***Rochester Institutes of Technology: Team 3***

**~ Final Data Mart Development Report ~**

Team # 3

Jon Aurit, Brianna Buttaccio, Lei Mon, & Jordan Rizzieri

December 10, 2017

ISTE-DW Data Warehousing

~ Table of Contents ~

[**I. Data Mart Design Definition**](#_30j0zll)2

[1. Universe of Discourse](#_3znysh7) 2

[2. Information Package](#_tyjcwt) 2

[**II. Dimensional Model**](#_3dy6vkm)3

[**III. Data Staging: ETL – Data Extract File Definitions**](#_3rdcrjn)5

[**IV. Data Staging: ETL – Source-to-Target Mappings**](#_35nkun2)6

[**V. SQL Code – Tables & Constraints**](#_2jxsxqh)6

[**VI. Data Staging Activities - ETL**](#_3j2qqm3)12

[1. Data Cleansing](#_4i7ojhp) 12

[2. Data Transformation](#_1ci93xb) 15

[3. Table Population](#_2bn6wsx) 16

[**VII. End User Applications**](#_qsh70q)16

[1. Queries](#_1pxezwc) 17

[2. A View](#_2p2csry) 18

[**VIII. Handling Slowly Changing Dimensions (SCD)**](#_147n2zr)19

[**IX. Many-to-Many (N-M) Relationship Implementation Option**](#_3o7alnk)22

[**X. Appendix (Fix Lab #3 Problems)**](#_23ckvvd)22

# I. Data Mart Design Definition

## 1. Universe of Discourse

|  |
| --- |
| Tracking manufacturer, supplier, customer and sale transaction of products from multiple organizations. |

## 2. Information Package

Process Name: Manufacturing and Sales Analysis

Grain: The sale of a single item.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Product** | **Manufacturing Cost** | **Sales Details** | **Order Dates** | **Sales Dates** | **Supplier** | **Customer** |
| productSK | manufacturingCostSk | salesDetalisSK | orderDateSK | salesDateSK | supplierSK | customerSK |
| prodID | prodID | shippingMethod | Year | Year | Name | custTypeID |
| Name | Year | orderMethod | Month | Month | Addr1 | customerID |
| Price1 | Month | paymentMethod | Quarter | Quarter | Addr2 | Name |
| Price2 | manufacturingCost |  | Day | Day | City | Addr1 |
| prodTypeID |  |  |  |  | State | Addr2 |
| typeDescription |  |  |  |  | zip | City |
| BUID |  |  |  |  |  | State |
| BUIDName |  |  |  |  |  | zip |
| BUIDAbbrev |  |  |  |  |  | typeName |
| saleBy |  |  |  |  |  |  |

Facts: amount, quantity, discounted, invoiceProcessingDays, invoiceID, invoiceDepartment, internalSale, unitCost, shippingCost

## 3. Entity Definitions

|  |  |
| --- | --- |
| **Entity** | **Entity Definition** (*genus differentia*) |
| Product | is a database table that indicates the unit price, unit cost of a particular products and suppliers’ information for each product |
| Manufacturing Cost | is a database table that describes cost of manufacturing of the products which has the cost of manufacturing a particular product in a particular year and month |
| Sales Details | is a database table that shows shipping method, order method and payment method for each sales transaction |
| Sales Dates | is a database table that indicates the sales dates of a product was sold to each customer by year, quarter, month and day |
| Order Dates | is a database table that indicates the order dates of the product from each customer by year, quarter, month and day |
| Supplier | is a database table that describes the suppliers’ information such as names and addresses. |
| Customer | is a database table that describes the customers’ information such as names and addresses. |

