**Data Management & Business Intelligence**



**Team Project Assignment:**

**ETL DESIGN FOR ABC COMPANY**

**Team Members:**

Andrew Yap Yiing Hann (G1902209K)

Lim Guo Wei (G1901873A)

Low Zhen Hui Gabriel (G1902357H)

Kong Jia Xin Willa (G1901857D)

Lee Sin Tat (G1800658J)

**Table of Contents**

[**1.** **Description of Business** 3](#_Toc27310882)

[1.1 Summary of the Business Process 3](#_Toc27310883)

[1.2 Key Process Assumptions 3](#_Toc27310884)

[**2.** **Business Requirements for Database** 4](#_Toc27310885)

[2.1. Requirements for OLTP 4](#_Toc27310886)

[2.2. Requirements for OLAP 4](#_Toc27310887)

[**3.** **OLTP Design** 5](#_Toc27310888)

[3.1 Entity-Relationship Diagram 5](#_Toc27310889)

[3.2 Table Structure 6](#_Toc27310890)

[3.2.1 Supplier 6](#_Toc27310891)

[3.2.2 Supplier\_Part 6](#_Toc27310892)

[3.2.3 Parts 7](#_Toc27310893)

[3.2.4 Part\_Assy 7](#_Toc27310894)

[3.2.5 Assemblies 7](#_Toc27310895)

[3.2.6 Work\_Order 8](#_Toc27310896)

[3.2.7 Stations\_WO 8](#_Toc27310897)

[3.2.8 Assembly\_Stations 9](#_Toc27310898)

[3.2.9 Invoice 9](#_Toc27310899)

[3.2.10 Customers 9](#_Toc27310900)

[3.2.11 Customer\_Order 10](#_Toc27310901)

[3.2.12 Order\_Assembly 10](#_Toc27310902)

[**4.** **OLAP Design** 11](#_Toc27310903)

[4.1 Entity-Relationship Diagram 11](#_Toc27310904)

[4.2 Table Structure 12](#_Toc27310905)

[4.2.1 Fact\_Part\_Usage 12](#_Toc27310906)

[4.2.2 Fact\_Assembly\_Time 12](#_Toc27310907)

[4.2.3 Fact\_Sales 12](#_Toc27310908)

[4.2.4 Dim\_Time 13](#_Toc27310909)

[4.2.5 Dim\_Assembly\_Station 13](#_Toc27310910)

[4.2.6 Dim\_Part 13](#_Toc27310911)

[4.2.7 Dim\_Assembly 13](#_Toc27310912)

[**5.** **OLTP Data Generation** 14](#_Toc27310913)

[5.1 Customer and Supplier Information 14](#_Toc27310914)

[5.2 Parts and Price Information 14](#_Toc27310915)

[5.3 Timestamp Information 16](#_Toc27310916)

[**6.** **ETL Process** 17](#_Toc27310917)

[6.1 Loading Data into Dimension Tables 17](#_Toc27310918)

[6.1.1 Dim\_Time 17](#_Toc27310919)

[6.1.2 Dim\_Assembly\_Station 18](#_Toc27310920)

[6.1.3 Dim\_Part 18](#_Toc27310921)

[6.1.4 Dim\_Assembly 18](#_Toc27310922)

[6.2 Loading Data into Fact Tables 19](#_Toc27310923)

[6.2.1 Fact\_Part\_Usage 19](#_Toc27310924)

[6.2.2 Fact\_Assembly\_Time 20](#_Toc27310925)

[6.2.3 Fact\_Sales 20](#_Toc27310926)

[**7.** **Business Intelligence Dashboard Design** 21](#_Toc27310927)

[***Appendix A: Data Generation Scripts*** 24](#_Toc27310928)

# **Description of Business**

This ETL solution aims to provide a manufacturing company, **ABC Company,** a data warehouse for its supply chain, production and sales process and department users.

## Summary of the Business Process

Sales Department receives customer orders

Sales Department raise one work order per assembly based on customer orders

Supply Chain Department purchase parts to meet demand based on work order

Production Department starts work based on work order

Sales Department issues invoice to bill customers

***Figure 1****:* ***Summary of the Business Process***

## Key Process Assumptions

* One customer order can have one assembly.
* Each customer order will be assigned a work order number.
* One work order can only have one assembly.
* Each work order or **[WO\_No]****[[1]](#footnote-1)** is assigned to one assembly station.
* Each station can only assemble one **[Assembly\_No]1**.
* Supply Chain Department will purchase parts based on the amount required to complete all work orders.

# **Business Requirements** **for Database**

## Requirements for OLTP

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirements for OLTP** | **Priority** |
| 1 | To provide an OLTP solution for daily tasks involving the sales, supply chain and production departments. | High |
| 2 | Sales Department shall be able to receive customer orders and release work orders. | High |
| 3 | Sales Department shall be able to access customer information. | High |
| 4 | Supply Chain Department shall have access to the work orders. | High |
| 5 | Supply Chain Department shall be able to monitor the quantity available for each part. | High |
| 6 | Supply Chain Department shall be able to access supplier information. | High |
| 7 | Production department shall have access to work orders. | High |
| 8 | Production department shall be able to assign work orders to stations. | High |
| 9 | Production department shall be able to record start and end times for each work order. | High |

