

THE CHINESE UNIVERSITY OF HONG KONG, SHENZHEN

CSC 3170

---

# Project Report

---

## Team Members

郭宇轩 118020017

郭俊杰 118010081

罗皓严 119010221

林铠越 119010176

方文欣 119010068

May 1<sup>st</sup>, 2021

## 1. Background & Introduction

The database system is the most active, most extensive, and largest application field in the computer application field. The database system can calculate, manage and process any form of data, such as the management of production enterprises, the calculation of accounts, and the retrieval of books and materials. The database system is developed with the needs of information processing and transaction processing of enterprises. Every department has to master and manage a large amount of data. If we manually check, modify, delete and update the data every day, it will be a waste of time and resources. However, if the enterprise is equipped with a database system, it can promote the automation of enterprise management.

Our database system is designed for a company that contracts Apple's production business. Employees of the company come from all over the world, so we specially add the attribute of skin color to the employees, such as yellow skin, white skin and so on. Besides, this company has different workshops and each workshop has its own manager and employee. Each workshop is also responsible for the production of one kind of component so that each workshop can produce different components independently and efficiently. Related data such as the name and type of each component are also recorded in the database. At the same time, we also recorded the warehouse data of the factory, including warehouse name and inventory. What's more, our database also contains the information of order and the buyer. For example, they recorded the total amount of each order or name of the buyer, thus facilitating the statistics of this company. When the data in this company reaches a certain amount, it will become very inefficient or even impossible to complete if it still relies on employees to process the data manually. Therefore, we have designed a database for such a company and put a large amount of data in the data table, so as to store the factory data conveniently and effectively. Further, it can improve the work efficiency and management level of the company. Besides, it can also avoid duplication and redundant work. Then employees can share the company database immediately. Then, by using the query, update, and even statistical tools provided by the database, these data can be used quickly and fully.

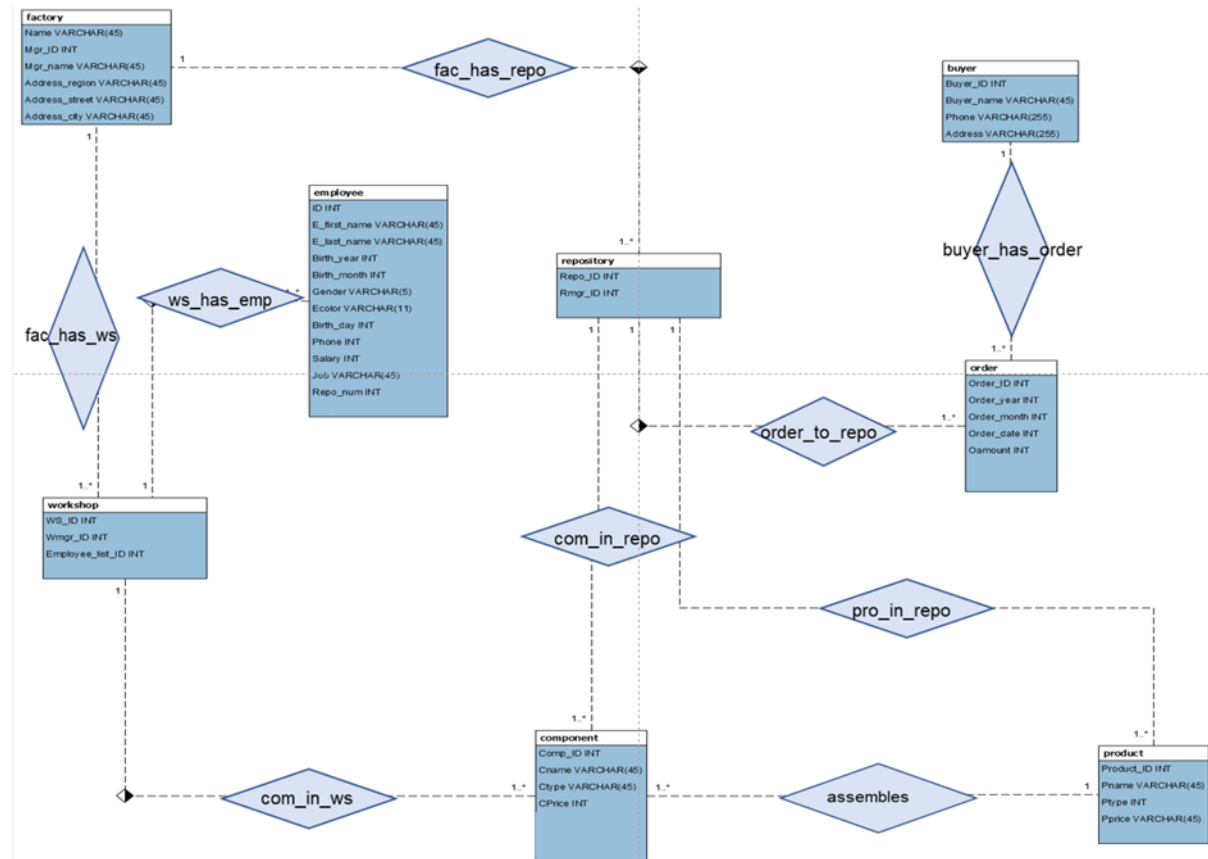
To design such a database system, we have designed an E-R diagram with eight entities as we mentioned above. Besides, there are ten relationships within these entities. For example, there is a one-to-many relationship between "factory" and "workshop", named "fac\_has\_ws". Similarly, all other relations are one-to-many relations. Besides, these relations are described by adding foreign keys to the "many-side" entities' schema to make the relation schema less redundant. On the other hand, the ER diagram was reduced into relation schema to help us develop the tables and dependencies and they are in Boyce Codd Normal Form. Then we generated the data for basic information of a factory, human resources information, and sales information. The generated data were imported into MySQL workbench. The workbench will generate queries automatically when we import data as a new table or do other operations. As for the UI design, there are two versions in total. The first version is implemented with Vue and Express, which is a typical version of the company database control panel. Another version is relatively simpler and like a normal website, which is a more straightforward and more accessible interface. Furthermore, we used the database to visualize some cases of data mining and data analysis, including univariate analysis and multivariate analysis.

