the table of one dimension as attributes. In addition, we also use Surrogate Keys to identify rows in the sale table. It is a good help for the clear presentation of data and matching queries for fields.

3.5 Identify Facts and Create a Fact Table

The fact table contains the actual business fact. Each data warehouse contains one or more fact tables and has actual measured data. The data in the transaction fact table is generated after a transaction event occurs, and the grain of the data is usually one record per transaction. Once the transaction is committed, the fact table data is inserted, and the data is not changed. The data update method is incremental update. Furthermore, designers should actively focus on the processing of null data, which will have a significant impact on the query.

In this fireworks case, we determine the Sale theme as a fact table firstly. Next, we connect the dimension table to the fact table. After the connection, the primary key of each dimension table is placed in the fact table as a foreign key. There are no descriptive attributes in the fact table but only clear measured data values.

3.6 Slowly Changing Dimension Processing

A slowly changing dimension means that in one dimension table, one or more attributes in the dimension table produce unstable and slow changes. In this Firework case, we think that there are three dimensions are changing slowly, the residential address, the price and the cost, in another word, they are PostCode in the customer table, the SalePrice in the sale table and the product purchase UnitCost in the product table. So, we decided to expand the data warehouse into a snowflake model. There are three practical ways to handle the slowly changing dimension. The first method is to overwrite the old data once there is new data written in the table. The second method is to add a new row to the system each time to record the save. This method will retain all historical data. The third method is to add a new column to the system. It will maintain the first data record and the last updated data. (Naseer, 2019)

In this Firework case, we utilized the first and the second method. Since we do not need to know the history of sale price, we choose to overwrite the data. Moreover, by creating a new Address table and ProductCost table, the user's address changes and the historical price changes of the product are recorded. Thus, these two new tables will record the start date and the end date, then record whether the data is the current usage data.

3.7 Adding an Index

The index is a separate file that contains pointers to table rows, to allow fast retrieval. We usually use indexes in the following cases. According to Wadley (2019), in the database system, primary and foreign keys are automatically indexed. Additionally, the index is also utilized in large tables and columns that require frequent use of WHERE clause queries. Next, the clauses that often use ORDER BY and GROUP BY also require the index. Finally, if a column has >100 distinct values but not if <30 values, it also needs to use the index.

In Firework case, all primary and foreign keys already use the index. According to the requirements, the customer's age and customer's address need to be classified. Therefore, we use the index in the Birthday attribute and PostCode attribute in the Customer table and the Address table respectively. Additionally, the time dimension also needs to be filtered, so we use the index for all the attributes in the time dimension. It is also necessary to use the index on PostCode in store table. At least, in sale table, OrderID, Quantity, SalePrice, Cost, Margin and Commission are utilized the index.

4. Addressing Business Problems

The data warehouse support business problem solving by produce strategic information. There are five business problems can be solved by this data warehouse.

4.1 Who are the Key Customers?

One definition of key customers can be the most profitable customers. To solve this business problem, it requires Sale, Customer and Time tables.

The fact table of Sale contains the attribute of Margin for each unit, the Margin was calculated by SalePrice – Cost. Firstly, we make summation of Quantity (unit sales), SalePrice * Quantity (dollar sales) and Margin * Quantity (profit), group by the CustomerID. Thus, we have the data that the total quantity of product, total price and total margin for each customer across all time. If we order the results by the total margin, then we can easily find the most profitable customers. But it needs to be noticed that the customers are grouped by the CustomerID, so if a customer moved to another suburb or made purchases in multiple stores (these actions will create a new CustomerID for this person), then this individual will be considered as a different customer in the system.

The problem requires business knowledge for various time periods, then the data above should be grouped by Year, Season (Quarter) or Month separately, they are attributes in Time dimension. In this way, the data warehouse can show the organization the most profitable customers in different time periods.

If it also requires the customers divided by their suburbs, the Customer dimension can help, its Address table contains the Suburb and Postcode of each customer. If SUM the total margin of all customers and group them by Suburb or Postcode, we can get the suburb that contributed most profit to the organization’s business.

If it is needed to aggregate customers in age group, the data should be group by birthday rather than CustomerID, or it can be grouped by the CustomerID first and then by the birthday, the results should be the same. The way to do this is using the time stamp, e.g., if today is 1/15/2019, then customers who have birthday before 1/15/1969 should be classified in the age group over 50. Again, if SUM the margin in all sales, the results would show the organization which age group contributed the most margin; but, if using COUNT(DISTINCT OrderID) rather than SUM(Margin*Quantity), the results would be the number of sales for each age group. The group of key customers can be easily identified after the results are sorted in descending order.