# II. Dimensional Model

# 

# 

# 

# 

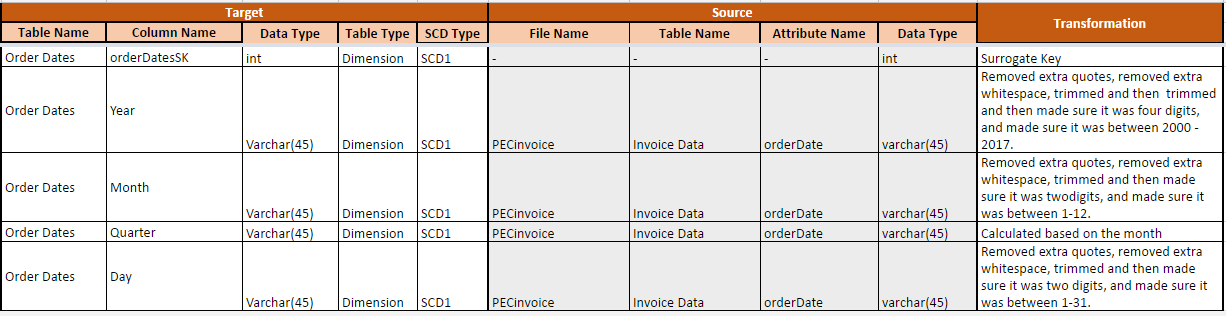
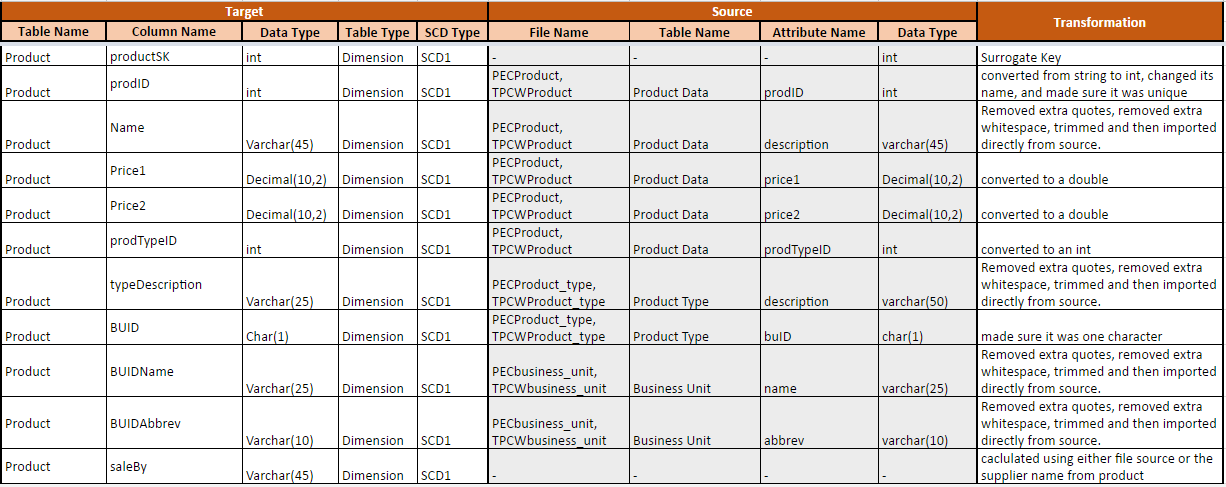
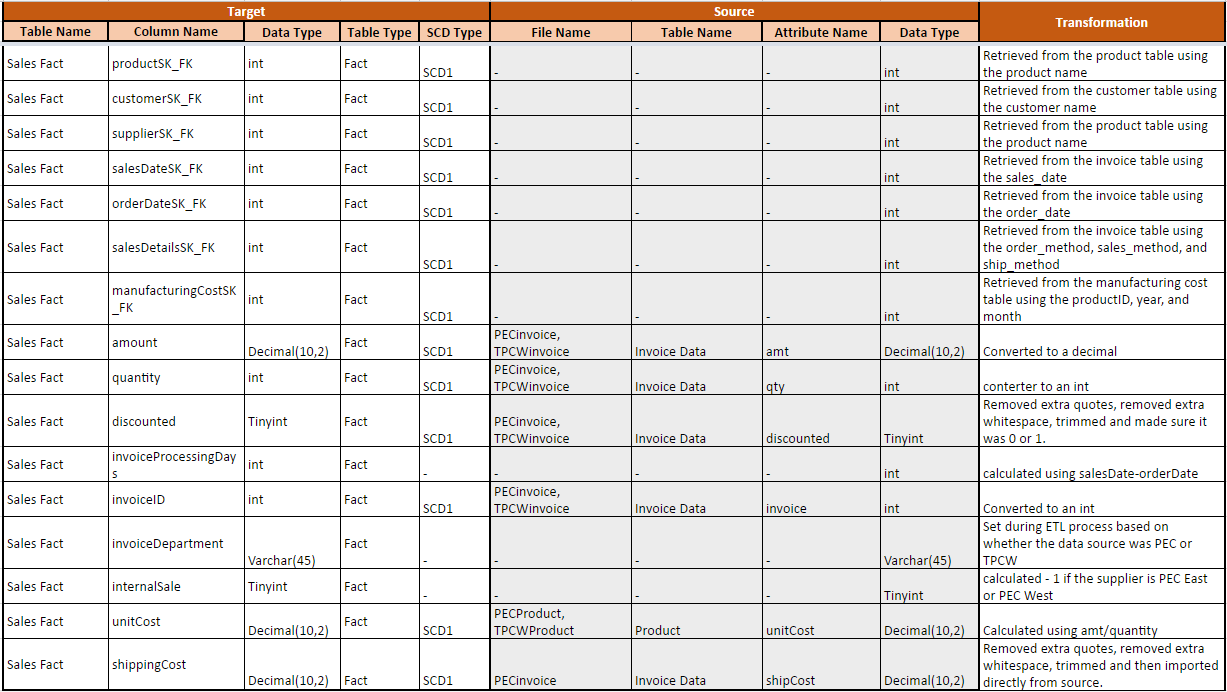
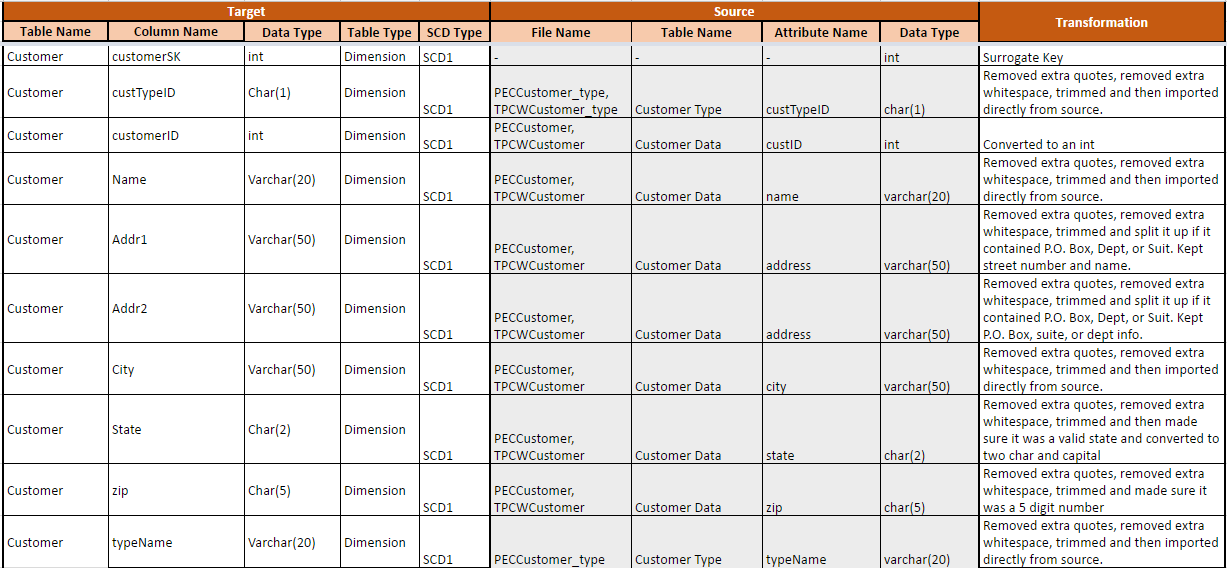
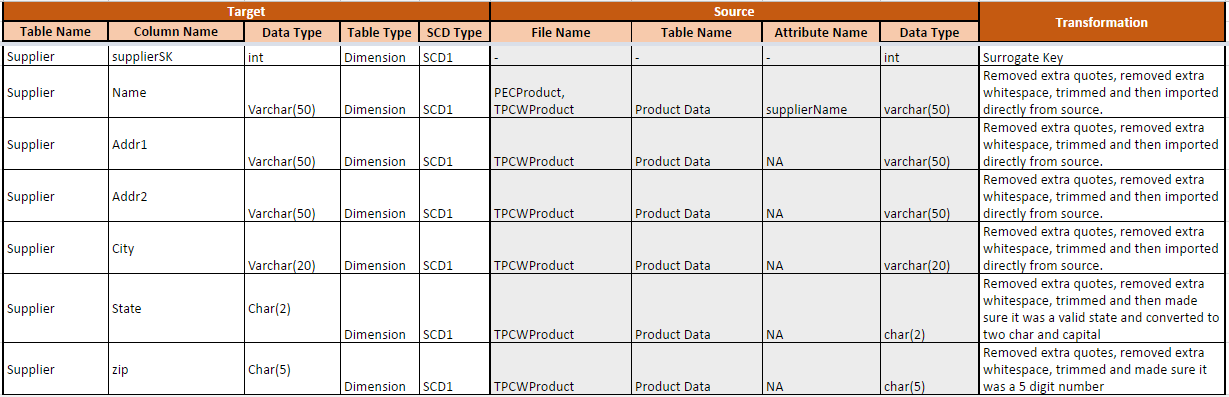
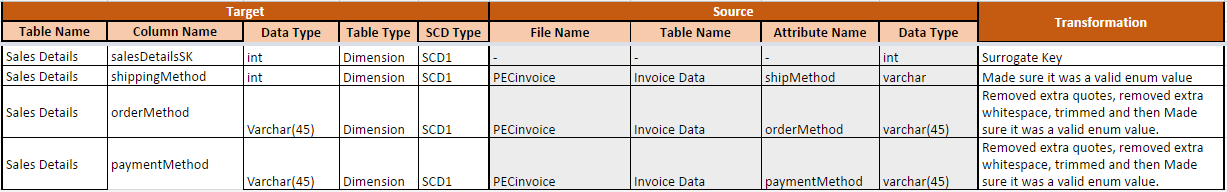
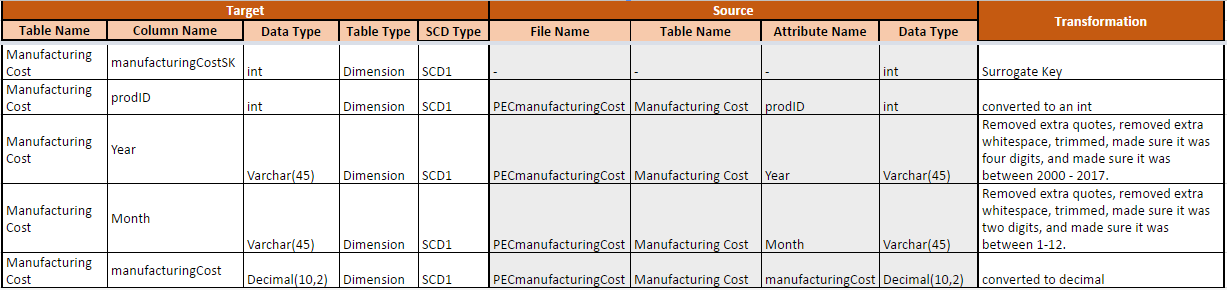
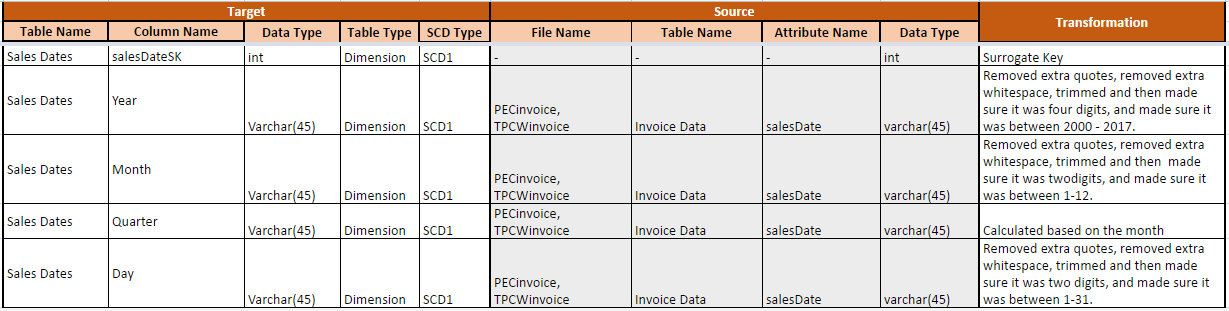
# III. Data Staging: ETL – Data Extract File Definitions