## 2. Design

In this section, we will focus on illustrating designing the Entity-Relationship Model, reducing ER diagram into relational schemas, and deciding which entities need to be indexed.

### 2.1. The Entity-Relationship Model

As shown in the E-R Diagram, there are 8 entities and 10 relationships.



The entity named “factory” is used to store factory names, factory managers’ IDs, and the factories’ addresses.

The entity named “employee” is used to store necessary personal information of employees, including their unique IDs (hence ID is also the primary key of this entity), names, birth dates, gender (valid values of this attribute is only male and female), phone number, salary, and two other attributes, Repo\_num and WS\_ID. Repo\_ID, which is used to specify whether an employee is responsible for a certain warehouse, uses a method that if an employee is responsible for one warehouse, then Repo\_ID is 0, otherwise, it is a nonzero numeric value. There should be an attribute used to specify which workshop an employee works in, so it is a foreign key referencing the entity “workshop”, with the domain being that of the entity “workshop”, so WS\_ID is added (see below).

The entity named “workshop” is used to store necessary information of workshops, so as to make managing workshops more convenient. The attribute WS\_ID is the primary key of this entity. Wmgr\_ID means the manager’s ID of the workshop. To specify which factory this

workshop belongs to, a foreign key “Factory\_name” was added.

The entity named “repository” is used to store the necessary information of warehouse repositories. The attribute Repo\_ID is the primary key of this entity. The attribute Rmgr\_ID is used to specify the manager’s ID of a repository, so it is not a primary key since a particular employee can be responsible for more than 1 repository.

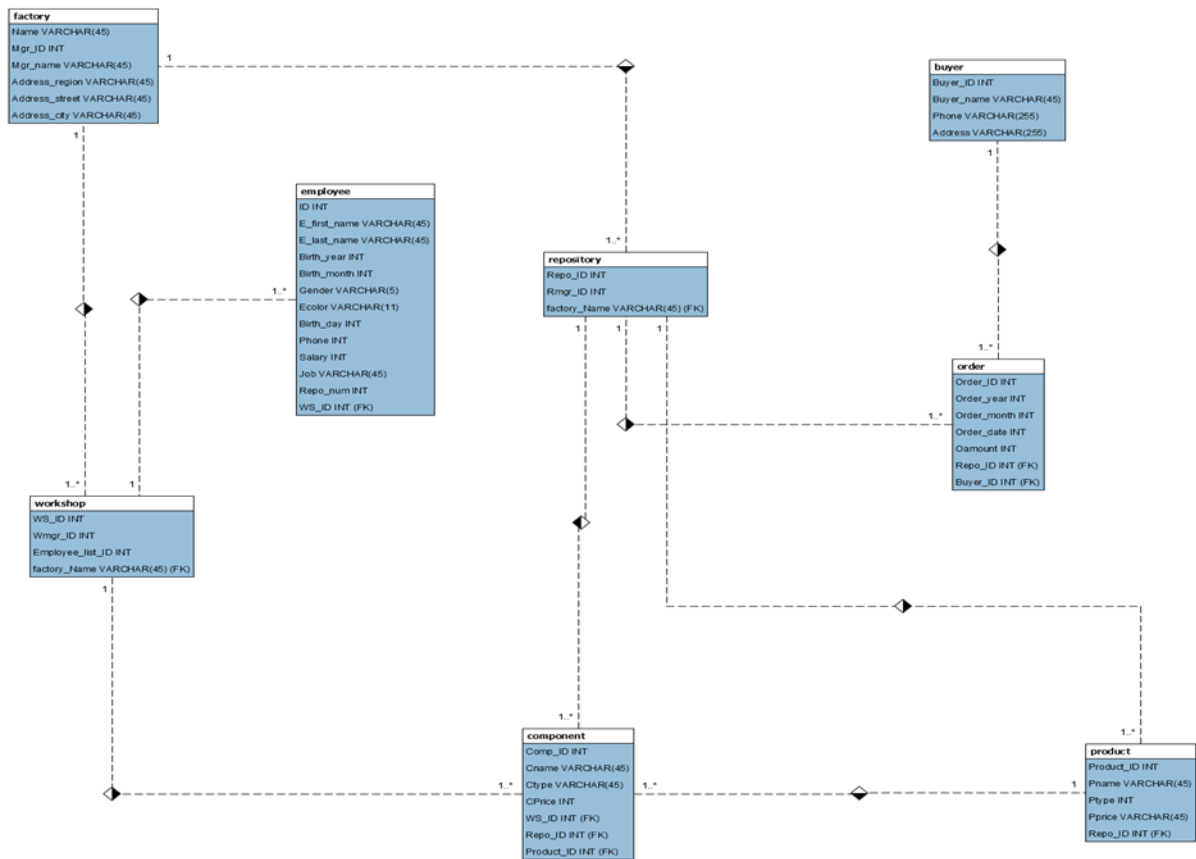
Since a factory owns many orders, this E-R Diagram also includes an entity named “order”, with the attribute Order\_ID being its primary key. This entity also has attributes to store the order dates (Order\_year, Order\_month, and Order\_day) and the amount of money involved (Oamount). Buyer\_ID is a foreign key, referencing entity “buyer”. For a factory, since it does not have orders as frequently as once in several seconds, it is almost impossible for any two orders to have exactly the same date and time (accurate to seconds). Hence, when an order is sent to this database, the date and the time are recorded, and the combination of them (e.g., 2021-0430-1200) can be used as the primary key, even though time is not recorded in the tuples.