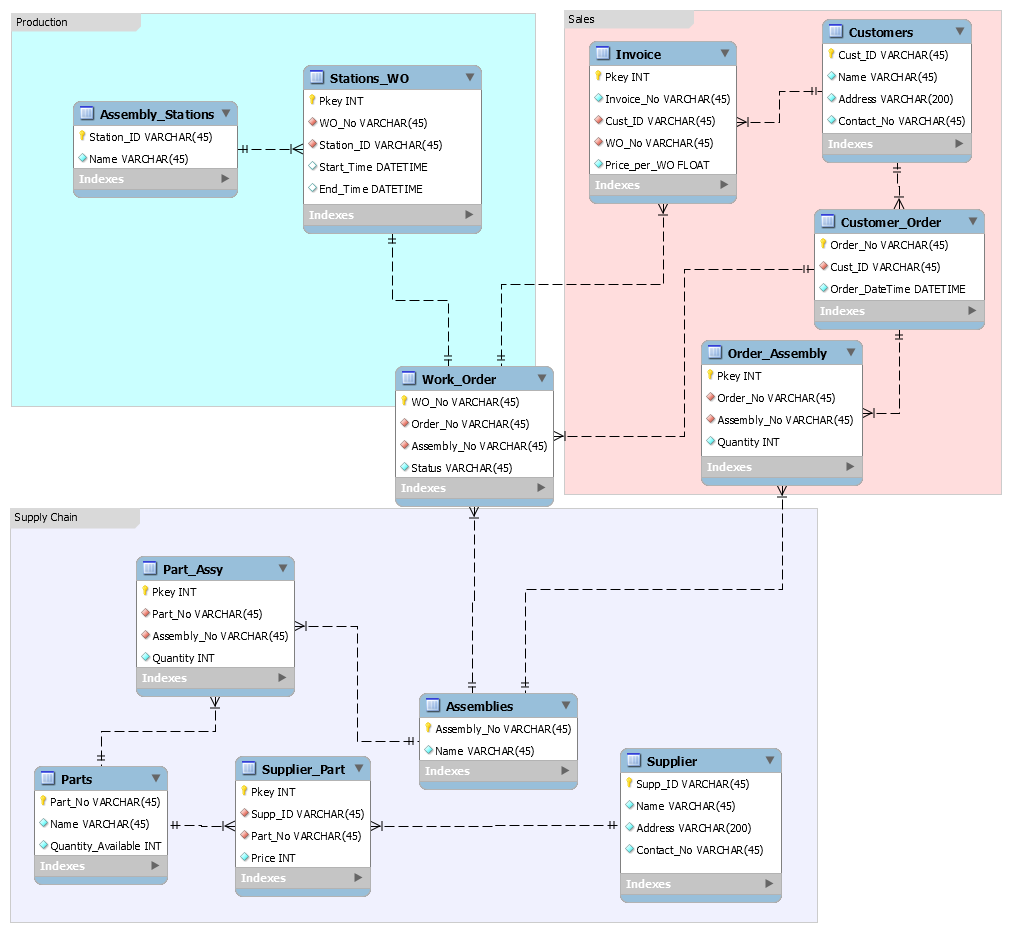
## Requirements for OLAP

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirements for OLAP** | **Priority** |
| 1 | To provide an OLAP solution to facilitate analytics for sales, production and supply chain departments. | High |
| 2 | Revenue statistics to be tracked at an assembly-level. | High |
| 3 | Revenue breakdown into number of units sold and average revenue of product to be tracked. | High |
| 4 | Assembly time trend can be tracked for improvement potential. This include determining the assembly that takes the shortest time to produce. | High |
| 5 | Revenue per assembly time ($/hour) trend to be tracked to determine revenue efficiency of production. | High |
| 6 | Revenue breakdown by assembly | Medium |

# **OLTP Design**

## Entity-Relationship Diagram

The OLTP was designed to address the user needs listed under ***Section 2.1 – Requirements for OLTP***. Refer to the OLTP Entity-Relationship (ER) Diagram below:



***Figure 3: OLTP ER Diagram***

## Table Structure

The following acronyms will be used to describe constraints.

|  |  |  |
| --- | --- | --- |
| **S/N** | **Constraint** | **Acronym** |
| 1 | PKEY | PK |
| 2 | NOT NULL | NN |
| 3 | UNIQUE | UQ |
| 4 | FOREIGN KEY | FK |

### Supplier

This table records information for all suppliers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Supp\_ID | VARCHAR(45) | PK | Supplier ID |
| 2 | Name | VARCHAR(45) | NN | Supplier Name |
| 3 | Address | VARCHAR(200) | NN | Supplier Address |
| 4 | Contact\_No | VARCHAR(45) | NN | Supplier Contact Number |

### Supplier\_Part

This table records the many-to-many relationship between “**Supplier**” and “**Parts**” tables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Primary Key |
| 2 | Supp\_ID | VARCHAR(45) | NN, UQ, FK | FK referencing **Supplier** table |
| 3 | Part\_No | VARCHAR(45) | NN, UQ, FK | FK referencing **Parts** table |
| 4 | Price | FLOAT | NN | Price of part |

### Parts

This table contains a list of parts and their quantities available.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Part\_No | VARCHAR(45) | PK | Part Number |
| 2 | Name | VARCHAR(45) | NN | Part Name |
| 3 | Quantity\_Available | INT | NN | Quantitiy available |

### Part\_Assy

This table records the many-to-many relationship between “**Parts**” and “**Assemblies**” tables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Primary Key |
| 2 | Part\_No | VARCHAR(45) | NN, UQ, FK | FK referencing **Parts** table |
| 3 | Assembly\_No | VARCHAR(45) | NN, UQ, FK | FK referencing **Assemblies** table |
| 4 | Quantity | INT | NN | Quantity of parts required per assembly |

### Assemblies

This table contains a list of all assemblies.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Assembly\_No | VARCHAR(45) | PK | Assembly Number |
| 2 | Name | VARCHAR(45) | NN | Assembly Name |

### Work\_Order

This table assigns a **[WO\_No]** to each assembly in the **[Order\_No]**. The Status column monitors the status of the work order.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | WO\_No | VARCHAR(45) | PK | Work Order Number |
| 2 | Order\_No | VARCHAR(45) | NN, FK | FK referencing **Customer\_Order** table |
| 3 | Assembly\_No | VARCHAR(45) | FK | FK referencing **Assemblies** table |
| 4 | Status | VARCHAR(45) | NN | Status of the Work Order, i.e Open or Close |

### Stations\_WO

This table records the many-to-one relationship between “**Assembly\_Stations**” and “**Work\_Order**” tables. It also records the start and end time of each work order.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Pkey |
| 2 | WO\_No | VARCHAR(45) | NN, FK | FK referencing **Work\_Order** table |
| 3 | Station\_ID | VARCHAR(45) | FK | FK referencing **Assemblies** table |
| 4 | Start\_Time | DATETIME | - | Assembly start time |
| 5 | End\_Time | DATETIME | - | Assembly end time |

### Assembly\_Stations

This table contains a list of all Assembly stations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Station\_ID | VARCHAR(45) | PK | Station ID |
| 2 | Name | VARCHAR(45) | NN | Station Name |

### Invoice

This table records all “**Invoice\_No**” and the “**WO\_No**” that is included.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Pkey |
| 2 | Invoice\_No | VARCHAR(45) | NN, UQ | Invoice number |
| 3 | Cust\_ID | VARCHAR(45) | NN, UQ, FK | FK referencing **Customers** table |
| 4 | WO\_No | VARCHAR(45) | NN, UQ, FK | FK referencing **Work\_Order** table |
| 5 | Price\_per\_WO | FLOAT | NN | Price per work order |

### Customers

This table contains information on all customers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Cust\_ID | VARCHAR(45) | PK | Customer ID |
| 2 | Name | VARCHAR(45) | NN | Customer Name |
| 3 | Address | VARCHAR(200) | NN | Customer Address |
| 4 | Contact\_No | VARCHAR(45) | NN | Contact Number |

### Customer\_Order

This table contains information on customer orders.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Order\_No | VARCHAR(45) | PK | Order Number |
| 2 | Cust\_ID | VARCHAR(45) | NN | FK referencing **Customers** table |
| 3 | Order\_DateTime | DATETIME | NN | Order date and time |

### Order\_Assembly

This table records the many-to-many relationship between “**Customer\_Order**” and “**Assemblies**” tables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Pkey |
| 2 | Order\_No | VARCHAR(45) | NN, UQ, FK | Order Number |
| 3 | Assembly\_No | VARCHAR(45) | NN, UQ, FK | Assembly Number |
| 4 | Quantity | INT | NN | Quantity ordered |