# 

|  |  |
| --- | --- |
| **Extract File Name** | **Description and format** |
| PEC Business Unit | Delimiter: ;  Enclosure: “  Header Row Present: true |
| PEC Product Type | Delimiter: ;  Enclosure: “  Header Row Present: true |
| PEC Invoice | Delimiter: ,  Enclosure: none  Header Row Present: true |
| TPCW invoice | Delimiter: ,  Enclosure: none  Header Row Present: true |
| PEC Customer | Delimiter: ;  Enclosure: none  Header Row Present: true |
| TPCW Customer | Delimiter: ;  Enclosure: “  Header Row Present: true |
| TPCW Customer Type | Delimiter: ;  Enclosure: “  Header Row Present: true |
| PEC Customer Type | Delimiter: ;  Enclosure: “  Header Row Present: true |
| TPCW Product | Delimiter: ;  Enclosure: “  Header Row Present: false |
| TPCW Product Type | Delimiter: ;  Enclosure: “  Header Row Present: true |
| TPCW Business Unit | Delimiter: ;  Enclosure: “  Header Row Present: true |
| PEC Product | Delimiter: ;  Enclosure: “  Header Row Present: true |
| PEC Manufacturing Costs | Delimiter: |  Enclosure: none  Header Row Present: true |

# 

# IV. Data Staging: ETL – Source-to-Target Mappings

Tables were split into multiple tables to make it easier to read on multiple pages.   V. SQL Code – Tables & Constraints

# 

-- 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='TRADITIONAL,ALLOW\_INVALID\_DATES';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema dw171\_salesorders

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema dw171\_salesorders

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `dw171\_salesorders` DEFAULT CHARACTER SET utf8 ;

USE `dw171\_salesorders` ;

-- -----------------------------------------------------

-- Table `dw171\_salesorders`.`customer`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `dw171\_salesorders`.`customer` (

`customerSK` INT(11) NOT NULL DEFAULT '0',

`custTypeID` VARCHAR(10) NULL DEFAULT NULL,

`customerID` TINYINT(4) NULL DEFAULT NULL,

`typeName` VARCHAR(20) NULL DEFAULT NULL,

`name` VARCHAR(50) NULL DEFAULT NULL,

`addr1` VARCHAR(50) NULL DEFAULT NULL,

`addr2` VARCHAR(50) NULL DEFAULT NULL,

`city` VARCHAR(20) NULL DEFAULT NULL,

`state` CHAR(2) NULL DEFAULT NULL,

`zip` CHAR(5) NULL DEFAULT NULL,

PRIMARY KEY (`customerSK`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `dw171\_salesorders`.`manufacturing\_cost`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `dw171\_salesorders`.`manufacturing\_cost` (

`manufacturingCostSK` INT(11) NOT NULL DEFAULT '0',

`prodID` INT(11) NULL DEFAULT NULL,

`year` VARCHAR(45) NULL DEFAULT NULL,

`month` VARCHAR(45) NULL DEFAULT NULL,

`manufacturingCost` DECIMAL(10,2) NULL DEFAULT NULL,

PRIMARY KEY (`manufacturingCostSK`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `dw171\_salesorders`.`product`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `dw171\_salesorders`.`product` (

`productSK` INT(11) NOT NULL DEFAULT '0',

`prodID` INT(11) NULL DEFAULT NULL,

`name` VARCHAR(45) NULL DEFAULT NULL,

`price1` DECIMAL(10,2) NULL DEFAULT NULL,

`price2` DECIMAL(10,2) NULL DEFAULT NULL,

`prodTypeID` INT(11) NULL DEFAULT NULL,

`typeDescription` VARCHAR(25) NULL DEFAULT NULL,

`BUID` CHAR(1) NULL DEFAULT NULL,

`BUIDname` VARCHAR(25) NULL DEFAULT NULL,

`BUIDabbrev` VARCHAR(10) NULL DEFAULT NULL,

`saleBy` VARCHAR(45) NULL DEFAULT NULL,

PRIMARY KEY (`productSK`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `dw171\_salesorders`.`sale\_details`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `dw171\_salesorders`.`sale\_details` (

`sale\_details\_sk` INT(11) NOT NULL DEFAULT '0',

`shipping\_method` VARCHAR(45) NULL DEFAULT NULL,

`order\_method` VARCHAR(45) NULL DEFAULT NULL,

`payment\_method` VARCHAR(45) NULL DEFAULT NULL,

PRIMARY KEY (`sale\_details\_sk`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `dw171\_salesorders`.`supplier`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `dw171\_salesorders`.`supplier` (

`supplierSK` INT(11) NOT NULL DEFAULT '0',

`name` VARCHAR(50) NULL DEFAULT NULL,

`addr1` VARCHAR(50) NULL DEFAULT NULL,

`addr2` VARCHAR(50) NULL DEFAULT NULL,

`city` VARCHAR(20) NULL DEFAULT NULL,

`state` CHAR(2) NULL DEFAULT NULL,

`zip` CHAR(5) NULL DEFAULT NULL,

PRIMARY KEY (`supplierSK`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `dw171\_salesorders`.`sales\_fact`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `dw171\_salesorders`.`sales\_fact` (

`productSK\_FK` INT(11) NOT NULL DEFAULT '1',

`customerSK\_FK` INT(11) NOT NULL DEFAULT '1',

`supplierSK\_FK` INT(11) NOT NULL DEFAULT '1',

`salesDateSK\_FK` INT(11) NOT NULL DEFAULT '1',

`orderDateSK\_FK` INT(11) NOT NULL DEFAULT '1',

`saleDetailsSK\_FK` INT(11) NOT NULL DEFAULT '1',

`manufacturingCostSK\_FK` INT(11) NOT NULL,

`amount` DECIMAL(10,2) NULL DEFAULT NULL,

`quantity` INT(11) NULL DEFAULT NULL,

`discounted` TINYINT(4) NULL DEFAULT NULL,

`invoiceProcessingDays` INT(11) NULL DEFAULT NULL,

`invoiceID` INT(11) NULL DEFAULT NULL,

`invoiceDepartment` VARCHAR(45) NULL DEFAULT NULL,

`internalSale` TINYINT(4) NULL DEFAULT NULL,

`unitCost` DECIMAL(10,2) NULL DEFAULT NULL,

`shippingCost` DECIMAL(10,2) NULL DEFAULT NULL,

PRIMARY KEY (`productSK\_FK`, `customerSK\_FK`, `supplierSK\_FK`, `salesDateSK\_FK`, `orderDateSK\_FK`, `saleDetailsSK\_FK`, `manufacturingCostSK\_FK`),

INDEX `customerSK\_idx` (`customerSK\_FK` ASC),

INDEX `supplierSK\_idx` (`supplierSK\_FK` ASC),

INDEX `saleDetails\_idx` (`saleDetailsSK\_FK` ASC),

INDEX `fk\_SALES\_FACT\_MANUFACTURING\_DATE1\_idx` (`manufacturingCostSK\_FK` ASC),

INDEX `salesDate\_idx` (`salesDateSK\_FK` ASC),

INDEX `orderDate\_idx` (`orderDateSK\_FK` ASC),

CONSTRAINT `customerSK`

FOREIGN KEY (`customerSK\_FK`)

REFERENCES `dw171\_salesorders`.`customer` (`customerSK`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_SALES\_FACT\_MANUFACTURING\_DATE1`

FOREIGN KEY (`manufacturingCostSK\_FK`)

REFERENCES `dw171\_salesorders`.`manufacturing\_cost` (`manufacturingCostSK`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `orderDate`

FOREIGN KEY (`orderDateSK\_FK`)

REFERENCES `dw171\_salesorders`.`dates` (`datesSK`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `productSK`

FOREIGN KEY (`productSK\_FK`)

REFERENCES `dw171\_salesorders`.`product` (`productSK`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `saleDetails`

FOREIGN KEY (`saleDetailsSK\_FK`)

REFERENCES `dw171\_salesorders`.`sale\_details` (`sale\_details\_sk`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `salesDate`

FOREIGN KEY (`salesDateSK\_FK`)

REFERENCES `dw171\_salesorders`.`dates` (`datesSK`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `supplierSK`

FOREIGN KEY (`supplierSK\_FK`)

REFERENCES `dw171\_salesorders`.`supplier` (`supplierSK`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

-- -----------------------------------------------------

-- Table `dw171\_salesorders`.`dates`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `dw171\_salesorders`.`dates` (

`datesSK` INT(11) NOT NULL AUTO\_INCREMENT,

`day` VARCHAR(45) NULL DEFAULT NULL,

`year` VARCHAR(45) NULL DEFAULT NULL,

`quarter` VARCHAR(45) NULL DEFAULT NULL,

`month` VARCHAR(45) NULL DEFAULT NULL,

PRIMARY KEY (`datesSK`))

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

INSERT INTO `customer` VALUES (1,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

INSERT INTO `manufacturing\_cost` VALUES (1,NULL,NULL,NULL,NULL);

INSERT INTO `dates` VALUES (1,null,null,null,null);

INSERT INTO `product` VALUES (1,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);

INSERT INTO `sale\_details` VALUES (1,NULL,NULL,NULL);