Entity “buyer” is used to store the information of buyers. Information including buyers’ IDs, names, phone numbers, addresses, and the orders they own.

Since this factory function is mainly to manufacture mobile phone components to assemble phones, it is required to design an entity “component” to store basic necessary information of components. Each component tuple has a unique Comp\_ID value, which is the primary key of this entity. Besides, this entity also has Cname to store the name of a component, Ctype to store the category that it belongs to, Cprice to store its price. To describe the entity’s relationships, it should have three more foreign keys, WS\_ID to specify which workshop it will be sent to assemble a phone, Repo\_ID to specify which repository it is located, and Product\_ID to specify which product it is assembled into (if the component is processed).

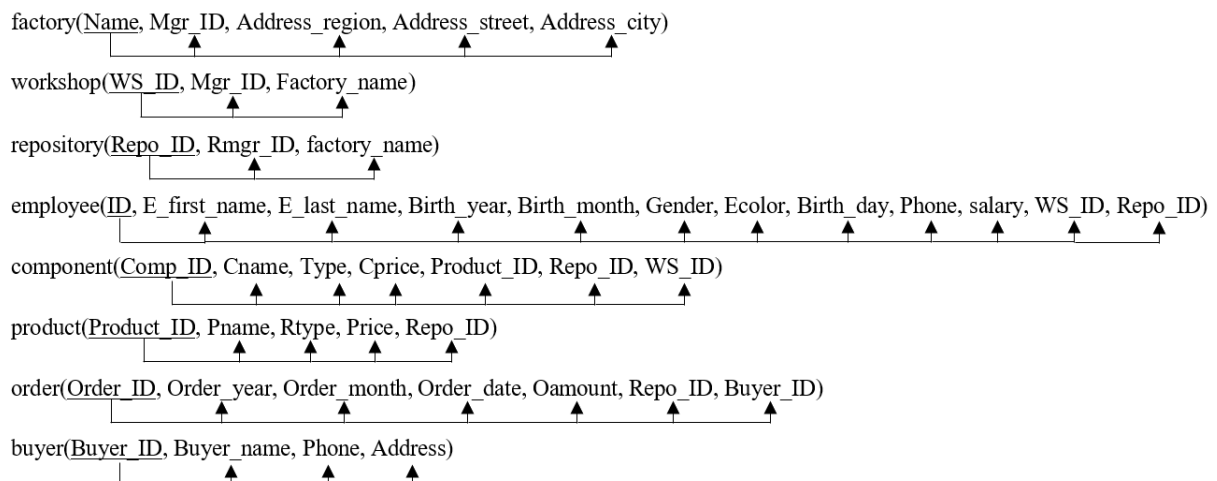
With the same reason stated in the last paragraph, entity “product” is also needed. It has Product\_ID as the primary key, Pname to store a product’s names, Ptype to store the category it belongs to, Pprice to store its price. Foreign key Repo\_ID should be added to specify which repository it is located.

In the graph shown above, there are several relations. Since a factory has more than one workshop, there is a one-to-many relationship between “factory” and “workshop”, named “fac\_has\_ws”, to describe this relation. Similarly, all other relations are one-to-many relations. To make the relation schema less redundant, these relations do not appear in the schema. Instead, they are described by adding foreign keys to the “many-side” entities’ schema. After that, the new E-R Diagram looks like the following graph.



## 2.2. The Relational Schemas

We need to reduce the ER diagram into relation schema to help us develop the tables and check the dependencies. We give each entity (e.g., person, product, etc.) a unique ID to keep them organized and clear. After checking the dependencies between attributes one by one, we can obtain the following figure, which shows that the relational schemas are in the Boyce Codd Normal Form.



## 2.3. Index and Hashing

As for a database for a factory, the repository, component and product tables are changing according to the producing process. They are not necessary to be indexed or hashed since

indexing shortens the search time, but it slows the updating because changes in the node may cause re-indexing. For table employees, it will gain great efficiency if we do the indexing. The employee information will not be changed frequently and they are certainly retrieved rather frequently. Employee tables are thought to change once or twice a year. Using hashing can reduce the searching time from  $O(n)$  to  $O(\lg(n))$ .

### 3. Implementations

In order to bring our design into reality, we took 3 steps:

1. Generating data according to our design.
2. Establishing the database system and importing the data.
3. Designing a friendly user interface.

In this section, we will illustrate how each step is completed.

#### 3.1. Data Generation

After designing the structure of the database, we implement the design by generating data first. To derive appropriate datasets for a production & sales factory database, we generate the data in three main sections: basic information of a factory, human resources information, and sales information.

First, according to our design, the basic structure includes the factory itself, workshops, and repositories. For the general factory information, we choose Foxcomm (a pseudo version of Foxconn) to be our first factory customer. Since the factory is a contractor which produces digital products for a technology company like Apple, its workshop produces different components like the white shell of iPhone 11 and stores them in corresponding repositories. Therefore, the data of workshops and repositories contain IDs and their factory name (foreign key constraints). These data are intendedly designed to ensure they are correlated (see Appendix 8.1).

Second, we collect most of the human resources data from the Kaggle website. As shown in Appendix 8.2 and Appendix 8.3, these datasets from the website have been verified and many people use them from time to time; therefore, the correctness and feasibility of the factory information can be guaranteed. These datasets are modified and combined into the information of one entity named “employee”. Users can query an employee’s gender, age, job, phone number, and salary.

Last, we assume that there exist several companies that have ordered products from the factory; therefore, we need data for order lists and buyers. In the order table, we collect information of the order’s ID (primary key), order’s date, the total amount of the order, and its Repo\_ID (i.e., the repository’s ID that the ordered) products/components belong to (foreign key). We used Python to generate data, in which four libraries “xlwt”, “random”, “time” and “pandas” are referenced to achieve the corresponding requirements.