4.2 Which Products are the Most Profitable?

This solution should engage Sale with Product and Time dimension.

Firstly, the requirement is providing information for various time periods. So, the data needs to be grouped by attributes in Time dimension, e.g., Year, Season or Month.

To return the value of unit sales, dollar sales and margin for different products, it needs to SUM those attributes multiplies quantity first, and then group by ProductID. Now, we have the total quantity, total dollar sale and the total margin for every product.
Next step is ordering the data by time periods and margin. The outcome would be a list of products in different time and sorted in descending order of the margin value, with their unit sales and dollar sales.

4.3 Which Store Location is the Most Profitable?

This query should join three tables: Sale, Store and Time.

This problem demands the data for each month of the previous year. The first step is checking the Time dimension, filter the data to find the sales which year attribute in Time dimension is equal to the last year (or current year - 1). Then SUM the Quantity, SalePrice * Quantity, and Margin * Quantity while group the data by Month and StoreID.

Ordering the output data by Month first, then by the Margin. The result would be the rank of stores in different month in last year. Because the Store dimension is also involved in this query, the details of store location like street, suburb, and state are also shown in the results.

4.4 Which Time Periods are the Most Profitable?

The time dimension is the only dimension involved in this problem.

Just like the cases above, the summation of quantity, dollar sales, and margin are necessary, and then it just needs to be grouped by different time periods. After joining Sale with Time through TimeID, the data can be grouped by Month, Season (Quarter) and Year. This is how the organization can view how much product was sold, how much money and profit were earned in a specific time period.

Also, the Time dimension has a week attribute which indicates the weekday of the sale date, if the data are grouped by this attribute, then the profit information would be shown in the perspective from Monday to Sunday. It is important to notice that only the original store opens in Sunday, so only the data of the original store should be evaluated in this case. This could help the organization to decide whether opening the stores on Sunday or not.

4.5 Who are the Key Employees?

This problem needs Sale, Employee and Time. The dollar sale and commission of every employee need to be evaluated. The Commission attribute in Sale fact table was calculated by SalePrice * CommissionRate.

To provide business knowledge for this problem, the first step is SUM SalePrice * Quantity, Margin * Quantity, and Commission * Quantity separately in fact table, and then group by EmployeeID and time periods. Thus, the employee sales, the margin they earned and commission for each employee are presented and classified by different time periods. Also, it is possible to COUNT(OrderID) to find out which employee made most sales.

If the results are sorted in descending order by employee sales, the outcome should rank the employee contribution to the store, and the top ones should be considered as the key employees.

5. Conclusion

This assignment records the design flow of a data warehouse, including tables, relationships, granularity, and selection of attributes. The whole process is implemented in the case of the facts. Furthermore, this paper also answered the questions raised in the firework case and solved the growing problems of commercial data analysis in modern society.

References

Jarke, M. (2003). *Fundamentals of Data Warehouses*. https://doi-org.ezp.lib.unimelb.edu.au/10.1007/978-3-662-05153-5

Maynard, S., & Naseer, H. (2018). *Data Warehousing D1A - Introduction to Subject & Overview and Concepts. The* University of Melbourne. https://app.lms.unimelb.edu.au

Mukherjee, R., & Kar, P. (2017). *A Comparative Review of Data Warehousing ETL Tools with New Trends and Industry Insight. 2017 IEEE 7th International Advance Computing Conference (IACC)*, 943-948. doi: 10.1109/IACC.2017.0192

Nagabhushana, S. (2006) *Data Warehousing OLAP and Data Mining* http://dx.doi.org/10.1109/IACC.2017.0192

Naseer, H. (2019). *ISYS90086 Data Warehousing D2B - Data Warehouse Design*. The University of Melbourne. https://app.lms.unimelb.edu.au

Wadley, G. (2018). *COMP90002 Database Systems & Information Modelling W06-PhysicalDesign*. The University of Melbourne. https://app.lms.unimelb.edu.au