INSERT INTO `supplier` VALUES (1,NULL,NULL,NULL,NULL,NULL,NULL);

# 

# 

# VI. Data Staging Activities - ETL

## 1. Data Cleansing

|  |  |  |  |
| --- | --- | --- | --- |
| **DM Table** | **Attribute** | **Problem** | **Resolution Strategy** (attach code) |
| PEC Business unit | ABBREV | missing values | Replace null values with missing values  **REF IMAGE: PECBusinessUnit1.PNG** |
| PECBusinessUnit | BUID | Extra Whitespace | strip the whitespace from the BUID current value  **REF IMAGE:**  **PECBusinessUnit2.PNG** |
| PEC Product | all attributes | extra quotes | remove all the extra quotes and duplicate records  **REF IMAGE: PECProduct1.PNG** |
| PEC Product Type | all attributes | Enclosing quotes around values | delete enclosing quotes around values  **REF IMAGE:**  **PECProductType1.PNG** |
| PEC Product Type | BUID | Extra Whitespace | strip the whitespace from the BUID current value  **REF IMAGE:**  **PECProductType2.PNG** |
| PEC Invoice | custID | invalid values | check if the custID is a valid value in the specified range  **REF IMAGE:**  **PECInvoice1.PNG** |
| PEC Invoice | prodID | invalid values | check if the value in prodID is valid in the specified range  **REF IMAGE: PECInvoice2.PNG** |
| PEC Invoice | shippingType | missing values | if missing value fill in null  **REF IMAGE: PECInvoice3.PNG** |
| PEC Invoice | paymentType | missing values | if missing value fill in null  **REF IMAGE: PECInvoice3.PNG** |
| PEC Invoice | orderType | missing values | if missing value fill in null  **REF IMAGE: PECInvoice3.PNG** |
| TPCW Invoice | custID | invalid values | check if the custID is a valid value in the specified range  **REF IMAGE: TPCWInvoice1.PNG** |
| TPCW Invoice | prodID | invalid values | check if the value in prodID is valid in the specified range  **REF IMAGE:**  **TPCWInvoice2.PNG** |
| PEC Customer | custID | if null | Checks if the customerType has a value if not then sets to null  **REF IMAGE: PECCustomer1PNG** |
| PEC Customer | custID | duplicate record | remove the duplicate custIDs  **REF IMAGE: PECCustomer2.PNG** |
| PEC Customer | custtype | custtype inconsistency | check the customer type is enum value  **REF IMAGE: PECCustomer3.PNG** |
| TPCW Business\_unit | ABBREV | missing value | set “Misc” under ABBREV attribute  **REF IMAGE: TPCWBusinessUnit1.PNG** |
| TPCW Customer | custID | custID | sort the custID  **REF IMAGE: TPCWCustomer1.PNG** |
| TPCW Customer | custID | duplicate record | remove the duplicate custIDs  **REF IMAGE: TPCWCustomer2.PNG** |
| TPCW Customer | custtype | custtype inconsistency | set all the custtype to be consistent  **REF IMAGE: TPCWCustomer3.PNG** |
| TPCW Customer | state | invalid state codes | check state code is valid type  **REF IMAGE: TPCWCustomer4.PNG** |
| TPCW Customer | zip | missing digits | check zip codes are valid type  **REF IMAGE: TPCWCustomer5.PNG** |
| TPCW Product | prodID | prodID | sort the prodID  **REF IMAGE: TPCWProduct1.PNG** |
| TPCW Product | prodID | duplicate product IDs | remove all duplicate product IDs  **REF IMAGE: TPCWProduct2.PNG** |
| TPCW Product | all attributes | Enclosing quotes around values | delete enclosing quotes around values  **REF IMAGE:**  **TPCWProduct3.PNG** |
| TPCW Product | city and zip | city and zip | split city and zip fields  **REF IMAGE:**  **TPCWProduct4.PNG** |
| TPCW Product\_Type | all attributes | Enclosing quotes around values | delete enclosing quotes around values  **REF IMAGE:**  **TPCWProduct\_Type1.PNG** |
| TPCW Invoice | custID | invalid custID | delete rows with invalid custIDs  **TPCWInvoice1.PNG** |
| TPCW Invoice | prodID | invalid prodID | delete rows with invalid prodIDs  **TPCWInvoice2.PNG** |

## 2. Data Transformation

|  |  |
| --- | --- |
| **DM Table** | **Image Creation Process** (attach code) |
| Product | Changed the data types, created a department column.  **Reference Images:** Product\_Transform1.PNG , Product\_Transform2.PNG |
| ProductType | Changed the data types.  **Reference Images:** Product\_Transform3.PNG |
| Invoice | Changes the data types, changed column names, changed date formats, create source dept  **Reference Images:** Product\_Transform4.PNG , Product\_Transform5.PNG , Product\_Transform6.PNG , Product\_Transform7.PNG, transform7.png. transform8.png |
| Manufacturing Costs | Change data types to int, set date to four years, validate the dates values are within range  **Reference Images:** transform1.png, transform2.png, transform3.png |
| CustomerType | Set customers types to consistent values.  **Reference Images:** transform4.png |
| Customer | Change data types to int, set customers types to consistent values.  **Reference Images:** transform5.png. transform6.png |

## 3. Table Population

|  |  |
| --- | --- |
| **DM Table** | **Table Population Process** (attach code) |
| Product | Make sure there are unique rows. Combine the PEC and TPCW IDs. Create surrogate key.  **Reference Images:** load\_1.png  load\_3.png  load\_4.png |
| Sales\_Details | Make sure there are unique rows. Create surrogate key.  **Reference Images:** load\_1.png  load\_3.png |
| Supplier | Make sure there are unique rows. Create surrogate key.  **Reference Images:** load\_1.png  load\_3.png |
| Customer | Make sure there are unique rows. Combine the PEC and TPCW IDs. Create surrogate key.  **Reference Images:** load\_1.png  **Reference Images:** load\_2.png  load\_3.png |
| Date | Load in the salesDate and order Date as two seperate inputs and store the dates into a new Date field. Merge the tables on the date field and get rid of duplicates. Pull day, month, year, and quarter from the date field and load them. Create surrogate key.  **Reference Images:** load\_1.png, load\_3.png, load\_5.png, load\_6.png, load\_7.png, load\_8.png, load\_9.png |
| Manufacturing Cost | Make sure there are unique rows. Create surrogate key.  **Reference Images:** load\_1.png  load\_3.png |
| Product, Sales\_Details, Supplier, Customer, Date, Manufacturing Cost | Insert null record in sql file. |
| Sales Fact | Retrieved from the product table using the product name Retrieved from the customer table using the customer name Retrieved from the product table using the product name Retrieved from the invoice table using the sales\_date Retrieved from the invoice table using the order\_date Retrieved from the invoice table using the order\_method, sales\_method, and ship\_method Retrieved from the manufacturing cost table using the productID, year, and month Converted to a decimal conterter to an int Removed extra quotes, removed extra whitespace, trimmed and made sure it was 0 or 1. calculated using salesDate-orderDate Converted to an int Set during ETL process based on whether the data source was PEC or TPCW calculated - 1 if the supplier is PEC East or PEC West Calculated using amt/quantity Removed extra quotes, removed extra whitespace, trimmed and then imported directly from source.  **Reference Image:** loding fact table.PNG |

# VII. End User Applications

## 

## 1. Queries

|  |
| --- |
| **User Question/Reporting Need** |
| The percentage of invoices that are COD. |
| **SQL Code** |
| SELECT sale\_details\_sk,  (SELECT COUNT(payment\_method)  FROM sale\_details  WHERE payment\_method='cod')\*100/(COUNT(payment\_method)) AS COD\_percentage  FROM sale\_details; |
| **Supporting Index(es)** |
| CREATE INDEX payment\_method\_idx ON sale\_details(payment\_method(2)); |

|  |
| --- |
| **User Question/Reporting Need** |
| The sales by supplier state to customer state. This would be useful to see if suppliers should ship directly to customers. |
| **SQL Code** |
| SELECT sales\_fact.amount AS Sales,  customer.state AS 'Customer State',  supplier.state AS 'Supplier State'  FROM customer  JOIN sales\_fact ON customer.customersk=sales\_fact.customersk\_fk  JOIN supplier ON supplier.suppliersk=sales\_fact.suppliersk\_fk  GROUP BY 2,3; |
| **Supporting Index(es)** |
| CREATE INDEX state\_idx ON customer(state(2)); |

|  |
| --- |
| **User Question/Reporting Need** |
| Show the total cost of products for each supplier. |
| **SQL Code** |
| SELECT supplier.name AS 'Supplier Name',  product.name AS 'Product Name',  SUM(price1) AS 'Total cost of products'  FROM supplier  JOIN sales\_fact ON supplier.supplierSK = sales\_fact.supplierSK\_FK  JOIN product ON product.productSK = sales\_fact.supplierSK\_FK  WHERE supplier.name IS NOT null  GROUP BY 1; |
| **Supporting Index(es)** |
| CREATE INDEX cost\_idx ON supplier(name(4)); |

mysql> #Question: The percentage of invoice that are COD.