As for the “buyer” entity, we have generated four sets of data: “Buyer\_ID”, “Buyer\_name”, “Phone” and “Address” in the function of `save_excel_way()`. “Buyer\_ID” is a random eight-digit number (see Appendix 8.4). “Buyer\_Name” is the real company name in the world. The Phone represents the buyers’ phone number, which is a random eleven-digit number. Besides, “Address” refers to the buyers’ address. Every buyer has a one-to-one relation with his/her specific address.

To generate the three sets of data: “Order\_year”, “Order\_month”, and “Order\_day”, the “time” library is needed. First, set the start time tuple and end date tuple, and then generate the start timestamp and the end timestamp. After that, the date string is randomly generated through the loop, which includes the year, month, and day. The attribute “Order\_ID” is a 14-digit generated by merging the time when an order is submitted.

Attribute “Oamout” represents the number of orders and the order of magnitudes of the amount of an order is millions or tens of millions, and then rewrite it into a standard amount, such as “1,234,567”. Note that a buyer can order only one type of product in a single order.

The foreign key “Repo\_ID” refers to the ID of the warehouses. We set them as 1, 2, and 3, which will be randomly distributed to each order.

As for the “order” entity, we have generated six sets of data: “Order\_ID”, “Order\_year”, “Order\_month”, “Order\_day”, “Oamout”, and “Repo\_ID” in the function of `save_excel_way1()` (see Appendix 8.5). The “Buyer\_ID” is randomly selected from the data in the entity “buyer”.

### **3.2. Database Implementation**

After collecting the data, the first thing we need to do is to standardize their format. Adding primary key and foreign key constraints to the attributes is an essential part. Using Navicat/MySQL Workbench, for example, “Order\_ID” is the primary key of entity Order and also the foreign key of entity Buyer (see Appendix 8.6). Each entity and relation are implemented as designed in the workbench.

It is not enough to only implement a raw database. Since we are a “company” providing service for another company which produces the digital product, we also need to implement a friendly user interface for our customer. The interface should be equipped with basic searching/filtering functions to improve query efficiency so that our counterparty can use it to access their data easily.

### **3.3. UI Design**

There are two versions of our user interface. The first version is implemented with Vue and Express. The first version is a typical version of the company database control panel. Another version is relatively simpler and just like a normal website (selected as showcase). This is because we also want to implement a more straightforward and more accessible interface.

The showcases of our UI are given in Appendix 8.7 and Appendix 8.8.

Besides accessing all the data in a simple interface, there are some searching functions based on simple SQL queries to access some frequent use data (Note that here the attribute we use to screen the data is the primary key). The examples are provided in Appendix 8.9 and Appendix 8.10.

#### 4. Sample Queries

This query will retrieve all information includes component ID, component name, type, price, workshop ID, and warehouse ID. It will list “i11-shell-black” in the sidebar, when the user clicks on the super link, the information will appear on the website.

```
select *from component
WHERE Component_name = "i11-shell-black";
```

This query will show the employees with salaries between 5000 to 6000, it can roughly show the salary distribution in this factory as the number of people in different salary range is different.

```
select *from employees where salary>5000 and salary < 6000;
```

This query shows the insertion of a certain component. Each component is unique, all the information required in the insertion should be put in by users.

```
INSERT INTO `db_project`.`component`
(`Component_ID`, `Component_name`, `Category`, `in_WH`, `in_WS`)
VALUES ('0-21-1987', 'i12-shell-black', '0', '1', '2');
```

This query shows the deletion of component with ID “0-21-1641”, users only need to put component ID to decide on the attribute to delete because it is the primary key.

```
DELETE FROM `db_project`.`component`
WHERE (`Component_ID` = '0-21-1641');
```

This query returns the number of components from the repository with a component name equal to “i11-glass”, it can represent the storage of different components, products. When a user requires to know the number of different components, products, and orders. If we store the storage in a table, it needs to be refreshed from time to time, and it is a waste of time and memory because it can be calculated if needed. Also, it is important to know how many components and products are left in the repository, to save space and the time to updating, when we need to know the storage, we simply apply the count function.

```
select count(*) from component where Component_name = "i11-glass";
```



## 5. Applications

In practice, the data stored in the database will be used for various purposes; therefore, whether a database system can provide suitable data for analysis has become an important criterium of a database system's quality. Here we will illustrate several scenarios where our database can be utilized for data mining purposes. Note that the data in this section are separately and directly generated for a more illustrative demonstration.

### 5.1. Univariate Analysis

One of the most classical tasks is to predict how many products will be sold in the next month so that the company can adjust its production in time to balance the demand. Under the assumptions that the change in the amount of product sold has fully reflected all changes in relevant factors and that the change of turnover follows Gaussian distribution, we can adopt Historical Simulation or Monte-Carlo Simulation to predict the amount of product sold in the next month. Note that in this section, we ignore the product types for simplicity. In practice, the percentage of sales that one type of product takes can be studied separately.

#### 5.1.1. Historical Simulation

The Historical Simulation treats all monthly changes in demand in the past as possible scenarios that will occur in the future. The SQL query is:

```
SELECT Order_year, Order_month, SUM(Pprice * Oamount) sales
FROM order, product
GROUP BY Order_year, Order_month
HAVING order.Repo_ID = product.Repo_ID
```

Data are visualized as Appendix 8.11. Then, we apply the Historical Simulation for prediction and the pseudo-code of the algorithm is:

```
INPUT x    # The amount of product sold in past months
pred = []
for i = 2:len(x)
    pred.append(x[len(x)] * x[i] / x[i-1])
OUTPUT pred # Predicted amount of product sold in next month
```

The predicted values distribution is given in Appendix 8.12. The prediction values have a mean value of 3,428, a standard deviation of 158, and 90-th percentile of 3,638. Hence, the result suggests that although the amount of product sold in the next month will not change in a statistical sense, the company should still prepare at least 3,638 products in case of an extreme situation.