Appendix 1 – Data Dictionary

- **Sale Fact Table**

Attribute	Description	Source
SaleLineID	Unique identifier for a line in all sales.	A unique number generated for data warehouse.
OrderID	Unique identifier for a sale.	OrderID attribute in SALES database table of the sales system
LineID	Unique identifier for one line of a sale.	Line attribute in SALES ITEM database table of the sales system
Product_ProductID	Unique identifier for a product.	Product attribute in SALES ITEM database table of the sales system
Time_TimeID	Unique identifier for a time record.	Date attribute in SALES database table of the sales system
Store_StoreID	Unique identifier for a store.	Store attribute in SALES database table of the sales system
Customer_CustomerID	Unique identifier for a customer in one store.	CustomerID attribute in SALES database table of the sales system
Employee_EmployeeID	Unique identifier for an employee.	Salesperson attribute in SALES database table of the sales system
Quantity	The amount of a particular product in a sale order.	Quantity attribute in SALES ITEM database table of the sales system
SalePrice	The amount of money paid for buying a particular unit of product from a store by a customer.	SalePrice attribute in SALES ITEM database table of the sales system
Cost	The cost of the product on the most recent order before the sale.	UnitCost attribute in ProductCost database table.
Margin	The amount of money the store earned as profit from one unit of product.	SalePrice - Cost
Commission	The amount of money an employee can acquire as a reward from one product sale.	SalePrice*CommissionRate. CommissionRate is an attribute in Employee dimension table.

- **Store Dimension Table**

Attribute	Description	Source
StoreID	Unique identifier for a store.	A unique number generated for data warehouse.
Description	The city where the store is located.	Description attribute in STORE database table of the sales system
Street	The street where the store is located.	Derived from Address attribute in STORE database table of the sales system
Suburb	The suburb where the store is located.	Derived from Address attribute in STORE database table of the sales system

State	The start where the store is located.	Derived from Address attribute in STORE database table of the sales system
Postcode	The postcode of the suburb where the store is located.	Derived from Address attribute in STORE database table of the sales system

- **Customer Dimension Table**

Attribute	Description	Source
CustomerID	Unique identifier for a customer in one store.	A unique number generated for data warehouse.
CustomerNumber	Unique number assigned to each customer.	Customer ID attribute in CUSTOMER database table of the sales system
FirstName	The first name of a customer.	The first part of Name attribute in CUSTOMER database table of the sales system.
LastName	The last name of a customer.	The last part of Name attribute in CUSTOMER database table of the sales system.
DateOfBirth	The birthday of a customer.	Date of Birth attribute in CUSTOMER database table of the sales system.
NominatedStore	The ID of store that this customer was registered.	Derived from CustomerNumber.

- **Customer Address Table**

Attribute	Description	Source
AddressID	Unique identifier for an address record of a customer.	A unique number generated for data warehouse.
Customer_CustomerID	Unique identifier for a customer in one store.	A unique number generated for data warehouse.
Postcode	The postcode of the suburb that the customer lives in.	Postcode attribute in CUSTOMER database table of the sales system
ValidUntilDate	The date that the customer moves to a new address.	Valid until date attribute in CUSTOMER database table of the sales system
IsCurrentAddress	Whether this address is where the customer lives in right now or not.	Derived from ValidUntilDate.

- **Product Dimension Table**

Attribute	Description	Source
ProductID	Unique identifier for a product.	A unique number generated for data warehouse.
PartNumber	Unique number assigned to each product.	PartNumber attribute in PRODUCT database table of the inventory system.
Description	A brief introduction of a product.	Description attribute in PRODUCT database table of the inventory system.

UnitPrice	The current RRP of a product.	Price attribute in PRODUCT database table of the inventory system.

- **Product Cost Table**

Attribute	Description	Source
CostID	Unique identifier for a product host record.	A unique number generated for data warehouse.
Product_ProductID	Unique identifier for a product.	A unique number generated for data warehouse.
UnitCost	The amount of money paid for purchasing a particular unit of product from a supplier by the organization.	Cost per Item attribute in PRODUCT database table of the inventory system.
ValidFromDate	The first day that the product start using this value of cost.	Date attribute in PRODUCT database table of the inventory system.
ValidUntilDate	The date that the cost of a product changed to a new value.	Date attribute in PRODUCT database table of the inventory system. If the cost for a product was changed, then the ValidFromDate of the new cost is also assigned to the ValidUntilDate of the old cost.
IsCurrentCost	Whether this UnitCost still valid or not.	Derived from ValidUntilDate. The UnitCost is valid if and only if the ValidUntilDate is null.