mysql> CREATE INDEX payment\_method\_idx ON sale\_details(payment\_method(2));

Query OK, 0 rows affected (0.04 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> SELECT sale\_details\_sk,

-> (SELECT COUNT(payment\_method)

-> FROM sale\_details

-> WHERE payment\_method='cod')\*100/(COUNT(payment\_method)) AS COD\_percentage

-> FROM sale\_details;

+-----------------+----------------+

| sale\_details\_sk | COD\_percentage |

+-----------------+----------------+

| 1 | 38.0952 |

+-----------------+----------------+

1 row in set (0.00 sec)

mysql> #Question: The sales by supplier state to customer state. This would be useful to see if suppliers should ship directly to customers

mysql> CREATE INDEX state\_idx ON customer(state(2));

Query OK, 0 rows affected (0.03 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> SELECT sales\_fact.amount AS Sales,

-> customer.state AS 'Customer State',

-> supplier.state AS 'Supplier State'

-> FROM customer

-> JOIN sales\_fact ON customer.customersk=sales\_fact.customersk\_fk

-> JOIN supplier ON supplier.suppliersk=sales\_fact.suppliersk\_fk

-> GROUP BY 2,3;

+--------+----------------+----------------+

| Sales | Customer State | Supplier State |

+--------+----------------+----------------+

| 320.00 | NULL | NULL |

| 344.00 | AK | FL |

| 203.00 | AK | KS |

| 361.00 | AK | KY |

| 113.00 | AK | MI |

| 191.00 | AK | MN |

| 301.00 | AK | PA |

| 161.00 | AK | VA |

| 133.00 | AK | WA |

| 317.00 | AZ | FL |

| 167.00 | AZ | KS |

| 257.00 | AZ | KY |

| 162.00 | AZ | MI |

| 106.00 | AZ | MN |

| 219.00 | AZ | PA |

| 379.00 | AZ | VA |

| 376.00 | AZ | WA |

| 346.00 | CA | FL |

| 113.00 | CA | KS |

| 110.00 | CA | KY |

| 381.00 | CA | MI |

| 322.00 | CA | MN |

| 148.00 | CA | PA |

| 206.00 | CA | VA |

| 290.00 | CA | WA |

| 190.00 | DC | FL |

| 350.00 | DC | KS |

| 167.00 | DC | KY |

| 351.00 | DC | MI |

| 250.00 | DC | MN |

| 161.00 | DC | PA |

| 134.00 | DC | VA |

| 125.00 | DC | WA |

| 129.00 | DE | FL |

| 240.00 | DE | KS |

| 102.00 | DE | KY |

| 351.00 | DE | MI |

| 399.00 | DE | MN |

| 307.00 | DE | PA |

| 191.00 | DE | VA |

| 205.00 | DE | WA |

| 228.00 | FL | FL |

| 327.00 | FL | KS |

| 397.00 | FL | KY |

| 185.00 | FL | MI |

| 335.00 | FL | MN |

| 382.00 | FL | PA |

| 387.00 | FL | VA |

| 261.00 | FL | WA |

| 361.00 | GA | FL |

| 259.00 | GA | KS |

| 357.00 | GA | KY |

| 251.00 | GA | MI |

| 115.00 | GA | MN |

| 361.00 | GA | PA |

| 174.00 | GA | VA |

| 267.00 | GA | WA |

| 265.00 | HI | FL |

| 246.00 | HI | KS |

| 363.00 | HI | KY |

| 155.00 | HI | MI |

| 211.00 | HI | MN |

| 399.00 | HI | PA |

| 385.00 | HI | VA |

| 177.00 | HI | WA |

| 330.00 | KY | FL |

| 163.00 | KY | KS |

| 226.00 | KY | KY |

| 394.00 | KY | MI |

| 265.00 | KY | MN |

| 321.00 | KY | PA |

| 210.00 | KY | VA |

| 105.00 | KY | WA |

| 182.00 | LA | FL |

| 379.00 | LA | KS |

| 194.00 | LA | KY |

| 338.00 | LA | MI |

| 181.00 | LA | MN |

| 202.00 | LA | PA |

| 200.00 | LA | VA |

| 294.00 | LA | WA |

| 128.00 | ME | FL |

| 135.00 | ME | KS |

| 259.00 | ME | KY |

| 224.00 | ME | MI |

| 316.00 | ME | MN |

| 340.00 | ME | PA |

| 357.00 | ME | VA |

| 245.00 | ME | WA |

| 358.00 | MS | FL |

| 157.00 | MS | KS |

| 309.00 | MS | KY |

| 162.00 | MS | MI |

| 103.00 | MS | MN |

| 349.00 | MS | PA |

| 176.00 | MS | VA |

| 176.00 | MS | WA |

| 168.00 | NC | FL |

| 178.00 | NC | KS |

| 283.00 | NC | KY |

| 129.00 | NC | MI |

| 369.00 | NC | MN |

| 346.00 | NC | PA |

| 398.00 | NC | VA |

| 117.00 | NC | WA |

| 361.00 | ND | FL |

| 292.00 | ND | KS |

| 328.00 | ND | KY |

| 335.00 | ND | MI |

| 348.00 | ND | MN |

| 131.00 | ND | PA |

| 126.00 | ND | VA |

| 308.00 | ND | WA |

| 286.00 | NE | FL |

| 360.00 | NE | KS |

| 346.00 | NE | KY |

| 123.00 | NE | MI |

| 334.00 | NE | MN |

| 123.00 | NE | PA |

| 308.00 | NE | VA |

| 297.00 | NE | WA |

| 126.00 | NH | FL |

| 216.00 | NH | KS |

| 117.00 | NH | KY |

| 126.00 | NH | MI |

| 389.00 | NH | MN |

| 364.00 | NH | PA |

| 173.00 | NH | VA |

| 224.00 | NH | WA |

| 173.00 | NM | FL |

| 261.00 | NM | KS |

| 353.00 | NM | KY |

| 232.00 | NM | MI |

| 294.00 | NM | MN |

| 104.00 | NM | PA |

| 332.00 | NM | VA |

| 241.00 | NM | WA |

| 307.00 | NY | FL |

| 180.00 | NY | KS |

| 231.00 | NY | KY |

| 369.00 | NY | MI |

| 338.00 | NY | MN |

| 265.00 | NY | PA |

| 332.00 | NY | VA |

| 355.00 | NY | WA |

| 311.00 | PA | FL |

| 127.00 | PA | KS |

| 158.00 | PA | KY |

| 381.00 | PA | MI |

| 150.00 | PA | MN |

| 271.00 | PA | PA |

| 218.00 | PA | VA |

| 307.00 | PA | WA |

| 141.00 | RI | FL |

| 219.00 | RI | KS |

| 339.00 | RI | KY |

| 121.00 | RI | MI |

| 143.00 | RI | MN |

| 277.00 | RI | PA |

| 281.00 | RI | VA |

| 323.00 | RI | WA |

| 145.00 | SD | FL |

| 298.00 | SD | KS |

| 201.00 | SD | KY |

| 361.00 | SD | MI |

| 142.00 | SD | MN |

| 357.00 | SD | PA |

| 210.00 | SD | VA |

| 335.00 | SD | WA |

| 128.00 | TX | FL |

| 115.00 | TX | KS |

| 104.00 | TX | KY |

| 124.00 | TX | MI |

| 282.00 | TX | MN |

| 337.00 | TX | PA |

| 126.00 | TX | VA |

| 340.00 | TX | WA |

| 374.00 | UT | FL |

| 264.00 | UT | KS |

| 363.00 | UT | KY |

| 249.00 | UT | MI |

| 236.00 | UT | MN |

| 320.00 | UT | PA |

| 203.00 | UT | VA |

| 215.00 | UT | WA |

| 261.00 | VT | FL |

| 274.00 | VT | KS |

| 163.00 | VT | KY |

| 100.00 | VT | MI |

| 177.00 | VT | MN |

| 326.00 | VT | PA |

| 224.00 | VT | VA |

| 184.00 | VT | WA |

| 215.00 | WA | FL |

| 341.00 | WA | KS |

| 213.00 | WA | KY |

| 397.00 | WA | MI |

| 382.00 | WA | MN |

| 246.00 | WA | PA |

| 362.00 | WA | VA |

| 242.00 | WA | WA |

| 156.00 | WV | FL |

| 162.00 | WV | KS |

| 268.00 | WV | KY |

| 167.00 | WV | MI |

| 391.00 | WV | MN |

| 322.00 | WV | PA |

| 244.00 | WV | VA |

| 392.00 | WV | WA |

| 136.00 | WY | FL |

| 328.00 | WY | KS |

| 274.00 | WY | KY |

| 365.00 | WY | MI |

| 138.00 | WY | MN |

| 238.00 | WY | PA |

| 290.00 | WY | VA |

| 120.00 | WY | WA |

+--------+----------------+----------------+

217 rows in set (0.12 sec)

mysql> #Question: Show the total cost of products for each supplier.