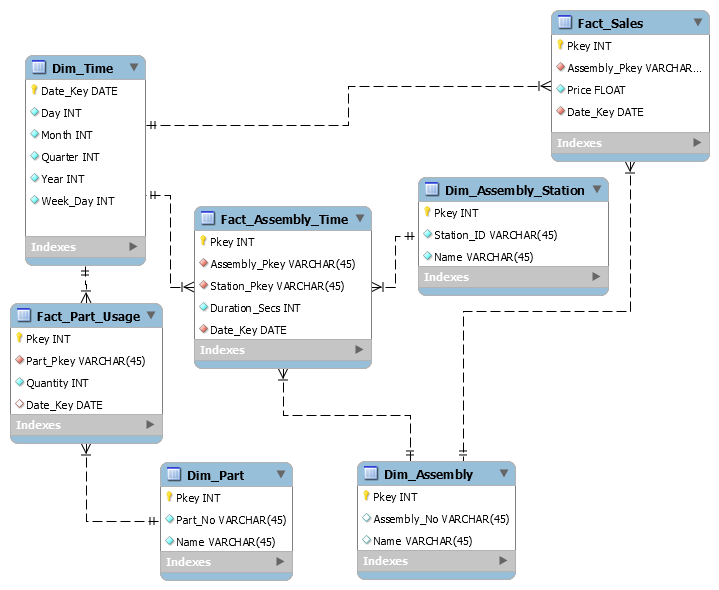
# **OLAP Design**

The data warehouse is designed to answer the following questions:

* Which station is the quickest in completing assemblies?
* Which assembly generates the highest sales?
* Which part is the most in demand?

## Entity-Relationship Diagram

Refer to the OLAP Entity-Relationship (ER) Diagram below:



**Figure 4: OLAP ER Diagram**

## Table Structure

### Fact\_Part\_Usage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Primary Key |
| 2 | Part\_Pkey | VARCHAR(45) | FK, NN | FK referencing Pkey of **Dim\_Part** table |
| 3 | Quantity | INT | NN | Quantity of parts used |
| 4 | Date\_Key | DATE | FK, NN | FK referencing Date \_Key of **Dim\_Time** table |

### Fact\_Assembly\_Time

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Primary Key |
| 2 | Assembly\_Pkey | VARCHAR(45) | FK, NN | FK referencing Pkey of **Dim\_Assembly** |
| 3 | Station\_Pkey | VARCHAR(45) | FK, NN | FK referencing Pkey of **Dim\_Assembly\_Station** |
| 4 | Duration\_Secs | INT | NN | Duration to complete Assembly in seconds |
| 5 | Date\_Key | DATE | FK, NN | FK referencing **Dim\_Time** table |

### Fact\_Sales

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Primary Key |
| 2 | Assembly\_Pkey | VARCHAR(45) | FK, NN | FK referencing Pkey of **Dim\_Assembly** |
| 4 | Price | FLOAT | NN | Total price |
| 5 | Date\_Key | DATE | FK, NN | FK referencing **Dim\_Time** table |

### Dim\_Time

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Date\_Key | DATE | PK | Primary Key |
| 2 | Day | INT | NN | Day of the month |
| 3 | Month | INT | NN | Month |
| 4 | Quarter | INT | NN | Quarter |
| 5 | Year | INT | NN | Year |
| 6 | Week\_Day | INT | NN | Day of the week. 0 = Mon |

### Dim\_Assembly\_Station

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Primary Key |
| 2 | Station\_ID | VARCHAR(45) | NN, UQ | Station ID |
| 3 | Name | VARCHAR(45) | NN | Station Name |

### Dim\_Part

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Primary Key |
| 2 | Part\_No | VARCHAR(45) | NN, UQ | Part Number |
| 3 | Name | VARCHAR(45) | NN | Part Name |

### Dim\_Assembly

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Column Name** | **Data Type** | **Constraints** | **Remarks** |
| 1 | Pkey | INT | PK | Surrogate Primary Key |
| 2 | Assembly\_No | VARCHAR(45) | NN, UQ | Assembly Number |
| 3 | Name | VARCHAR(45) | NN | Assembly Name |

# **OLTP Data Generation**

Python language was mainly used to generate synthetic data in consideration of business requirements and conformity to database schema and integrity constraints. The datasets were output into .csv format and loaded into their respective tables within the database. Data stored in tables (“**Assembly\_Stations, Part\_Assy and Assemblies**”), which was expected to contain only a couple of data records, were created directly using Excel spreadsheet. Refer to ***Appendix A*** for the python scripts.

Assumptions relating to the business aspect of Manufacturing Company ABC is as described. The company produces five types of electronics, namely radio, television, watch, clock, and laptop. Only one type of electronics is assembled in each assembly stations. Some of the parts required to form each product are also used for assembling other products. The parts are sourced from a pool of suppliers who may be offering similar sets of items. The company business has been running from 2015 to 2019.

## Customer and Supplier Information

Customer names, supplier names and address information were generated using Faker package while the phone numbers were produced using “**Faker-e164**” package. A total of 500 customers’ and 40 suppliers’ information were generated using this approach. Duplicated names were dropped to preserve authenticity. The links to the abovementioned packages are as followed:

* Faker: <https://github.com/joke2k/faker>
* Faker-e164: <https://github.com/crowdcomms/faker-e164>

## Parts and Price Information

The price of the parts offered by the suppliers was sampled from Gaussian distribution. The mean and standard deviation of the price of each part were specified and the price was returned randomly using “**NumPy.random.normal**” package. This approach reduced the likelihood of any same product being offered at the same price by different supplier. In this case, the price generated would be a better representation of the real-world competitive market.

The selling price of all products assembled by Manufacturing Company ABC was created to increase slightly over the years. For example, the price of the laptop increases from $3200 to $3500 from 2015 to 2019. This was performed via a deterministic approach which involved using simple Python If-else statements to draw predefined values stored in Python dictionaries.

## Timestamp Information

The datetime information of the customers’ orders were randomly generated to be within 2015 and 2019. A total of 600 customers’ orders were made during this time frame by 500 different customers. Both the “**Customers**” and “**Customer\_Order**” tables were created based on the following logic:

* Each customer must made at least one order
* 20% of the overall customers are return customer who have made at least two orders over the years

The start time required for each work order recorded in the “**Stations\_WO**” table was created while assuming that each work order starts immediately when customers place their order. The time taken to complete each product was sampled using Python inbuilt “**random.choices”** package. For example, the time taken to complete assembling a radio would be either 2, 3 or 4 days with a relative weight sequence of 8, 3 and 1 respectively. In other words, 4 days will be required to assemble a single radio 1 out of 12 times.

# **ETL Process**

To simplify the project, the ETL process will be done based on the following conditions:

* OLTP and OLAP tables will reside in the same schema;
* OLAP tables are identified by **fact** or **dim** at the front of the table name;
* Data is loaded from OLTP to OLAP tables via SQL functions or statements.

Data will be first loaded into the dimension tables followed by the fact tables.

## Loading Data into Dimension Tables

This section explains how the data can be loaded from the OLTP tables to the OLAP Dimension tables.

