**High Fashion Inc.**

**Project Milestone 2**

**IT-509 Data Management**

**Professor: Dr. Virginia Ilie**

**Date:04/24/2019**

**By:**

**Divya Subhedar**

**California Lutheran University**

****

**ERD 1 (Divya)**



**ERD 2 (Divya)**



**Assumptions:**

1. Each product is manufactured by only one company.
2. Stores may carry 0 to many products. A product might sell-out.
3. A customer can make zero or more purchases.
4. Every sale is purchased by one and only one customer.
5. Customers can make many purchases but only one specific receipt and sale can go to a customer
6. According to purchase, customer can return one or more products without receipt. (With unique voucher code).
7. Customers can return one or multiple products with receipts.
8. Only one store can receive only one receipt and give cash back on items.
9. Returns need to be connected to inventory, in case return is considered good quality and the customer’s reason was due to gift return (wrong size or dislike of article).
10. Return with receipt needs to be connected to stores due to store tax rate and sales for receipt to document which sales was returned.
11. Stores must have at least one worker and can have many workers, but they cannot have 0.
12. Returns cannot have an overlap, they either have a receipt or they do not.
13. There are no other specializations for returns, therefore it must be a total specialization

**Rational Schema (Divya, Huizi, Shanel)**



**Data Dictionary (Divya, Huizi, Shanel)**

**TABLE: MANUFACTURERS**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **COMPANY\_ID** | **VARCHAR (5)** | **Primary Key** |
| **NAME** | **CHAR (30)** |  |
| **STREET** | **VARCHAR (50)** |  |
| **CITY** | **CHAR (20)** |  |
| **STATE** | **CHAR (20)** |  |
| **COUNTRY** | **CHAR (20)** |  |
| **ZIP\_CODE** | **INT** |  |
| **PHONE\_NUMBER** | **VARCHAR (15)** |  |
| **EMAIL** | **VARCHAR (50)** |  |

**TABLE: ASSOCIATES**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **ASSOCIATE\_ID** | **VARCHAR (5)** | **Primary Key** |
| **FIRST\_NAME** | **CHAR (20)** |  |
| **LAST\_NAME** | **CHAR (20)** |  |
| **STREET** | **VARCHAR (50)** |  |
| **CITY** | **CHAR (20)** |  |
| **STATE** | **CHAR (20)** |  |
| **COUNTRY** | **CHAR (20)** |  |
| **ZIP\_CODE** | **INT** |  |
| **PHONE\_NUMBER** | **VARCHAR (15)** |  |
| **EMAIL** | **VARCHAR (40)** |  |
| **SALARY** | **FLOAT (10, 2)** |  |
| **MANAGER\_ID** | **VARCHAR (5)** | **Foreign Key** |
| **STORE\_NUMBER** | **INT** | **Foreign Key** |

**TABLE: STORES**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **STORE\_NUMBER** | **INT** | **Primary Key** |
| **STREET** | **VARCHAR (50)** |  |
| **CITY** | **CHAR (20)** |  |
| **STATE** | **CHAR (20)** |  |
| **COUNTRY** | **CHAR (20)** |  |
| **ZIP\_CODE** | **INT** |  |
| **PHONE\_NUMBER** | **VARCHAR (15)** |  |
| **EMAIL** | **VARCHAR (40)** |  |
| **TAX\_RATE** | **FLOAT (4, 2)** |  |

**TABLE: STORES\_PRODUCTS**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **STORE\_NUMBER** | **INT** | **Primary Key: Foreign Key** |
| **PRODUCT\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **QUANTITY** | **INT** |  |

**TABLE: PRODUCTS**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **PRODUCT\_ID** | **VARCHAR (5)** | **Primary Key** |
| **STYLE\_NUMBER** | **VARCHAR (5)** |  |
| **PRODUCT\_NAME** | **VARCHAR (30)** |  |
| **DESCRIPTION** | **NVARCHAR (100)** |  |
| **COLOR** | **CHAR (15)** |  |
| **SIZE** | **VARCHAR (10)** |  |
| **PRODUCT\_CATEGORY** | **CHAR (15)** |  |
| **PRICE** | **FLOAT (10, 2)** |  |
| **COMPANY\_ID** | **VARCHAR (5)** | **Foreign Key** |

**TABLE: STORES\_PRODUCTS\_SALES**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **STORE\_NUMBER** | **INT** | **Primary Key: Foreign Key** |
| **PRODUCT\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **SALE\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **QUANTITY** | **INT** |  |
| **PRICE** | **FLOAT (10, 2)** |  |

**TABLE: SALES**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **SALE\_ID** | **VARCHAR (5)** | **Primary Key** |
| **RECEIPT\_NUMBER** | **INT** |  |
| **REGISTER\_NUMBER** | **INT** |  |
| **DATE** | **DATE** |  |
| **TIME** | **TIME** |  |
| **TAX\_RATE** | **FLOAT (4, 2)** |  |
| **PAYMENT\_METHOD** | **CHAR (15)** |  |
| **SELLING\_PRICE** | **FLOAT (10, 2)** |  |
| **DISCOUNT\_CODE** | **VARCHAR (6)** |  |
| **DISCOUNT\_PERCENT** | **FLOAT (4, 2)** |  |
| **CUSTOMER\_ID** | **VARCHAR (5)** | **Foreign Key** |
| **ASSOCIATE\_ID** | **VARCHAR (5)** | **Foreign Key** |