mysql> CREATE INDEX cost\_idx ON supplier(name(4));

Query OK, 0 rows affected (0.04 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> SELECT supplier.name AS 'Supplier Name',

-> product.name AS 'Product Name',

-> SUM(price1) AS 'Total cost of products'

-> FROM supplier

-> JOIN sales\_fact ON supplier.supplierSK = sales\_fact.supplierSK\_FK

-> JOIN product ON product.productSK = sales\_fact.supplierSK\_FK

-> WHERE supplier.name IS NOT null

-> GROUP BY 1;

+-------------------------------+---------------------------------+------------------------+

| Supplier Name | Product Name | Total cost of products |

+-------------------------------+---------------------------------+------------------------+

| Afg Industries Inc. | Bandage Manufacturing Equip | 5254036.49 |

| American General Ventures Inc | Chalmers Polishing Equipment | 6932745.99 |

| Black Hills Corporation | Defeated Tray Supplies | 5213020.10 |

| Dollar General Corporation | Enhanced Covers | 2756478.59 |

| Fedders Corporation | Just Paper Supplies | 3075449.91 |

| First Bancshares Inc Mo | Pervasively Manufacturing Equip | 6471765.29 |

| Immunex Corporation | Significants Freezing Chemicals | 1108081.61 |

| Saurer Gruppe Holding Ag | Suing Manufacturing Equip | 1007714.40 |

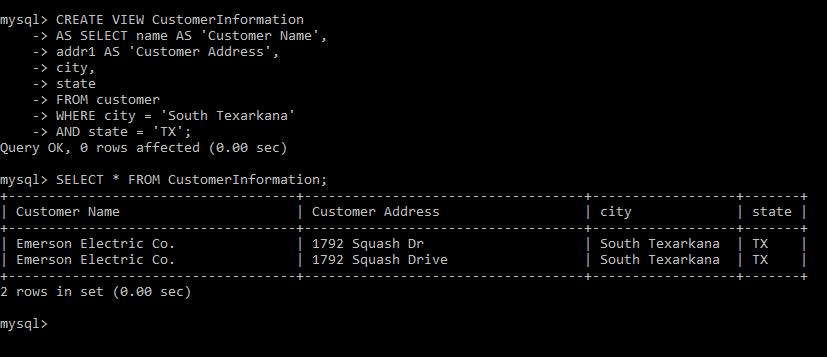
| Sinclair Broadcast Group Inc. | Tailor Jacks | 3830147.41 |

+-------------------------------+---------------------------------+------------------------+

9 rows in set (0.23 sec)

mysql> notee

## 2. A View



## 3. Aggregated Mata Marts

# 

# **Lost:**

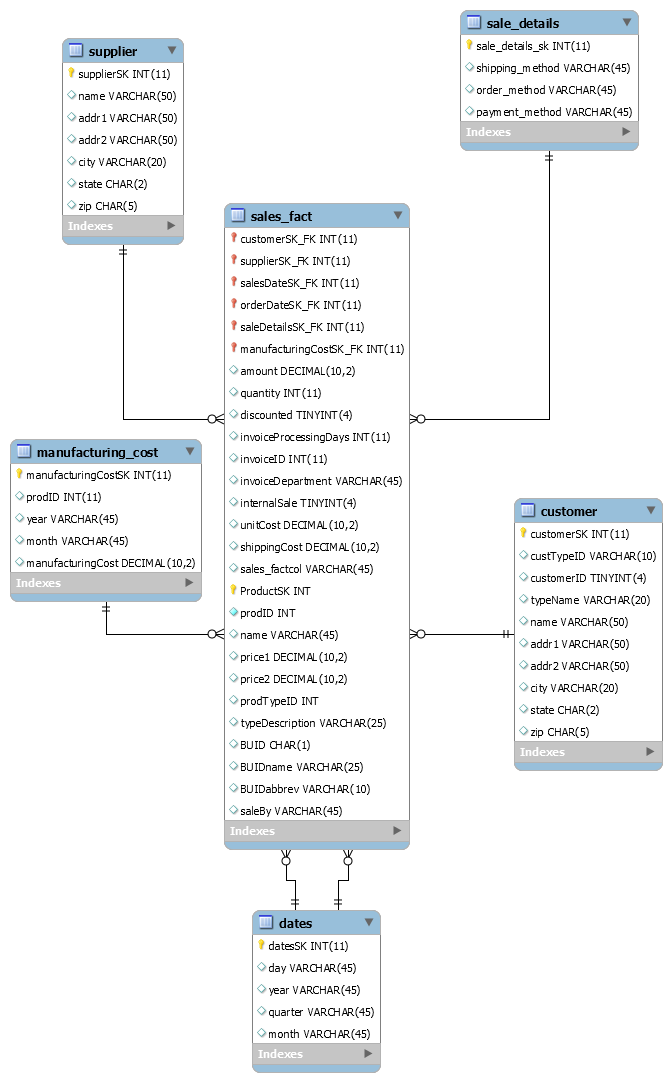
Use Case:

The company decides that they don’t need this dimension for the queries that they are running. Therefore, the dimension is only taking up unneeded space and we want this data mart to be as user-centric as possible.

Sample Queries:

SELECT customer.name, product.name FROM sales\_fact JOIN customer ON sales\_fact.customerSK\_FK = customer.customerSK JOIN product ON sales\_fact.productSK\_FK = product.productSK WHERE prodID=25;

SELECT supplier.name, dates.quarter FROM sales\_fact JOIN supplier ON sales\_fact.customerSK\_FK = customer.customerSK JOIN product ON sales\_fact.productSK\_FK = product.productSK WHERE supplier.zipcode=14826;

****

**Collapsed:**

Use Case:

The product data is used in every query. To speed up the queries, the product data is added to the fact table so that the product data does not need to be joined during every query.

Sample Queries:

SELECT product.name, amount FROM sales\_fact WHERE amount > 100;

SELECT product.name, unitCost, shippingCost FROM sales\_fact WHERE unitCost < 20;

# 

# 

# VIII. Handling Slowly Changing Dimensions (SCD)

**Reference Files/Folders:**

All\_SCD\_Files\_Final.zip

**SCD1**

**Change Type:**

Slowly Changing Dimension Type 1

**Before Image:**

****

**After Image:**

****

**Rationale for change:**

For this situation we were pretending that the name of the product contained a misspelled word. So we overwrote the name to be correct.

**SCD2**

**Change Type:**

Slowly Changing Dimension Type 2

**Before Image:**

****

**After Image:**

****

**Rationale for change:**

For this situation we were pretending that the BUID of the product was changed. So as this was a type 2 change the original record remains and we added an additional record to reflect the change made.

**SCD3**

**Change Type:**

Slowly Changing Dimension Type 3

**Before Image:**

****

**After Image:**

****

**Rationale for change:**

For this situation we were pretending that the Name of the product needed to be changed. So as this was a type 3 change we updated the name column to ‘previousName’ and then added an additional column called ‘newName’ so as to preserve the old name.

**SCD4**

**Change Type:**

Slowly Changing Dimension Type 1

**Before Image:**

****

**After Image:**

****

**Rationale for change:**

For this situation we were pretending that the ProdID was incorrect. This was a type 1 change so we just updated the column to the new ProdID.

**SCD5**

**Change Type:**

Slowly Changing Dimension Type 6

**Before Image:**

****

****

**After Image:**

****

**Rationale for change:**

For this situation we were pretending that the Price1 needed to be updated. So as this was a type 6 change we had to create the columns ‘preivousPrice1’ and ‘newPrice1’. Then we inserted a record into the table with the previous price 1 being equal to the current and new price 1. After that we inserted another new record into the table with the previous price 1 being equal to the first record then a new price 1 where the price was changed.

# IX. Many-to-Many (N-M) Relationship Implementation Option

We are proposing to have to separate dimensional table for shipping and supplier, each with a foreign key reference in the fact table. This implementation is shown in the ER diagram below.

Another option would have been to have a single table for shipping/supplier with only the shippingID, supplierID, and shippingSupplierSK. The fact table would have shippingSupplierSK as a foriegn key and part of the primary key. Then, the shippingSupplier table would have a foreign key reference to a shipping table and a supplier table.