### Dim\_Time

Firstly, a DATE table was created with a single column called “**Date**”. A python script “**datetime.py**” was written to generate SQL INSERT statements into “**DATE**” table. SQL statements were then used to populate the “**Dim\_Time**” table.

DDL to create DATE table:

|  |
| --- |
| CREATE TABLE **DATE**(Date DATE); |

Python script to insert into “**DATE**” table:

|  |
| --- |
| from datetime import date, timedelta  d1 = date(2015, 1, 1) # start date  d2 = date(2019, 12, 31) # end date  delta = d2 - d1 # timedelta  with open("INSERT\_DATE.txt","w") as text\_file:  for i in range(delta.days + 1):  print("INSERT INTO DATE VALUES ('"+str(d1 + timedelta(i))+"');",file=text\_file) |

**6.1.1 Dim\_Time (Cont’d)**

Insert statements to populate “**Dim\_Time**”:

|  |
| --- |
| INSERT INTO DIM\_TIME (DateKey,Day,Month,Quarter,Year,Week\_Day)  SELECT date as DateKey,  DAY(date) as DayNumber,  MONTH(date) as MonthNumberOfYear,  QUARTER(date) as CalendarQuarter,  YEAR(date) as CalendarYear,  WEEKDAY(date) as WeekDay  FROM date; |

### Dim\_Assembly\_Station

“**Dim\_Assembly\_Station**” is populated by loading data from “**Assembly\_Stations**” table.

|  |
| --- |
| INSERT INTO DIM\_ASSEMBLY\_STATION (STATION\_ID,NAME)  SELECT STATION\_ID,NAME  FROM ASSEMBLY\_STATIONS; |

### Dim\_Part

“**Dim\_Part**” is populated by loading data from “**Parts**” table.

|  |
| --- |
| INSERT INTO DIM\_PART (PART\_NO,NAME)  SELECT PART\_NO, NAME FROM PARTS; |

### Dim\_Assembly

“**Dim Assembly**” is populated by loading data from “**Assemblies**” table:

|  |
| --- |
| INSERT INTO DIM\_ASSEMBLY (ASSEMBLY\_NO,NAME)  SELECT ASSEMBLY\_NO,NAME  FROM assemblies; |

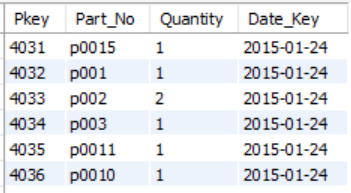
## Loading Data into Fact Tables

### Fact\_Part\_Usage

This table is used to capture information on part usage over time. The ETL script is as shown:

|  |
| --- |
| DELIMITER //  DROP PROCEDURE part\_usage;  CREATE PROCEDURE part\_usage()  BEGIN  DECLARE d DATE;  SET d='2015-01-01';  WHILE d<'2019-12-31' DO  INSERT INTO fact\_part\_usage (Part\_No,Quantity,date\_key)  SELECT pa.Part\_No,sum(quantity\*  assembly\_counts)part\_counts,d  FROM part\_assy pa JOIN (  SELECT date(sw.End\_Time),wo.Assembly\_No, COUNT(Assembly\_No) assembly\_counts  FROM stations\_wo sw JOIN work\_order wo ON sw.wo\_no=wo.wo\_no  WHERE date(sw.end\_time)<d  GROUP BY Assembly\_No  )assy\_count ON pa.assembly\_no=assy\_count.Assembly\_No  GROUP BY pa.part\_no;  SET d=DATE\_ADD(d, INTERVAL 1 DAY);  END WHILE;  END//  DELIMITER ;  CALL part\_usage(); |

The picture below shows a snapshot of the “**Fact\_Part\_Usage**” fact table.

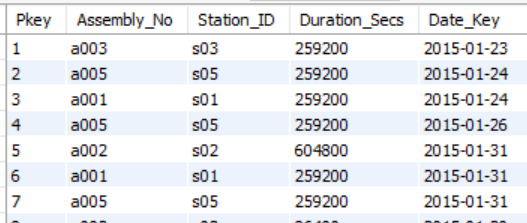


### Fact\_Assembly\_Time

This fact table captures the variation of assembly duration over time. The ETL script is as shown:

|  |
| --- |
| INSERT INTO fact\_assembly\_time (Assembly\_No, Station\_ID, Duration\_Secs, Date\_Key)  SELECT wo.Assembly\_No, sw.station\_id,timestampdiff(second,sw.Start\_Time,sw.End\_Time) duration, date(sw.end\_time) date  FROM stations\_wo sw  JOIN work\_order wo ON sw.wo\_no=wo.wo\_no; |

The picture below is a snapshot of the “**Fact\_Assembly\_Time**” fact table.

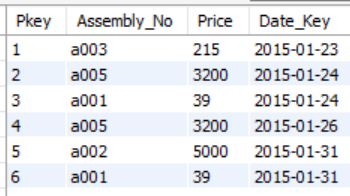


### Fact\_Sales

This fact table captures assembly sales over time. The ETL script is as shown:

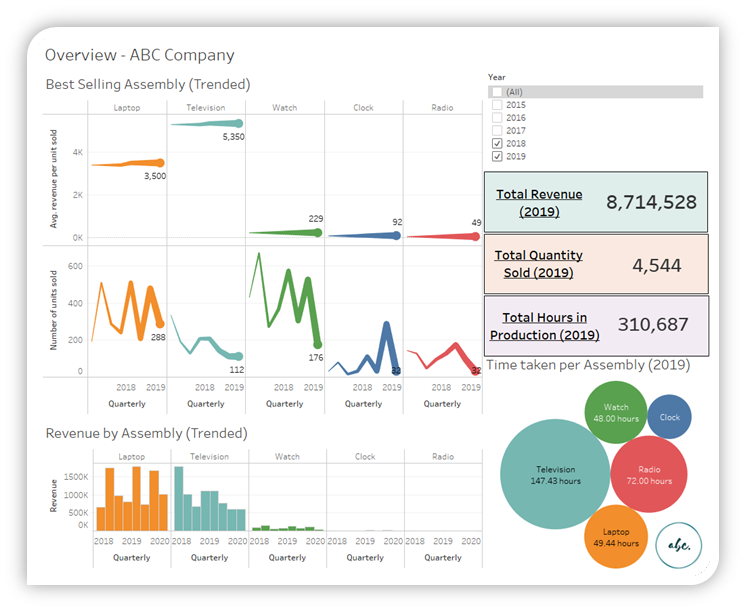
|  |
| --- |
| INSERT INTO fact\_sales (assembly\_no,price,date\_key)  SELECT wo.Assembly\_No,i.Price\_per\_WO,date(sw.end\_time)  FROM invoice i  JOIN work\_order wo ON wo.wo\_no=i.wo\_no  JOIN order\_assembly oa ON oa.order\_no=wo.order\_no  JOIN stations\_wo sw ON sw.wo\_no=wo.wo\_no; |

The picture below is a snapshot of the “**Fact\_Sales**” fact table.



# **Business Intelligence Dashboard Design**

Two Tableau dashboards (See ***Figures 7.1*** and ***7.2*** below) were created to meet the OLAP business requirements, by connecting Tableau to the MySQL data warehouse.