There are 2 drawbacks to the Historical Simulation in this situation. For one thing, this method can only predict monthly data using monthly data; however, the size of monthly data for a company is so seriously restricted that it is sometimes insufficient to ensure a credible result. One solution is to predict the amount of product sold in the following 31 days and sum

them up. However, here comes the second drawback that this method can only predict 1 time lag after the history, that is to say, we can only predict the next day using daily data.

### 5.1.2. Monte-Carlo Simulation

We can adopt Monte-Carlo Simulation to prevent the drawbacks of Historical Simulation. The crucial idea is to treat the predicted value at time  $t$  as ground truth and use it to predict the value at time  $t + 1$ . Hence, we use daily data to predict the amount of product sold in the following 31 days. The SQL query is:

```
SELECT SUM(Pprice * Oamount) sales
FROM order, product
HAVING order.Repo_ID = product.Repo_ID
```

Data are visualized as Appendix 8.13. Then, we apply the Monte-Carlo Simulation for prediction and the pseudo-code of the algorithm is:

```
INPUT x      # The amount of product sold in past days
trialNum = 1000
delta = []
pred = []
for i = 2:len(x)
    delta.append(x[i] - x[i-1])
mean = sum(delta) / len(delta)
var = 0
for i = 1:len(delta)
    var += (delta[i] - mean)^2 / (len(delta) - 1)
for i = 1:trialNum
    pred_values = [x[len(x)]]
    for j = 1:31
        rand <- Generate a random value from distribution N(mean, var)
        pred_values.append(max(0, pred_value + rand))
    pred.append(sum(pred_value[2:]))
OUTPUT pred  # Predicted amount of product sold in next month
```

The predicted values distribution is given in Appendix 8.14. The prediction values have a mean value of 3,346. Other statistics such as variance and percentile are less significant in this simulation because they may be exaggerated.

The company can also choose other time intervals (e.g., week) to perform this simulation if a low simulation level is preferred.

## 5.2. Multivariate Analysis

In practice, we may be interested in considering several variables at the same time. For example, we want to find the relation between variables and see how one variable will change after controlling others. These topics have made multivariate analysis very popular in many industries. In this section, we will introduce how to use database to extract data and perform

Multiple Linear Regression. For simplicity, we assume that there is only 1 factory.

Assume we are interested in whether the following factors have a significant effect on the workshop employees' salary:

- Gender
- Whether a person is yellow or not
- The workshop that the employee is in
- Age

Then we construct a linear model as:

$$\text{salary}_i = \text{age}_i + \text{isMale}_i + \text{isBlack}_i + \text{isWhite}_i + \text{isBrown}_i + \text{inWS1}_i + \text{inWS2}_i + \text{inWS3}_i + \text{inWS4}_i + \text{inWS5}_i + e_i$$

The SQL query is:

```
SELECT
    salary,
    age,
    (case when gender = "male" then 1 else 0 end) isMale,
    (case when color = "black" then 1 else 0 end) isBlack,
    (case when color = "white" then 1 else 0 end) isWhite,
    (case when color = "brown" then 1 else 0 end) isBrown,
    (case when WS_ID = 1 then 1 else 0 end) inWS1,
    (case when WS_ID = 2 then 1 else 0 end) inWS2,
    (case when WS_ID = 3 then 1 else 0 end) inWS3,
    (case when WS_ID = 4 then 1 else 0 end) inWS4,
    (case when WS_ID = 5 then 1 else 0 end) inWS5
FROM employee
WHERE WS_ID != 0
```

The regression result is shown in Appendix 8.15, which suggests that one employee's age, gender, and whether he/she is yellow are not significant in affecting the employee's salary at a 10% significant level because their p-value all exceed 0.1.

However, one employee's salary is affected by the workshop that he/she is in. The salary of employees in workshops 2, 3, and 4 does not significantly differ from the salary of employees in workshop 6, but the salary of an employee in workshops 1 and 5 differs from the salary of employees in workshop 6 at a 1% significance level. After observing that the salary of employees in workshop 6 is close to the average salary of all employees in the factory, we can also conclude that if an employee is in workshop 1, he/she probably has a higher salary than the factory average salary, and if an employee is in workshop 5, he/she probably has a lower salary than the factory average salary.

## 6. Conclusion

In this project, we considered a company that assembles digital products and it requires a

powerful database system to manage its production and sales. After analyzing the company's demands, we designed a Production & Sales Management System to ensure efficient operations of the company.

We recognized factories, workshops, repositories, employees, components, products, buyers, and orders as main entities and studied the relation between them. Then, we designed the ER diagram and tried to combine relations with entities to improve query efficiency and lower the data redundancy level. We also reduce the refined ER diagram to the relational schemas and verified that they are in the Boyce Codd Normal Form.

Besides, we tried to bring our design into reality by generating data, establishing a real database, importing them into the database. To demonstrate our database system in a more straightforward way, we created some sample queries and designed a friendly user interface. In addition, we also visualized certain situations where our database can be used for data mining purposes. The abovementioned sample queries, data mining potentials, and user interface have proved that our database system can satisfy not only the company's daily operation but also further demands in data analysis.

## **7. Self-evaluation**

In this project, we learned to think from both the customers' view and from a database engineers' view. It helps us to better understand the actual implementation of a database and the real need for a database. From deciding on which topic to do, everyone in our group showed anticipation in this project. The discussion and job splitting are efficient. The data are generated by 3 of our group mates. The whole group collaborates and communicates efficiently. The database we design is simple but complete. For information display, we listed out the useful information in the sidebar, which can simplify the searching process. To conclude, our group finishes the task of designing a complete database for a factory. It stores the human resources, products, components, and orders. It will simplify the management of a factory a lot.