- **Time Dimension Table**

Attribute	Description	Source
TimeID	Unique identifier for a time record.	A unique number generated for data warehouse.
Day	The day that the sale happened.	Day value of Date attribute in SALES database table of the sales system.
Week	The day in a week that the sale happened. E.g. Monday, Tuesday, etc.	Derive from the Day, Month and Year attribute.
Month	The month that the sale happened.	Month value of Date attribute in SALES database table of the sales system.
Season	The season/quarter that the sale happened.	Derive from the Day and Month attribute.
Year	The year that the sale happened.	Year value of Date attribute in SALES database table of the sales system.

- **Employee Dimension Table**

Attribute	Description	Source
EmployeeID	Unique identifier for an employee.	A unique number generated for data warehouse.
EmployeeNumber	Unique number assigned to each employee.	ID attribute in SALES PERSON database table of the sales system.

FirstName	The first name of an employee.	The first part of Name attribute in SALES PERSON database table of the sales system.
LastName	The last name of an employee.	The last part of Name attribute in SALES PERSON database table of the sales system.
CommissionRate	The portion of reward an employee can acquire from a sale.	Commission Rate attribute in SALES PERSON database table of the sales system.
WorkingStore	The store the employee works in.	Derive from the first letter in EmployeeNumber.

Appendix 2 – SQL

The reasons for using index on attributes have already explained in the design part.

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema Fantastic Firework
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Table `Product`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Product` ;

CREATE TABLE IF NOT EXISTS `Product` (
  `ProductID` SMALLINT NOT NULL,
  `PartNumber` VARCHAR(20) NOT NULL,
  `Description` VARCHAR(100) NOT NULL,
  `UnitPrice` DECIMAL(5,2) NOT NULL,
  PRIMARY KEY (`ProductID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Time`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Time` ;

CREATE TABLE IF NOT EXISTS `Time` (
  `TimeID` INT NOT NULL,
  `Day` CHAR(2) NOT NULL,
  `Week` TINYINT(1) NOT NULL,
  `Month` CHAR(2) NOT NULL,
  `Season` CHAR(1) NOT NULL,
  `Year` CHAR(4) NOT NULL,
  PRIMARY KEY (`TimeID`),
  INDEX `Day` (`Day` ASC) VISIBLE,
  INDEX `Week` (`Week` ASC) VISIBLE,
  INDEX `Month` (`Month` ASC) VISIBLE,
  INDEX `Season` (`Season` ASC) VISIBLE,
  INDEX `Year` (`Year` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Store`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Store` ;
```

```
CREATE TABLE IF NOT EXISTS `Store` (
  `StoreID` TINYINT NOT NULL,
  `Description` VARCHAR(45) NOT NULL,
  `Street` VARCHAR(45) NOT NULL,
  `Suburb` VARCHAR(45) NOT NULL,
  `State` VARCHAR(3) NOT NULL,
  `PostCode` CHAR(4) NOT NULL,
  PRIMARY KEY (`StoreID`),
  INDEX `PostCode` (`PostCode` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Customer`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Customer` ;

CREATE TABLE IF NOT EXISTS `Customer` (
  `CustomerID` MEDIUMINT NOT NULL,
  `CustomerNumber` VARCHAR(9) NOT NULL,
  `FirstName` VARCHAR(45) NOT NULL,
  `LastName` VARCHAR(45) NOT NULL,
  `DateOfBirth` DATE NOT NULL,
  `NominatedStore` TINYINT NOT NULL,
  PRIMARY KEY (`CustomerID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Employee`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Employee` ;

CREATE TABLE IF NOT EXISTS `Employee` (
  `EmployeeID` SMALLINT NOT NULL,
  `FirstName` VARCHAR(45) NOT NULL,
  `LastName` VARCHAR(45) NOT NULL,
  `Commission` DECIMAL(2,2) NOT NULL,
  `WorkingStore` TINYINT NOT NULL,
  PRIMARY KEY (`EmployeeID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Sale`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Sale` ;