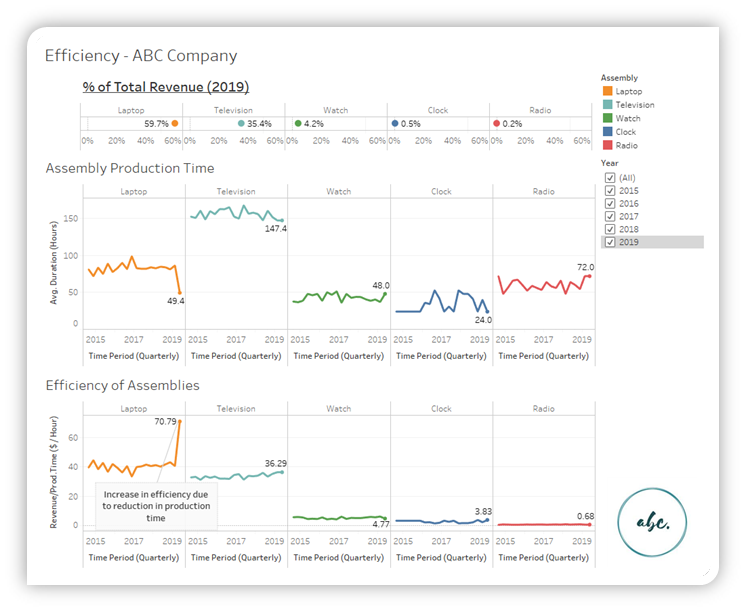
***Figure 7.1 - Dashboard - Overview - ABC Company***

Firstly, an overview dashboard shown in ***Figure 7.1*** shows an overview of the key statistics which the management team is interested in tracking.

In this overview dashboard, key trended (quarterly) statistics by assembly are tracked, with the flexibility of filtering relevant years. This includes the average price per unit sold, number of units sold and total revenue across these quarters. Also, a snapshot of 2019 top-line figures are displayed, such as the total revenue, quantity sold and production hours.

The takeaway would be that we need to streamline efforts in pushing our television products, as there is a downward trend in the number of units sold and the associated revenue figures. Further investigation should be done to ascertain the reasons which may be due to intensified competition recently. Laptop sales had continued their seasonal trend, but watches, clocks and radio sales remained stagnant. It may be worthwhile to divert the Company limited resources to better strengthen its core product propositions around higher-tech products like laptops and televisions.

**7. Business Intelligence Dashboard Design (Cont’d)**



***Figure 7.2 - Dashboard - Efficiency - ABC Company***

***Figure 7.2*** shows the second dashboard, which outlines the efficiency metrics in a trended view as well, sorted by percentage of total revenue in 2019. It has highlighted the benefits of leasing more efficient machinery for laptop assembly in the recent quarter, where the average duration to assemble a laptop reduced from an average of around 86.4 hours to only 49.4 hours (-42.8% in time required). This resulted in the revenue per production hour for laptops to reach an all-time high level of $70.79 for this quarter’s laptop performance.

**7. Business Intelligence Dashboard Design (Cont’d)**

These dashboards will allow the following OLAP-related business requirements (Refer to ***Section 2.2: Requirements for OLAP***) to be met:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requirements for OLAP** | **Priority** | **How did Our Design Address the Requirements?** |
| 1 | To provide an OLAP solution to facilitate analytics for sales, production and supply chain departments. | High | Our BI and OLAP could give a high-level overview: At both a revenue and efficiency standpoint |
| 2 | Revenue statistics to be tracked at an assembly-level | High | Shown in ***Figure 7.1*** |
| 3 | Revenue breakdown into number of units sold and average revenue of product to be tracked | High | Shown in ***Figure 7.1*** |
| 4 | Assembly time (trended) is tracked for improvement potential. This include determining the assembly that takes the shortest time to produce. | High | ***Figure 7.2*** has shown that laptop production efficiency has improved due to reduction in production time, realising the benefits of leasing higher-tech machinery. Time taken to produce laptops have fallen below the level required for radios in the recent quarter. |
| 5 | Revenue per assembly time ($/hour) trend to be tracked to determine revenue efficiency of production. | High | ***Figure 7.2*** shows that it has been much more efficient to produce laptops in the recent quarter than previous years. |
| 6 | Revenue breakdown by assembly | Medium | ***Figure 7.2*** shows that revenue from the laptop vertical makes up 59.7% of ABC company’s total revenue, followed by television at 35.4%. |

# ***Appendix A: Data Generation Scripts***

The following python scripts were used to generate the data in the OLTP database:

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

############################################ Supplier ############################################

from faker import Faker

from faker\_e164.providers import E164Provider

import pandas as pd

numberOfRows = 20

custID = []

for i in range(numberOfRows):

if i <= 9:

customer = 's' + '0' + '0' + '0' + str(i)

if i <= 99:

customer = 's' + '0' + '0'+ str(i)

else:

customer = 's' + '0' + str(i)

custID.append(customer)

fake = Faker()

name = [fake.company() for i in range(numberOfRows)]

address = [fake.address() for i in range(numberOfRows)]

fake.add\_provider(E164Provider) # Get phone number from E164Provider library

contactNo = [fake.e164(region\_code="GB") for i in range(numberOfRows)]

data = {'Supp\_ID': custID, 'Name': name, 'Address': address, 'Contact\_No': contactNo}

df = pd.DataFrame(data)

df = df[df != df.duplicated(['Name'])]

df.dropna()

df.reset\_index()

df.to\_csv("Supplier.csv", index=False)

############################################ Supplier\_Part ############################################

# Cross product suppid with partid and drop 3/4 of the combinations

# Create price value by drawing samples from normal distribution with specified mean and standard deviation values

import pandas as pd

from itertools import product

import random

import numpy as np

random.seed(549)

dfsupplier = pd.read\_csv("Supplier.csv")

dfparts = pd.read\_csv("Parts.csv")

suppid = dfsupplier["Supp\_ID"].values.tolist()

partno = dfparts["Part\_No"].values.tolist()

combinelist = list(sorted(product(suppid, partno)))

randomList = []

numbersOfRow = int(len(combinelist)/4)

# print(numbersOfRow)

for i in range(numbersOfRow):

randomchoice = random.choice(combinelist)

randomList.append(randomchoice)

randomList = sorted(randomList)

suppidList = []

partnoList = []

for i in randomList:

suppidList.append(i[0])

partnoList.append(i[1])

pkey = list(range(1, numbersOfRow+1))

data = {'Pkey': pkey, 'Supp\_ID': suppidList, 'Part\_No': partnoList}

df = pd.DataFrame(data)

# print(df)

meanPriceDict = {"p000": 1000, "p001": 5, "p002": 6, "p003": 4, "p004": 10, "p005": 3, "p006": 2, "p007": 20,

"p008": 100, "p009": 10, "p0010": 300, "p0011": 20, "p0012": 8, "p0013": 230, "p0014": 18, "p0015": 80}

stddecPriceDict = {"p000": 100, "p001": 3, "p002": 2, "p003": 5, "p004": 5, "p005": 1, "p006": 1, "p007": 3,