## 8. Appendix

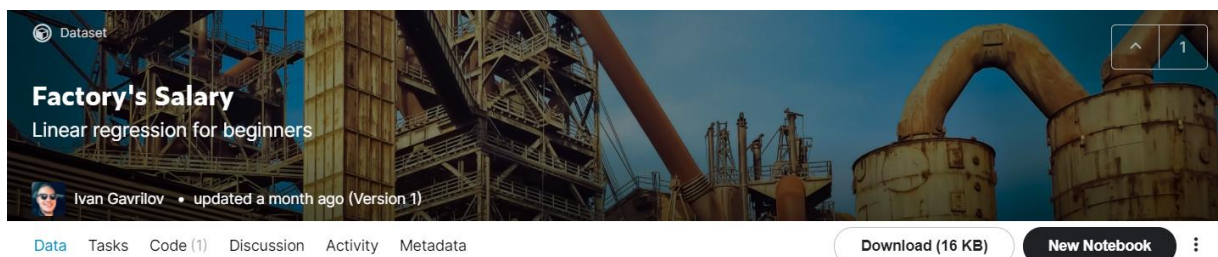
Relevant files have been included in the zip file.

Repo_ID	Rmgr_ID	factory_name
1	337148842	Foxcomm
2	306994878	Foxcomm
3	594116286	Foxcomm

Appendix 8.1



Appendix 8.2



Appendix 8.3

```
#first column buyer id
for i in range(1,888):
    data = ''.join(str(random.choice(range(10))) for _ in range(8))
    ws.write(i, 0, data) #row, column, data

#second column buyer name
for i in range(1,888):
    data = buyer_list[random.choice(range(len(buyer_list)))]
    service_for_address_list.append(data)
    ws.write(i,1,data)

#third column phone number
for i in range(1,888):
    num_start = "1"
    num_follow = ''.join(str(random.choice(range(10))) for _ in range(10))
    data = num_start + num_follow
    ws.write(i,2,data)

#forth column address
for i in range(1,888):
    data = dict_address_buyer[service_for_address_list[i-1]]
    #data = address_list[random.choice(range(len(address_list)))]
    ws.write(i,3,data)

#fifth column order id
data = pd.read_excel("order.xls",header= None) # read Excel files

for i in range(1,data.shape[0]):
    ws.write(i,4,data.iloc[i][0])

wb.save("buyer.xls") # save as Excel files
```

Appendix 8.4

```

import xlwt
import random
import time
import pandas as pd
from random import choice

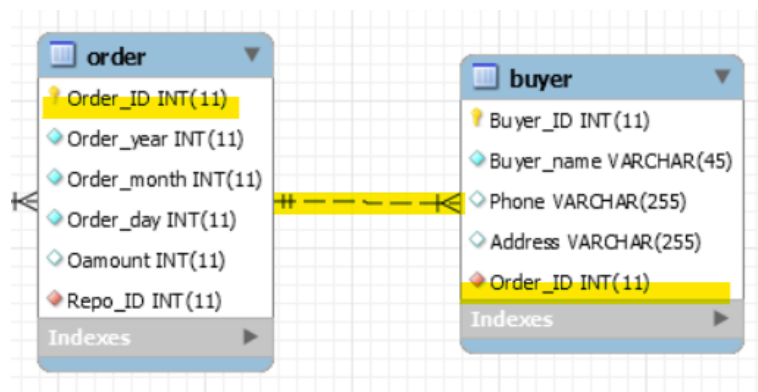
def save_excel_way():
    # creat Excel files
    wb = xlwt.Workbook()
    ws = wb.add_sheet('keyword_extract')

    # change row
    style = xlwt.XFStyle() # Initialization style
    style.alignment.wrap = 1 # Wrap automatically

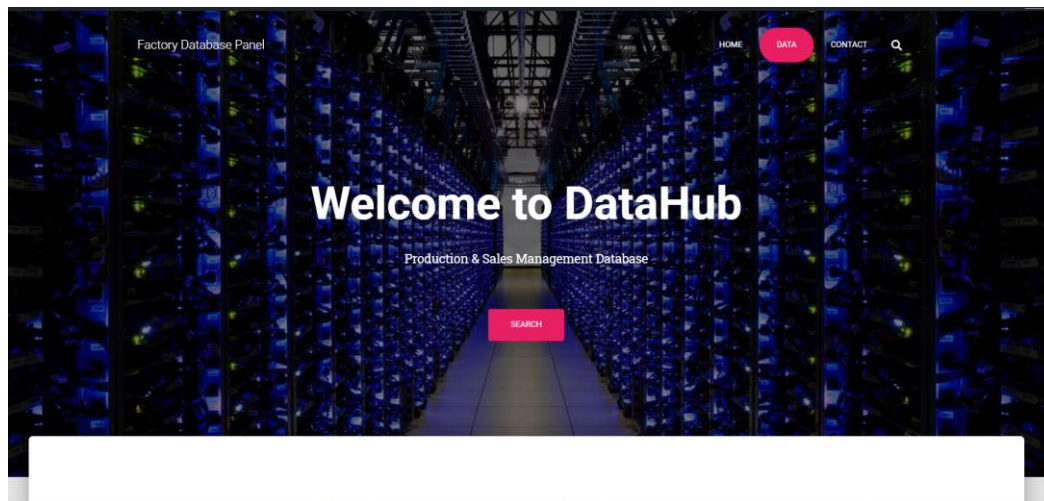
    ws.write(0, 0, "Buyer_ID")
    ws.write(0, 1, "Buyer_name")
    ws.write(0, 2, "Phone")
    ws.write(0, 3, "Address")
    ws.write(0, 4, "Order_ID")