**TABLE: CUSTOMERS**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **CUSTOMER\_ID** | **VARCHAR (5)** | **Primary Key** |
| **FIRST\_NAME** | **CHAR (20)** |  |
| **LAST\_NAME** | **CHAR (20)** |  |
| **STREET** | **VARCHAR (50)** |  |
| **CITY** | **CHAR (20)** |  |
| **STATE** | **CHAR (20)** |  |
| **COUNTRY** | **CHAR (20)** |  |
| **ZIP\_CODE** | **INT** |  |
| **PHONE\_NUMBER** | **VARCHAR (15)** |  |
| **EMAIL** | **VARCHAR (50)** |  |

**TABLE: COUPONS**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **COUPON\_ID** | **VARCHAR (5)** | **Primary Key** |
| **COUPON\_TYPE** | **CHAR (15)** |  |
| **START\_DATE** | **DATE** |  |
| **END\_DATE** | **DATE** |  |
| **DISCOUNTCODE** | **VARCHAR (6)** |  |
| **DISCOUNTPERCENT** | **FLOAT (4, 2)** |  |

**TABLE: CUSTOMERS\_COUPONS**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **CUSTOMER\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **COUPON\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |

**TABLE: CUSTOMER\_RETURNS**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **RETURN\_ID** | **VARCHAR (5)** | **Primary Key** |
| **DATE** | **DATE** |  |
| **REASON** | **TEXT** |  |
| **RETURN\_TYPE** | **CHAR (30)** |  |

**TABLE: WITH\_RECEIPT**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **W-RETURN\_ID** | **VARCHAR (5)** | **Primary Key** |

**TABLE: WITHOUT\_RECEIPT**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **WO-RETURN\_ID** | **VARCHAR (5)** | **Primary Key** |
| **VOUCHER\_NUMBER** | **VARCHAR (5)** |  |

**TABLE: SALES\_PRODUCTS\_RECEIPT**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **W-RETURN\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **PRODUCT\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **SALE\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |

**TABLE: CUSTOMERS\_PRODUCTS\_WITHOUT\_RECEIPT**

|  |  |  |
| --- | --- | --- |
| **COLUMN NAME** | **TYPE/LENGTH** | **CONSTRAINTS** |
| **WO-RETURN\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **CUSTOMER\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **PRODUCT\_ID** | **VARCHAR (5)** | **Primary Key: Foreign Key** |
| **PRODUCT\_AMOUNT** | **FLOAT (10, 2)** |  |
| **TAX\_RATE** | **FLOAT (4, 2)** |  |

**Datatypes Research**

**STRING DATA TYPES:**

|  |  |  |
| --- | --- | --- |
| ***MS. SQL SERVER DATATYPES*** | ***DESCRIPTION*** | ***WHEN TO USE*** |
| **CHAR(SIZE)** | Fixed length string (letters, numbers, and special characters) that is specified in parenthesis up to 8000 characters | When there is a fixed length value (for example USA, or state CA) that doesn’t need any complex formulas |
| **VARCHAR(SIZE)** | Variable character is a variable length string with a max size specified in parenthesis up to a max value of 8000 characters | Flexible character length with a max size. Perfect for street addresses |
| **TEXT** | String with a max length of 2^31-1 characters | Store articles, not large enough for books |
| **NTEXT** | Variable width Unicode string, max of 2^30-1of text data, and no storage description | Gives a uniform encoding standard with text that can be used to make uniform data |
| **NCHAR** | Fixed width Unicode string, with a max of 4,000 characters, and a storage defined with x 2 | Gives a uniform encoding standard with characters and can be used to make uniform data |
| **NVARCHAR** | Variable width Unicode string with a max of 4,000 characters, and no storage description | Gives uniform encoding standard with variable characters and can be used to make uniform data |
| **BINARY** | Fixed width binary string with max of 8,000 bytes and no storage description | Use when size of columns are relatively uniform |
| **VARBINARY** | Variable width binary string, with max 8,000 bytes, and no storage description | When variables are expected to exceed 8000 bytes |
| **IMAGE** | Variable width binary string 2^31-1 bytes, and no storage description | Store images, office documents, compressed data |