"p008": 20, "p009": 2, "p0010": 40, "p0011": 3, "p0012": 2, "p0013": 50, "p0014": 3, "p0015": 16}

partkeys = list(meanPriceDict.keys())

testno = np.random.normal(meanPriceDict["p003"], stddecPriceDict["p003"], 1)

for index, row in df.iterrows():

for i in partkeys:

if df.at[index, "Part\_No"] == i:

newPrice= np.random.normal(meanPriceDict[i], stddecPriceDict[i], 1)

if newPrice <= 0:

newPrice = 1

df.at[index, "Price"] = newPrice

else:

df.at[index, "Price"] = newPrice

df["Price"] = df["Price"].round(0).astype(int)

df = df.drop\_duplicates(subset=['Supp\_ID', 'Part\_No'], keep='first')

df.reset\_index()

noOfRows = len(df.index) + 1

df["Pkey"] = [i for i in range(1, noOfRows)] # Renumber PKey column

df.to\_csv("Supplier\_Part.csv", index=False)

############################################ Parts ############################################

import pandas as pd

nameRadio = ["battery", "speaker", "controller", "cable", "antenna", "housing"]

nameTV = ["picture tube", "controller", "led", "cable", "housing"]

nameWatch = ["battery", "beeper", "strap", "buckle", "glass", "housing"]

nameClock = ["battery", "beeper", "screws", "glass", "housing"]

nameLaptop = ["speaker", "cable", "bag", "camera", "case", "housing"]

nameList = [nameRadio, nameTV, nameWatch, nameClock, nameLaptop]

name = []

for i in range(len(nameList)):

name.append(nameList[i])

nameFlat = [i for sublist in name for i in sublist]

nameFlat = list(set(nameFlat))

numberOfRows = len(nameFlat)

partID = []

for i in range(numberOfRows):

if i <= 9:

part = 's' + '0' + '0' + '0' + str(i)

if i <= 99:

part = 's' + '0' + '0'+ str(i)

else:

part = 's' + '0' + str(i)

partID.append(part)

quantity = [1000000, 5000, 50000, 100000, 10000, 3000, 200000, 35000, 50000, 78000, 35000, 40000, 60000, 50000, 3000, 3000]

data = {'Part\_No': partID, 'Name': nameFlat, 'Quantity\_Available': quantity}

df = pd.DataFrame(data)

df.to\_csv("Parts.csv", index=False)

############################################ Work\_Order ############################################

import pandas as pd

dfOrderAssembly = pd.read\_csv("Order\_Assembly.csv")

workorderno = []

for i in range(len(dfOrderAssembly.index)):

if i <= 9:

part = 'workorder' + '0' + '0' + '0' + str(i)

if i <= 99:

part = 'workorder' + '0' + '0'+ str(i)

else:

part = 'workorder' + '0' + str(i)

workorderno.append(part)

dfOrderAssembly = dfOrderAssembly.rename(columns={"Pkey": "WO\_No", "Order\_No": "Order\_No",

"Assembly\_No": "Assembly\_No", "Quantity": "Status"})

dfOrderAssembly["WO\_No"] = workorderno

dfOrderAssembly["Status"] = ["Close" for i in range(len(dfOrderAssembly.index))]

# Set last 10 orders as Open (Chronological order)

for index, row in dfOrderAssembly.iterrows():

if dfOrderAssembly.at[index, "Order\_No"] in ["order0590", "order0591", "order0592", "order0593", "order0594",

"order0595", "order0596", "order0597", "order0598", "order0599"]:

dfOrderAssembly.at[index, "Status"] = "Open"

dfOrderAssembly.to\_csv("Work\_Order.csv", index=False)

############################################ Stations\_WO ############################################

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

from random import choices

dfworkorder = pd.read\_csv("Work\_Order.csv")

dfworkorder = dfworkorder.rename(columns={"WO\_No": "WO\_No", "Order\_No": "Order\_No",

"Assembly\_No": "Station\_ID", "Status": "Status"})

dfcustomerorder = pd.read\_csv("Customer\_Order.csv")

mergeDF = dfworkorder.merge(dfcustomerorder, left\_on="Order\_No", right\_on="Order\_No")

mergeDF["Order\_DateTime"] = pd.to\_datetime(mergeDF["Order\_DateTime"])

dayFromOrderToAssy = [1, 2, 3]

probability\_dayFromOrderToAssy = [5, 3, 1]

for index, row in mergeDF.iterrows():

randomDay = choices(dayFromOrderToAssy, probability\_dayFromOrderToAssy, k=1)[0]

mergeDF.at[index, "Start\_Time"] = mergeDF.at[index, "Order\_DateTime"] + timedelta(days=randomDay)

# For a001 (Radio):

dayFromStartToEnd\_radio = [2, 3, 4]

probability\_dayFromStartToEnd\_radio = [8, 3, 1]

# For a002 (TV):

dayFromStartToEnd\_tv = [5, 6, 7]

probability\_dayFromStartToEnd\_tv = [2, 6, 9]

# For a003 (Watch):

dayFromStartToEnd\_watch = [1, 2, 3]

probability\_dayFromStartToEnd\_watch = [5, 8, 2]

# For a004 (clock):

dayFromStartToEnd\_clock = [1, 2, 3]

probability\_dayFromStartToEnd\_clock = [7, 3, 1]

# For a005 (laptop):

dayFromStartToEnd\_laptop = [3, 4, 5]

probability\_dayFromStartToEnd\_laptop = [7, 4, 1]

for index, row in mergeDF.iterrows():

if mergeDF.at[index, "Station\_ID"] == "a001":

randomDay = choices(dayFromStartToEnd\_radio, probability\_dayFromStartToEnd\_radio, k=1)[0]

mergeDF.at[index, "End\_Time"] = mergeDF.at[index, "Start\_Time"] + timedelta(days=randomDay)

elif mergeDF.at[index, "Station\_ID"] == "a002":

randomDay = choices(dayFromStartToEnd\_tv, probability\_dayFromStartToEnd\_tv, k=1)[0]

mergeDF.at[index, "End\_Time"] = mergeDF.at[index, "Start\_Time"] + timedelta(days=randomDay)

elif mergeDF.at[index, "Station\_ID"] == "a003":

randomDay = choices(dayFromStartToEnd\_watch, probability\_dayFromStartToEnd\_watch, k=1)[0]

mergeDF.at[index, "End\_Time"] = mergeDF.at[index, "Start\_Time"] + timedelta(days=randomDay)

elif mergeDF.at[index, "Station\_ID"] == "a004":

randomDay = choices(dayFromStartToEnd\_clock, probability\_dayFromStartToEnd\_clock, k=1)[0]

mergeDF.at[index, "End\_Time"] = mergeDF.at[index, "Start\_Time"] + timedelta(days=randomDay)

else:

randomDay = choices(dayFromStartToEnd\_laptop, probability\_dayFromStartToEnd\_laptop, k=1)[0]

mergeDF.at[index, "End\_Time"] = mergeDF.at[index, "Start\_Time"] + timedelta(days=randomDay)