```

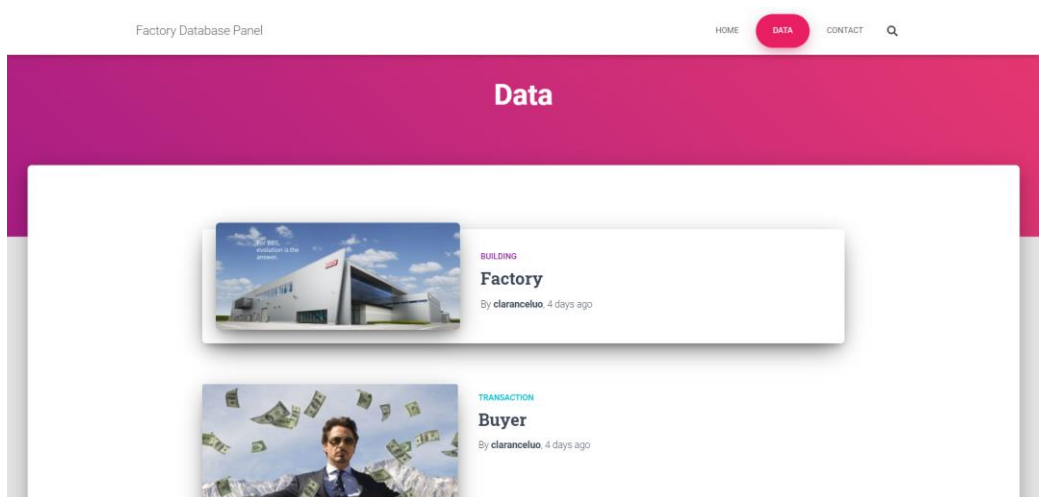
Appendix 8. 5



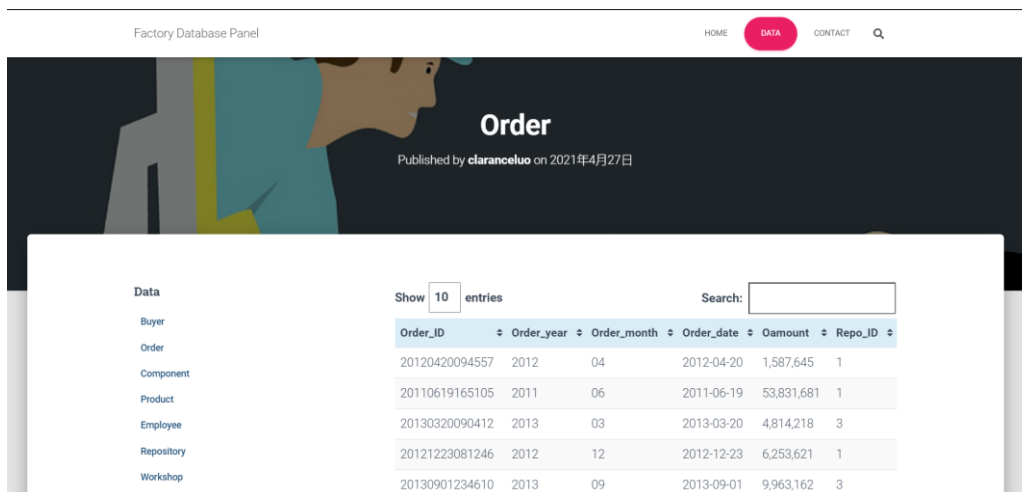
Appendix 8. 6



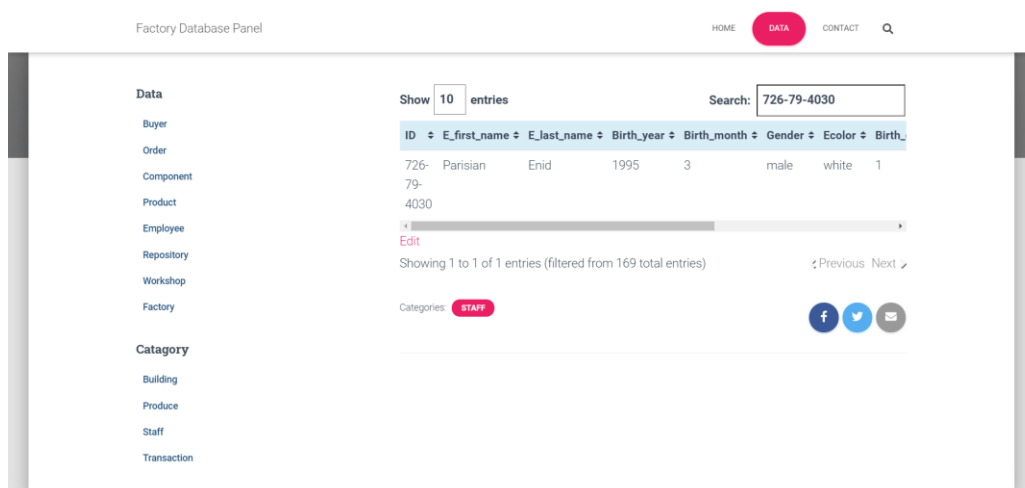
Appendix 8. 7 Home Page



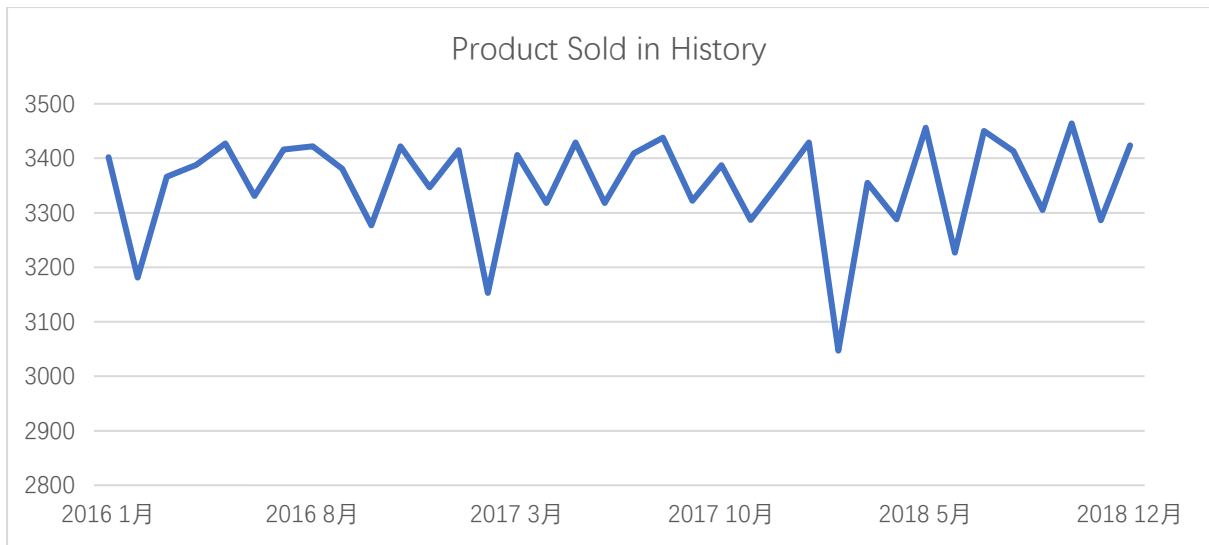