We chose this method because it keeps the simple one-to-many relationships between the tables. This makes quicker because there are less joins than using the snowflake model.

We reference “Many-to-Many Relationships in the Data Warehouse” by Michelle A.

Poolet, Jun 26, 2008 from http://www.itprotoday.com/microsoft-sql-server/many-many-

relationships-data-warehouse. The article is attached in the appendix.

# 

# X. Appendix (Fix Lab #3 Problems)

|  |  |  |
| --- | --- | --- |
| **Part 1** | **Correction(s) Made** | **Comments** |
| **Final Schema Design** | Added both the sales\_date dimension and order\_date dimension instead of having the single dates dimension with an explanation that it represents the role playing dimensions. | N/A |
| **Rationale For final schema design** | **Our Rationale:**  --Dimension Tables  order\_date:  The order\_date table was necessary to allow the user to query based on order\_date OR sales\_Date. This is a role playing dimension.  sales\_date:  The sales\_date table was necessary to allow the user to query based on order\_date OR sales\_Date. This is a role playing dimension.  supplier:  The supplier dimension is necessary because it allows the user to query based on information from the supplier.  customer:  The customer dimension is necessary because it allows the user to query based on information from the customer.  product:  The product dimension is necessary because it allows the user to query based on information from the product.  sales\_details:  This is our junk dimension which is a table of attributes that did not belong in our other dimensional tables or the fact table. The user needs these attributes for queries.  manufacturing\_costs:  The manufacturing\_costsdimension is necessary because it allows the user to query based on information from the manufacturing cost.  --Fact Table  sales\_fact:  The fact table provides all of the needed information to cover all of the types of queries that the user stated they would need in the lab documentation. | N/A |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 2** | **Correction(s) Made** | **Comments** | **Reference Image** |
| **PECproduct** | -The blank values are now set to (NA) for supplierName  -We added an additional cleansing step to remove duplicates.  -Calculate empty unit cost by dividing amt / qty  - Create field saleBy |  | calculating unit cost.PNG  Product Create Field SaleBy.PNG |
| **TPCWproduct** | - re-ordered pentaho icons so that states were formatted before printing out clean data  - Create field saleBy |  | Product Create Field SaleBy.PNG |
| **TPCWproduct** | -We added an additional cleansing step to remove duplicates. |  |  |
| **TPCWcustomer** | -We added an additional cleansing step to remove the commas and periods from the address field.  - We split the Dept and Suit to addr2  -Blank values for zip codes were replaced with NA |  | TPCWCustomer remove periods and commas.PNG  TPCW Customer split addresses.PNG |
| **PECcustomer** | - We split the PO box to addr2  -We added an additional cleansing step to remove the commas and periods from the address field. |  | TPCW Customer split addresses.PNG  TPCWCustomer remove periods and commas.PNG |
| **TPCinvoice** | -The output for the TPCinvoice\_clean now has the correct date format.  -We now correctly calculate the salesAmount | We pulled the clean csv from the wrong step in our previous submission |  |
| **Sales Details** | - We sorted the sales details rows and remove duplicate rows |  | Sales Details Unique Rows.PNG  Sales Details Sort Rows.PNG |
| **Supplier** | - We sorted the supplier rows and remove duplicate rows |  | Supplier Sort for Unique.PNG  Sales Details Unique Rows.PNG |
| **Rationale for changes in star schema from cleansing decisions** | - We added saleBy in the product table to track who processed the sale.  - We removed description from product because name and description both referenced the name of the product.  - We added addr2 to customer to split the extra address information into a separate field. | N/A |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 3** | **Correction(s) Made** | **Comments** | **Reference Image** |
| Customer Dimension | -We now split the address field.  -We now delete duplicates | N/A |  |
| Product Dimension | -We now correctly implement salesby.  -States are now in one format.  -We removed duplicates from supplier. | N/A |  |
| Junk Dimension | -Added additional cleaning steps to sales\_details to remove the extra rows. The actual rows should be closer to the expected now. | N/A |  |
| Fact Table | -We actually load the data into the fact table now. | N/A |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 4** | **Correction(s) Made** | **Comments** | **Reference Image** |
| Load Fact Table | -We actually load the data into the fact table now. | N/A | loading fact table.PNG |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 5** | **Correction(s) Made** | **Comments** | **Reference Image** |
| Define 3 User Queries | -Query 1  -Query 2  -Query 3 | N/A |  |

# XI. Appendix (Many-to-Many Reference)

**Many-to-Many Relationships in the Data Warehouse**

Sometimes, a data warehouse requires many-to-many (M:N) relationships. Market-basket analysis is a technique used to determine how items in the data warehouse are related. Retail businesses frequently use market-basket analysis to evaluate customer buying behavior.

[Michelle A. Poolet](http://www.itprotoday.com/author/michelle-poolet) | Jun 26, 2008

Just like any well-designed database, a data warehouse contains relationships. From everything that you’ve read in this series over the past year, you might be tempted to think that all relationships in a data warehouse are one-to-many (1:M), from the dimension to the fact table. However, that’s not true. Occasionally, a data warehouse requires a many-tomany (M:N) relationship.

You can successfully argue that the star schema is inherently a M:N relationship. So why are we having this discussion? Competition is forcing businesses to become more innovative, and this innovation can bring with it more products and services in combinations that might not have been considered before. This innovation also maps to increasingly complicated business models, and subsequently, to more complex data warehouse designs. A business intelligence (BI) analyst who knows how to evaluate customer buying behavior can extract information from the data warehouse to determine which combinations of products and services will entice a customer to make a purchase.

One technique the BI analysts use is called *marketbasket analysis*, which is the process of evaluating collections of items in the data warehouse to determine whether they’re somehow related. Market-basket analysis is frequently (but not exclusively) used in the retail sector. Information gleaned from market-basket analysis helps retailers understand buyers’ needs. Based on this analysis, retailers can better appeal to buyers by doing things such as reorganizing a store layout. For example, the idea of co-locating beer and baby diapers—so that when Dad runs to the market to pick up baby diapers, he also buys beer—was a finding that came out of early market-basket analysis efforts. Understanding the buyers’ mind-set can enable retailers to develop cross-promotional marketing programs, capture new buyers, trim inventories down to the products that sell best, and design intelligent sales events. As a data warehouse designer and DBA, you need to know how to model your database to support market-basket analysis so that your BI analysts can extract the information your business needs to beat the competition.

A good example for this discussion is a build-to-order checking account—a bank account in which customers select the features they want from a menu, essentially customizing the account and deciding how much they will pay per month in bank fees. Some features are free and some are not. The free features might include a check card with reward points (choose your favorite rewards program), online banking, bill-pay service, email alerts, no minimum balance, no direct deposit required, and identity theft insurance. Premium services might include no fees for using another bank’s ATM, rebates on ATM fees if you do use another bank’s services, interest-earning accounts, double rewards points, money back or extra rewards points on check-card purchases, anniversary cash bonuses, and overdraft fee forgiveness. Customers choose some or all of the free features and one or two of the premium features; these features are bundled into a customized checking account. Then customers can choose as many additional premium features to add to the custom account as they want for a fee of say, $2 per feature per month.

**The OLTP Database Schema**

If you were designing the OLTP database schema for a build-to-order checking account, it would look similar to Figure 1. This schema stores each account configuration, with general account information stored in the Account table and the features for each account stored in the AccountFeatures fact table. What’s not included in this figure are tables to capture the many types of transactions associated with an account, such as check-card purchases, purchases made with paper checks, bill-pay records, and ATM withdrawals.

Figure 1: An OLTP database schema for the build-to-order checking account

Let’s assume that your datasource looks similar to Figure 1, with Account and AccountDetail in a 1:M relationship. Now your company’s CFO and vice president of marketing want to know which customers’ accounts are using which features most often and in which combinations, if any. They want to measure the efficacy of the free features versus the premium features and determine which features sell the best, so they can better direct the bank’s marketing efforts. They also want to better understand their customers’ buying behavior. In other words, the CFO and vice president of marketing want to do market-basket analysis. Why does a customer pay for a purchase one time with a check, but pays for his or her next purchase with a check card? What are the circumstances surrounding these varying types of buying behavior? If the cost to process a paper check is a set fee per check, but the cost to process a check card transaction is a percentage of the total transaction amount, it might behoove the bank to figure out a way to encourage its customers— especially the big spenders—to pay with checks.