**NUMBER DATA TYPES:**

|  |  |  |
| --- | --- | --- |
| ***MS. SQL SERVER DATATYPES*** | ***DESCRIPTION*** | ***WHEN TO USE*** |
| **BIT** | Bits are integers that can be 0,1, or NULL, no storage description | Used for true or false situations (0 1) |
| **TINYINT** | Whole numbers from 0 to 255, can store up to 1 byte (8bits) | When numbers are not expected to exceed 1byte |
| **SMALLINT** | Whole numbers between ~ -32.7 thousand to 32.7 thousand, storage up to 2 bytes | When numbers are not expected to exceed 2 bytes |
| **INT** | Whole numbers between ~ -2billion to 2billion, storage 4 bytes | When numbers are not expected to exceed 4 bytes |
| **BIGINT** | Whole numbers between ~ -9quintillion to 9quintillion, storage 8 bytes | When data is expected exceed 2billion to 2billion, or 4 bytes |
| **DECIMAL(P,S)** | Fixed precision and scale numbers from -10^38 + 1 to 10^38 – 1. Precision parameter indicates max digits stored both left and right of the decimal point valued from 1 to 38 (default 18). Scale parameter indicates the max number of digits stored right of the decimal point, where s is valued from 0 to p (default 0). Storage is 5 – 17 bytes | Allows users to make precise decimal points in their numbers. This can be used for percentages, large data, etc. |
| **NUMERIC(P,S)** | Fixed precision and scale numbers from -10^38 + 1 to 10^38 – 1. P and S rules are the same as decimal. Storage is same as decimal. | Very similar to decimal. |
| **SMALLMONEY** | Monetary data from ~ -214,748.3648 to 214,748.3647, storage 4 bytes | When dealing with monetary data that will be less than $214,000 |
| **MONEY** | Monetary data from ~9.2trillion (.5808) to 9 trillion (.5807), storage 8 bytes | When dealing with monetary data that will be less than $9 trillion |
| **FLOAT(N)** | Floating precision number data from -1.79E + 308 to 1.79E + 308. N parameter indicates whether the field will hold 4 or 8 bytes. Float (24) holds a 4-byte field and float (53) holds an 8-byte field (default 53 or 8 bytes), storage 4 or 8 bytes | Very similar to decimal but takes up less storage space |
| **REAL** | Floating precision number data from -3.40E + 38 to 3.40E + 38, storage 4 bytes. Defined as float (24). | Very similar to float and decimal but takes up 4 bytes of storage. |

**DATE DATA TYPES:**

|  |  |  |
| --- | --- | --- |
| ***MS. SQL SERVER DATATYPES*** | ***DESCRIPTION*** | ***WHEN TO USE*** |
| **DATETIME** | Holds datetime ranging from 1.1.1753 to 12.31.9999 with an accuracy of 3.33 milliseconds, storage 8 bytes | Uses a lot of data for near pinpoint accurate time. Will want to use when precise time matters |
| **SMALLDATETIME** | Range 1.1.1900 to 6.6.2079, accuracy 1 minute, storage 4 bytes | Good for generic date data. When it is not important for the date to be accurate within milliseconds |
| **TIMESTAMP** | Unique number stored when a row is created or modified. Each table may only have one timestamp variable | Specifically, for when time has to be tracked and cannot be changed. |

**OTHER DATA TYPES:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***MS. SQL SERVER DATATYPES*** | ***DESCRIPTION*** | ***WHEN TO USE*** | |
| **SQL\_VARIANT** | Stores up to 8,016 bytes of various types of data (except text, ntext and timestamp) | | When the value needs to be extremely flexible. Will take up a lot of storage to do so. |
| **UNIQUEIDENTIFIER** | Stores a globally unique identifier (GUID) | | Globally unique, meaning no other ID will be like it. |
| **CURSOR** | Stores a reference to a cursor used for database operations | | Temporary reference to manipulate data |
| **TABLE** | Stores a result-set (row and column data) for later processing | | For reviewing data in an easier format |

**Normalization:**

Normalization is the process of minimizing data redundancy and data dependencies in a database. This process allows analysts to build databases utilizing as little disk space as possible, increasing the performance of the database. The process is divided into three steps to reach optimal normalization, known as normal form: 1NF, 2NF, and 3NF.

The first normal form (1NF) requires that data can be inserted and deleted with ease and that there are no multivalued or composite values in the tables. This has been demonstrated in our team’s database. Insertion and deletion of data is done with ease and does not affect other tables’ data.

The second normal form (2NF) requires that the relation has a simple primary key (PK), that the relation has no non-key attributes, and that the relation has a composite PK with all non-key attributes fully dependent on it. Our team’s database represents these factors for 2NF and is represented in our relational schema and our data dictionary. Each PK has a unique but simple key with a limited identification (ID) length, and all attributes are key attributes that depend on the PK.

The third normal form (3NF) mandates that there are not transitive dependencies in the database. We have resolved any transitive dependencies by adding foreign keys (FK) to our tables.

Normalization helps with making a clean database, however, there are times that normalization will make a database too large and time consuming. To save time and money, organizations will choose to Denormalize their database. Denormalization allows analysts to logically reduce the tables by combining different entities together. Combining tables can save storage space in a database, however, it can be risky with a higher chance of error in the database and may cause issues in deletion and insertion. Strategically choosing the minimum storage datatypes can keep the storage space small and may allow for organizations to avoid Denormalization.