# Change to Station\_ID

for index, row in mergeDF.iterrows():

if mergeDF.at[index, "Station\_ID"] == "a001":

mergeDF.at[index, "Station\_ID"] = "s01"

elif mergeDF.at[index, "Station\_ID"] == "a002":

mergeDF.at[index, "Station\_ID"] = "s02"

elif mergeDF.at[index, "Station\_ID"] == "a003":

mergeDF.at[index, "Station\_ID"] = "s03"

elif mergeDF.at[index, "Station\_ID"] == "a004":

mergeDF.at[index, "Station\_ID"] = "s04"

elif mergeDF.at[index, "Station\_ID"] == "a005":

mergeDF.at[index, "Station\_ID"] = "s05"

mergeDF = mergeDF.sort\_values(by=["Start\_Time"])

# If status == "Open", End\_Time = NaN

for index, row in mergeDF.iterrows():

if mergeDF.at[index, "Status"] == "Open":

mergeDF.at[index, "End\_Time"] = ""

mergeDF = mergeDF.drop(["Order\_No", "Status", "Cust\_ID", "Order\_DateTime"], axis=1)

pkey = [i for i in range(1, len(mergeDF.index)+1)]

mergeDF.insert(loc=0, column="Pkey", value=pkey)

mergeDF.to\_csv("Stations\_WO.csv", index=False)

############################################ Invoice ############################################

import pandas as pd

import numpy as np

dfworkorder = pd.read\_csv("Work\_Order.csv")

dfcustomerorder = pd.read\_csv("Customer\_Order.csv")

dfmerge = dfworkorder.merge(dfcustomerorder, left\_on="Order\_No", right\_on="Order\_No")

print(dfmerge.tail())

## Set price for each item

# a001 (radio), a002 (tv), a003 (watch), a004 (clock), a005 (laptop)

# Assume selling price increases over years

pricing\_2015 = {"a001": 39, "a002": 5000, "a003": 215, "a004": 79, "a005": 3200}

pricing\_2016 = {"a001": 40, "a002": 5200, "a003": 216, "a004": 79, "a005": 3250}

pricing\_2017 = {"a001": 42, "a002": 5250, "a003": 219, "a004": 80, "a005": 3300}

pricing\_2018 = {"a001": 45, "a002": 5299, "a003": 225, "a004": 82, "a005": 3400}

pricing\_2019 = {"a001": 49, "a002": 5350, "a003": 229, "a004": 92, "a005": 3500}

dfmerge["Order\_DateTime"] = pd.to\_datetime(dfmerge["Order\_DateTime"]) # Convert to pandas datetime format for .year attribute

print(type(dfmerge["Order\_DateTime"][0]))

for index, row in dfmerge.iterrows():

# if dfmerge.at[index, "Status"] == "Close":

if dfmerge.at[index, "Order\_DateTime"].year == 2015:

if dfmerge.at[index, "Assembly\_No"] == "a001":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2015["a001"]

elif dfmerge.at[index, "Assembly\_No"] == "a002":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2015["a002"]

elif dfmerge.at[index, "Assembly\_No"] == "a003":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2015["a003"]

elif dfmerge.at[index, "Assembly\_No"] == "a004":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2015["a004"]

elif dfmerge.at[index, "Assembly\_No"] == "a005":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2015["a005"]

elif dfmerge.at[index, "Order\_DateTime"].year == 2016:

if dfmerge.at[index, "Assembly\_No"] == "a001":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2016["a001"]

elif dfmerge.at[index, "Assembly\_No"] == "a002":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2016["a002"]

elif dfmerge.at[index, "Assembly\_No"] == "a003":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2016["a003"]

elif dfmerge.at[index, "Assembly\_No"] == "a004":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2016["a004"]

elif dfmerge.at[index, "Assembly\_No"] == "a005":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2016["a005"]

elif dfmerge.at[index, "Order\_DateTime"].year == 2017:

if dfmerge.at[index, "Assembly\_No"] == "a001":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2017["a001"]

elif dfmerge.at[index, "Assembly\_No"] == "a002":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2017["a002"]

elif dfmerge.at[index, "Assembly\_No"] == "a003":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2017["a003"]

elif dfmerge.at[index, "Assembly\_No"] == "a004":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2017["a004"]

elif dfmerge.at[index, "Assembly\_No"] == "a005":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2017["a005"]

elif dfmerge.at[index, "Order\_DateTime"].year == 2018:

if dfmerge.at[index, "Assembly\_No"] == "a001":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2018["a001"]

elif dfmerge.at[index, "Assembly\_No"] == "a002":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2018["a002"]

elif dfmerge.at[index, "Assembly\_No"] == "a003":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2018["a003"]

elif dfmerge.at[index, "Assembly\_No"] == "a004":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2018["a004"]

elif dfmerge.at[index, "Assembly\_No"] == "a005":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2018["a005"]

elif dfmerge.at[index, "Order\_DateTime"].year == 2019:

if dfmerge.at[index, "Assembly\_No"] == "a001":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2019["a001"]

elif dfmerge.at[index, "Assembly\_No"] == "a002":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2019["a002"]

elif dfmerge.at[index, "Assembly\_No"] == "a003":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2019["a003"]

elif dfmerge.at[index, "Assembly\_No"] == "a004":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2019["a004"]

elif dfmerge.at[index, "Assembly\_No"] == "a005":

dfmerge.at[index, "Price\_per\_WO"] = pricing\_2019["a005"]

# else:

# To be used if workorder is inserted into invoice table only when Work\_Order.Status is "Close"

# Meaning, if assembly is not completed, the item will not appear in invoice table

# dfmerge.at[index, "Price\_per\_WO"] = np.nan

dfmerge = dfmerge.drop(["Order\_No", "Assembly\_No", "Status", "Order\_DateTime"], axis=1)

print(dfmerge.head())

invoiceNoList = []

for i in range(len(dfmerge.index)):

if i <= 9:

invoiceNo = 'invoice' + '0' + '0' + '0' + str(i)

if i <= 99:

invoiceNo = 'invoice' + '0' + '0'+ str(i)

else:

invoiceNo = 'invoice' + '0' + str(i)

invoiceNoList.append(invoiceNo)

pkey = [i for i in range(1, len(dfmerge.index)+1)]

dfmerge["Invoice\_No"] = invoiceNoList

dfmerge["Pkey"] = pkey

dfmerge = dfmerge[["Pkey", "Invoice\_No", "Cust\_ID", "WO\_No", "Price\_per\_WO"]]

dfmerge.to\_csv("Invoice.csv", index=False)

############################################ Order\_Assembly ############################################

import pandas as pd

from random import choices

dfcustorder = pd.read\_csv("Customer\_Order.csv")

dfassy = pd.read\_csv("assemblies.csv")

orderno = dfcustorder["Order\_No"].values.tolist()

dfassy = dfassy ["Assembly\_No"].values.tolist()

def selectRandomAssyFromList(assy, order):

assylist = []

for i in order:

# Customer preference of items: a003 (watch) > a005 (laptop) > a002 (tv) > a001 (radio) > a004 (clock)

assylist.append(choices(assy, weights=[3, 5, 10, 1, 8], k=1))

flatList = [item for sublist in assylist for item in sublist]

return flatList

def selectRandomOrderFromList(order, fractionOfCurrent=5):

orderlist = []

for i in order:

orderlist.append(choices(order, k=int(len(order)/fractionOfCurrent)))

return orderlist[0]

orderlist\_1 = orderno

assylist\_1 = selectRandomAssyFromList(dfassy, orderlist\_1)

orderlist\_2 = selectRandomOrderFromList(orderlist\_1)

assylist\_2 = selectRandomAssyFromList(dfassy, orderlist\_2)

orderlist\_3 = selectRandomOrderFromList(orderlist\_1)

assylist\_3 = selectRandomAssyFromList(dfassy, orderlist\_3)

fullOrderList = orderlist\_1

fullAssyList = assylist\_1

fullOrderList.extend(orderlist\_2)

fullAssyList.extend(assylist\_2)

fullOrderList.extend(orderlist\_3)

fullAssyList.extend(assylist\_3)

pkey = [i for i in range(1, len(fullOrderList) + 1)]

data = {"Pkey": pkey, "Order\_No": fullOrderList, "Assembly\_No": fullAssyList}

df = pd.DataFrame(data)

for index, row in df.iterrows():

if df.at[index, "Assembly\_No"] == "a001":

# Radio

moq = [2, 3, 4, 5, 6]

prob = [1, 2, 5, 2, 1]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

elif df.at[index, "Assembly\_No"] == "a002":

# TV

moq = [1, 2, 3]

prob = [5, 3, 1]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

elif df.at[index, "Assembly\_No"] == "a003":

# Watch

moq = [6, 7, 8, 9, 10]

prob = [4, 5, 6, 7, 10]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

elif df.at[index, "Assembly\_No"] == "a004":

# clock

moq = [6, 7, 8, 9, 10]

prob = [4, 5, 6, 7, 10]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

else:

# laptop

moq = [1, 2, 3]

prob = [5, 3, 1]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

df.reset\_index()

df["Quantity"] = df["Quantity"].round(0).astype(int)

df = df.drop\_duplicates(subset=['Order\_No', 'Assembly\_No'], keep='first')

df.to\_csv("Order\_Assembly.csv", index=False)

############################################ Customer\_Orders ############################################

import pandas as pd

from random import choices

dfcust = pd.read\_csv("Customers.csv")

custid = dfcust["Cust\_ID"].values.tolist()

percentOfReturnCustomer = 20

randomSample = int((percentOfReturnCustomer/100)\*len(custid))

custidExtra = choices(custid, k=randomSample) # random with replacement

custid.extend(custidExtra)

# https://gist.github.com/rg3915/db907d7455a4949dbe69 (random DATETIME generator)

import random

from datetime import datetime, timedelta

def gen\_datetime(min\_year=2015, max\_year=datetime.now().year):

# generate a datetime in format yyyy-mm-dd hh:mm:ss.000000

start = datetime(min\_year, 1, 1, 00, 00, 00)

years = max\_year - min\_year + 1

end = start + timedelta(days=365 \* years)

output = start + (end - start) \* random.random()

output = output.strftime("%Y-%m-%d %H:%M:%S") # Modified to drop milliseconds

return output

noOfRows = len(custid)

orderDateTime = sorted([gen\_datetime() for i in range(noOfRows)])

orderNo = []

for i in range(noOfRows):

if i <= 9:

order = 'order' + '0' + '0' + '0' + str(i)

if i <= 99:

order = 'order' + '0' + '0'+ str(i)

else:

order = 'order' + '0' + str(i)

orderNo.append(order)

data = {'Order\_No': orderNo, 'Cust\_ID': custid, 'Order\_DateTime': orderDateTime}

df = pd.DataFrame(data)

df.to\_csv("Customer\_Order.csv", index=False)

############################################ Order\_Assembly ############################################

import pandas as pd

from random import choices

dfcustorder = pd.read\_csv("Customer\_Order.csv")

dfassy = pd.read\_csv("assemblies.csv")

orderno = dfcustorder["Order\_No"].values.tolist()

dfassy = dfassy ["Assembly\_No"].values.tolist()

def selectRandomAssyFromList(assy, order):

assylist = []

for i in order:

# Customer preference of items: a003 (watch) > a005 (laptop) > a002 (tv) > a001 (radio) > a004 (clock)

assylist.append(choices(assy, weights=[3, 5, 10, 1, 8], k=1))

flatList = [item for sublist in assylist for item in sublist]

return flatList

def selectRandomOrderFromList(order, fractionOfCurrent=5):

orderlist = []

for i in order:

orderlist.append(choices(order, k=int(len(order)/fractionOfCurrent)))

return orderlist[0]

orderlist\_1 = orderno

assylist\_1 = selectRandomAssyFromList(dfassy, orderlist\_1)

orderlist\_2 = selectRandomOrderFromList(orderlist\_1)

assylist\_2 = selectRandomAssyFromList(dfassy, orderlist\_2)

orderlist\_3 = selectRandomOrderFromList(orderlist\_1)

assylist\_3 = selectRandomAssyFromList(dfassy, orderlist\_3)

fullOrderList = orderlist\_1

fullAssyList = assylist\_1

fullOrderList.extend(orderlist\_2)

fullAssyList.extend(assylist\_2)

fullOrderList.extend(orderlist\_3)

fullAssyList.extend(assylist\_3)

pkey = [i for i in range(1, len(fullOrderList) + 1)]

data = {"Pkey": pkey, "Order\_No": fullOrderList, "Assembly\_No": fullAssyList}

df = pd.DataFrame(data)

for index, row in df.iterrows():

if df.at[index, "Assembly\_No"] == "a001":

# Radio

moq = [2, 3, 4, 5, 6]

prob = [1, 2, 5, 2, 1]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

elif df.at[index, "Assembly\_No"] == "a002":

# TV

moq = [1, 2, 3]

prob = [5, 3, 1]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

elif df.at[index, "Assembly\_No"] == "a003":

# Watch

moq = [6, 7, 8, 9, 10]

prob = [4, 5, 6, 7, 10]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

elif df.at[index, "Assembly\_No"] == "a004":

# clock

moq = [6, 7, 8, 9, 10]

prob = [4, 5, 6, 7, 10]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

else:

# laptop

moq = [1, 2, 3]

prob = [5, 3, 1]

choice = choices(moq, weights=prob, k=1)[0]

df.at[index, "Quantity"] = choice

df.reset\_index()

df["Quantity"] = df["Quantity"].round(0).astype(int)

df = df.drop\_duplicates(subset=['Order\_No', 'Assembly\_No'], keep='first')

df.to\_csv("Order\_Assembly.csv", index=False)

############################################ End ############################################

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **[WO\_No]** and **[Assembly\_No]** are fields that can be found in “**WO\_Order**” table under ***Section 3.2.6***. [↑](#footnote-ref-1)