**The Star Schema**

To create the star schema that will help determine the answers to these questions, you’ll need a Transaction fact table, a dimension for the Account, and an AccountFeatures fact table. In addition, you’ll need categorical dimensions for the features, account category (e.g., free, premium), and the rewards program (e.g., airline, hotel, rental car). Complexity in the schema arises from the fact that to answer the CFO’s questions, a transaction needs to link to one or more features in the AccountFeatures fact table, and each of the Account- Feature records needs to link back to its parent account. This complexity mandates a M:N relationship between the Account dimension and the Transaction fact table, as shown in Figure 2. In addition, there’s a conventional 1:M relationship between the Account dimension and the Transaction fact table.

[Figure 2: A M:N relationship in the data warehouse](http://www.itprotoday.com/sites/sqlmag.com/files/uploads/2013/06/Poolet%20Fig2.jpg)

This schema is multifunctional because you can use it for different types of analyses. The Account, Location, Customer, and Time dimensions are all conformed, meaning they can participate in multiple star schemas. Taken as a unit, the Account, Customer, Location, and Time dimensions and the Transaction fact table comprise a standard star schema model, which enables BI analysts to create cubes from these tables and extract information without regard to which account feature was used to generate the transaction. The M:N relationship between the Account dimension and the Transaction fact table, which is manifested in AccountFeatures, is the structure that will enable market-basket analysis without having to create a separate star schema. By linking each transaction with the account feature that was used to generate the transaction, BI analysts can begin to find answers to the questions asked about customer buying behavior.

By not creating two separate star schemas (i.e., a Transaction star schema and an AccountFeatures star schema), you’re saving disk storage. If you were to combine the Account and AccountFeatures tables into a single dimension, that dimension would swell by a factor of 15—more if additional features are added to the build-to-order checking account program. In previous articles, such as “[Data Warehousing: Junk Dimensions](http://sqlmag.com/database-administration/data-warehousing-junk-dimensions),” May 2008, I have discussed the desirability of keeping dimensions tight and concise; adding features to the Account dimension would create a large, unwieldy dimension that wouldn’t function well when being cubed. Denormalizing the Transaction fact table by incorporating account feature attributes wouldn’t be the best idea, either. The fact table would contain the most records of all the tables in the schema; increasing its size and adding more keys would diminish performance and cause locking and blocking conflicts when the conventional BI analyst and the market-basket BI analyst run their programs simultaneously.

**Beyond Retail**

Market-basket analysis is an important operation that has uses that go beyond the retail sector. For example, you can discover relationships in a medical database, such as patients who had procedure X subsequently developing condition Y within n days/weeks/months. In a law enforcement or social profiling database, you might discover that people who engage in behavior A by age n have a d percent chance of engaging in behavior B for the rest of their life.

**Market-Basket Hero**

Market-basket/affinity analysis is predictive: It attempts to forecast future behavior based on past actions. Business is getting more and more competitive; in response to this pressure, they are becoming more innovative, offering product and service combinations that are derived from complex analysis of customers’ buying behaviors. You’ll need to be able to support market-basket efforts with your data warehouse design; understanding how to do so will make you a hero.

# XII. Appendix (Data Dictionary)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute**  **Name** | **Table**  **Name** | **Data Type** | **Description** | **Sample** |
| customerSK | customer | int(11) | Primary key for the customer table | 01 |
| custTypeID | customer | varchar(10) | The ID for the CustomerType table | “C” for “Commercial” |
| customerID | customer | tinyint(4) | The ID of the customer | 1 (or) 2 (or) 10 |
| typeName | customer | varchar(20) | The type of customer | Education |
| name | customer | varchar(50) | The name of the customer | Maya Brewer |
| addr 1 | customer | varchar(50) | The first address line of the customer | 6836 At Rd |
| addr 2 | customer | varchar(50) | The second address line of the customer | P.O. Box 825 |
| city | customer | varchar(20) | The city of the customer | Moraga |
| state | customer | varchar(20) | The official USPS state abbreviation | AZ |
| zip | customer | char(5) | The 5 digit USPS zip code | 70314 |
| supplierSK | supplier | int(11) | Primary key for the supplier table | 02 |
| name | supplier | varchar(50) | Name of the supplier | Specialty Teleconstructors Inc |
| addr1 | supplier | varchar(50) | Address 1 of the supplier | 2161 Chalmers Street |
| addr2 | supplier | varchar(50) | Address 2 of the supplier | 6037 Ecuadorian Street |
| city | supplier | varchar(20) | City of the supplier | Philadelphia |
| state | supplier | char(2) | State of the supplier | PA |
| zip | supplier | char(5) | Zip of the supplier | 19103 |
| orderDateSK | order\_date | int(11) | Primary key for the order\_date table | 03 |
| day | order\_date | varchar(45) | Day of the month of the order date | 21 |
| year | order\_date | varchar(45) | Year of the order date | 2017 |
| quarter | order\_date | varchar(45) | Quarter of the order date | 04 |
| month | order\_date | varchar(45) | Month of the order date | 03 |
| salesDateSK | sales\_date | int(11) | Primary key for the sales\_date table | 04 |
| day | sales\_date | varchar(45) | Day of the month of the sales date | 22 |
| year | sales\_date | varchar(45) | Year of the sales date | 2017 |
| quarter | sales\_date | varchar(45) | Quarter of the sales date | 03 |
| month | sales\_date | varchar(45) | Month of the sales date | 02 |
| productSK | product | int(11) | The primary key for the product table | 01 |
| prodID | product | int(11) | The ID of the product | 02 |
| name | product | varchar(45) | The name of the product | Whale Lifts |
| description | product | varchar(50) | The description of the product | Tailor Jackets |
| price1 | product | decimal(10,2) | The standard price of the product per unit | 10.00 |
| price2 | product | decimal(10,2) | The discounted price of the product per unit | 12.00 |
| prodTypeID | product | int(11) | The ID for the product type | 003 |
| typeDescription | product | varchar(25) | The type description of the product | Lifts |
| BUID | product | char(1) | The ID for business unit | ABCD |
| BUIDname | product | varchar(25) | The name of the business unit | Processing Equipment |
| BUIDabbrev | product | varchar(10) | The abbreviation of the business unit ID | Equipment |
| saleBy | product | varchar(45) | The division who sell the product | TPCW |
| sales\_details\_SK | sales\_details | int(11) | Primary key for the supplier table | 05 |
| shipping\_method | sales\_details | varchar(45) | Shipping method for the sale | air |
| order\_method | sales\_details | varchar(45) | Order method for the sale | email |
| payment\_method | sales\_details | varchar(45) | Payment method for the sale | COD |
| manufacturingCostSK | manufacturing\_cost | int(11) | Primary key fot eh manufacturing\_cost table | 06 |
| prodID | manufacturing\_cost | int(11) | Id of the product that the manufacturing data is for | 01 |
| year | manufacturing\_cost | varchar(45) | The year the manufacturing data is for | 2017 |
| month | manufacturing\_cost | varchar(45) | The month the manufacturing data is for | 03 |
| manufacturingCost | manufacturing\_cost | decimal(10,2) | The cost of the manufacturing data | 13.57 |
| productSK | sales\_fact | int(11) | The foreign key that references the primary key for the product table | 1 (or) 2 (or) 3 |
| customerSK | sales\_fact | int(11) | The foreign key that references the primary key for the customer table | 1 (or) 2 (or) 3 |
| supplierSK | sales\_fact | int(11) | The foreign key that references the primary key for the supplier table | 1 (or) 2 (or) 3 |
| salesDateSK | sales\_fact | int(11) | The foreign key that references the primary key for the sales\_date table | 1 (or) 2 (or) 3 |
| orderDateSK | sales\_fact | int(11) | The foreign key that references the primary key for the order\_date table | 1 (or) 2 (or) 3 |
| saleDetailsSK | sales\_fact | int(11) | The foreign key that references the primary key for the sale\_details table | 1 (or) 2 (or) 3 |
| manufacturingCostSK | sales\_fact | int(11) | The foreign key that references the primary key for the manufacturing\_cost table | 1 (or) 2 (or) 3 |
| amount | sales\_fact | decimal(10,2) | The amount of sales | 13.01 |
| quantity | sales\_fact | int(11) | The sales quantity | 1 |
| discounted | sales\_fact | tinyint(4) | The boolean value whether or not product is discounted | Either 0 or 1 |
| invoiceProcessingDays | sales\_fact | int(11) | The processing days for the product | 10 |
| invoiceID | sales\_fact | int(11) | The ID for the invoice | 1 |
| invoiceDepartment | sales\_fact | varchar(45) | The department of the invoice | PEC |
| internalSale | sales\_fact | tinyint(4) | Products that are supplied by TPCW, TPCE, or PEC | 0 or 1 |
| unitcost | sales\_fact | decimal(10,2) | The standard cost of the product per unit | 11.00 |
| shippingCost | sales\_fact | decimal(10,2) | The cost of the shipping of a product | 12.00 |