CREATE TABLE IF NOT EXISTS `Sale` (
  `SaleLineID` INT NOT NULL,
  `OrderID` INT NOT NULL,
  `Line` INT NOT NULL,
  `Product_ProductID` SMALLINT NOT NULL,
  `Time_TimeID` INT NOT NULL,
  `Store_StoreID` TINYINT NOT NULL,
  `Customer_CustomerID` MEDIUMINT NOT NULL,
  `Employee_EmployeeID` SMALLINT NOT NULL,
  `Quantity` SMALLINT NOT NULL,
  `SalePrice` DECIMAL(5,2) NOT NULL,
  `Cost` DECIMAL(5,2) NOT NULL,
  `Margin` DECIMAL(5,2) NOT NULL,
  `Commission` DECIMAL(5,2) NOT NULL,
  INDEX `fk_Sale_Product_idx` (`Product_ProductID` ASC) VISIBLE,
  INDEX `fk_Sale_Time1_idx` (`Time_TimeID` ASC) VISIBLE,
  INDEX `fk_Sale_Store1_idx` (`Store_StoreID` ASC) VISIBLE,
  INDEX `fk_Sale_Customer1_idx` (`Customer_CustomerID` ASC) VISIBLE,
  INDEX `fk_Sale_Employee1_idx` (`Employee_EmployeeID` ASC) VISIBLE,
  PRIMARY KEY (`SaleLineID`),
  INDEX `Quantity` (`Quantity` ASC) VISIBLE,
  INDEX `SalePrice` (`SalePrice` ASC) VISIBLE,
  INDEX `Cost` (`Cost` ASC) VISIBLE,
  INDEX `Margin` (`Margin` ASC) VISIBLE,
  INDEX `Commission` (`Commission` ASC) VISIBLE,
  INDEX `OrderID` (`OrderID` ASC) VISIBLE,
  CONSTRAINT `fk_Sale_Product`
    FOREIGN KEY (`Product_ProductID`)
    REFERENCES `Product` (`ProductID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Sale_Time1`
    FOREIGN KEY (`Time_TimeID`)
    REFERENCES `Time` (`TimeID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Sale_Store1`
    FOREIGN KEY (`Store_StoreID`)
    REFERENCES `Store` (`StoreID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Sale_Customer1`
    FOREIGN KEY (`Customer_CustomerID`)
    REFERENCES `Customer` (`CustomerNumber`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Sale_Employee1`
    FOREIGN KEY (`Employee_EmployeeID`)
    REFERENCES `Employee` (`EmployeeID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Address`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Address` ;

CREATE TABLE IF NOT EXISTS `Address` (
  `AddressID` TINYINT NOT NULL,
  `Customer_CustomerID` INT NOT NULL,
  `Postcode` CHAR(4) NOT NULL,
  `ValidUntilDate` TIMESTAMP NULL,
  `IsCurrentAddress` TINYINT(1) NOT NULL,
  PRIMARY KEY (`AddressID`),
  INDEX `fk_Address_Customer1_idx` (`Customer_CustomerID` ASC) VISIBLE,
  INDEX `PostCode` (`Postcode` ASC) VISIBLE,
  CONSTRAINT `fk_Address_Customer1`
    FOREIGN KEY (`Customer_CustomerID`)
    REFERENCES `Customer` (`CustomerNumber`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ProductCost`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ProductCost` ;

CREATE TABLE IF NOT EXISTS `ProductCost` (
  `CostID` VARCHAR(45) NOT NULL,
  `Product_ProductID` INT NOT NULL,
  `UnitCost` DECIMAL(5,2) NOT NULL,
  `ValidFromDate` TIMESTAMP NOT NULL,
  `ValidUntilDate` TIMESTAMP NULL,
  `IsCurrentCost` TINYINT(1) NOT NULL,
  INDEX `fk_ProductCost_Product1_idx` (`Product_ProductID` ASC) VISIBLE,
  PRIMARY KEY (`CostID`),
  CONSTRAINT `fk_ProductCost_Product1`
    FOREIGN KEY (`Product_ProductID`)
    REFERENCES `Product` (`ProductID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Appendix 3 – Work Breakdown

The assignment1 work breakdown is 50% of Yuming Lin and 50% of Xian Dong.

The design of the data warehouse was done by both members, we discussed every detail of the data warehouse together. Those details including:

- The fact table
- The dimension tables
- The solution to slow changing
- The attribute and data type of each table (data dictionary)
- The ER diagram
- The solution to business problems

The work breakdown of the paper work (report writing):
Yuming Lin:

1. Section 1 - Executive Summary
2. Section 3 - Design of the Data Warehouse
3. Appendix 2 – SQL

Xian Dong:

1. Section 2 - Data Warehousing Overview
2. Section 4 - Addressing Business Problems
3. Appendix 1 – Data Dictionary