Appendix 8. 8 Dataset Page



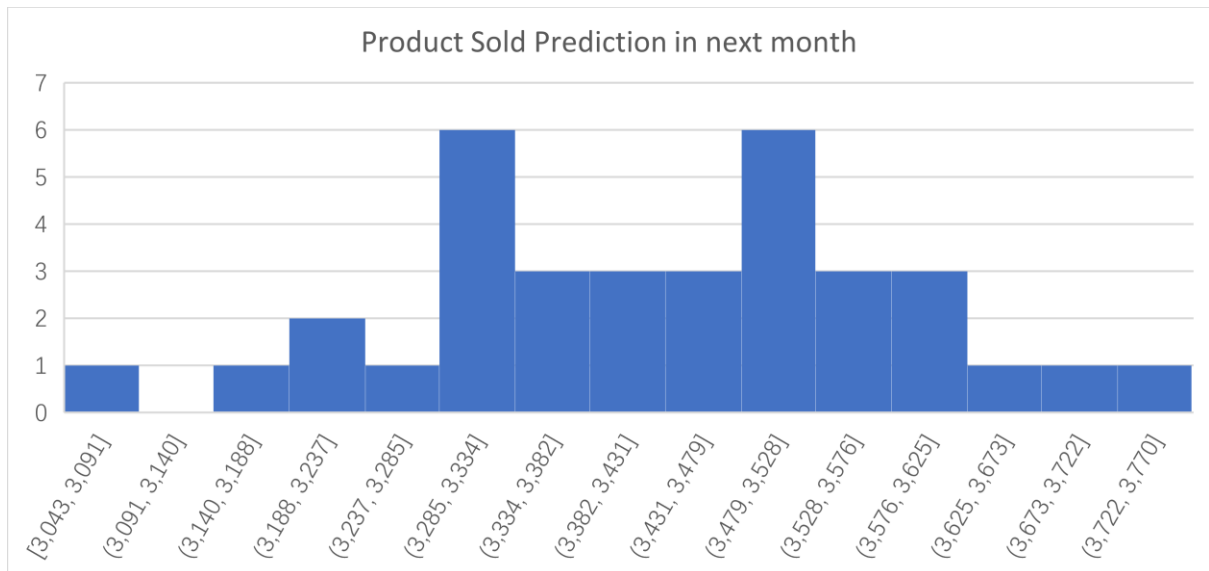
Appendix 8. 9 Order Entity



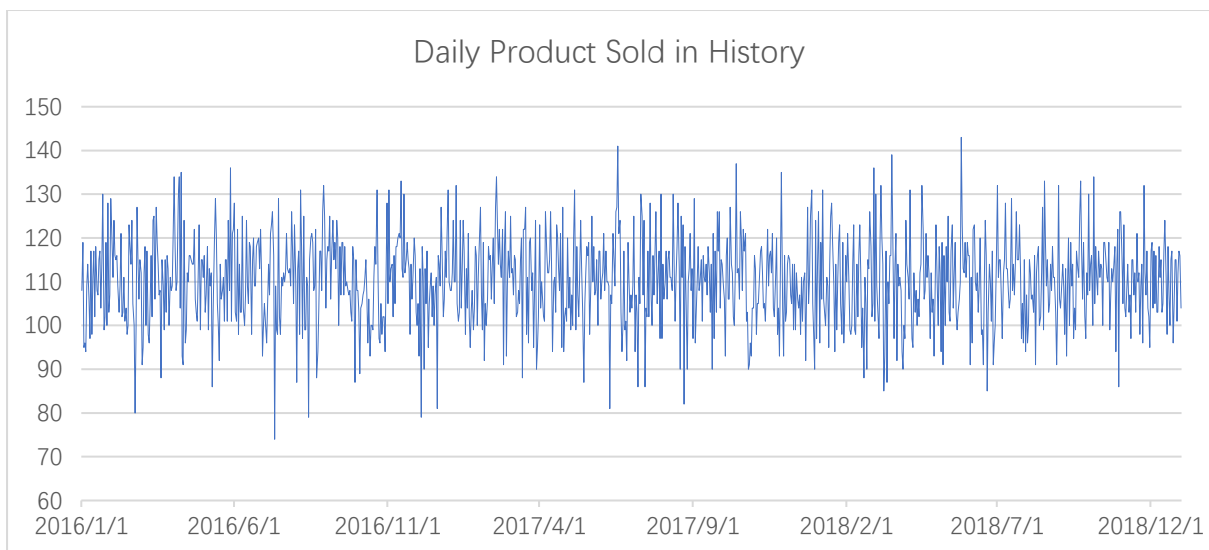
Appendix 8. 10 Employee Entity (Select on ID)



Appendix 8. 11

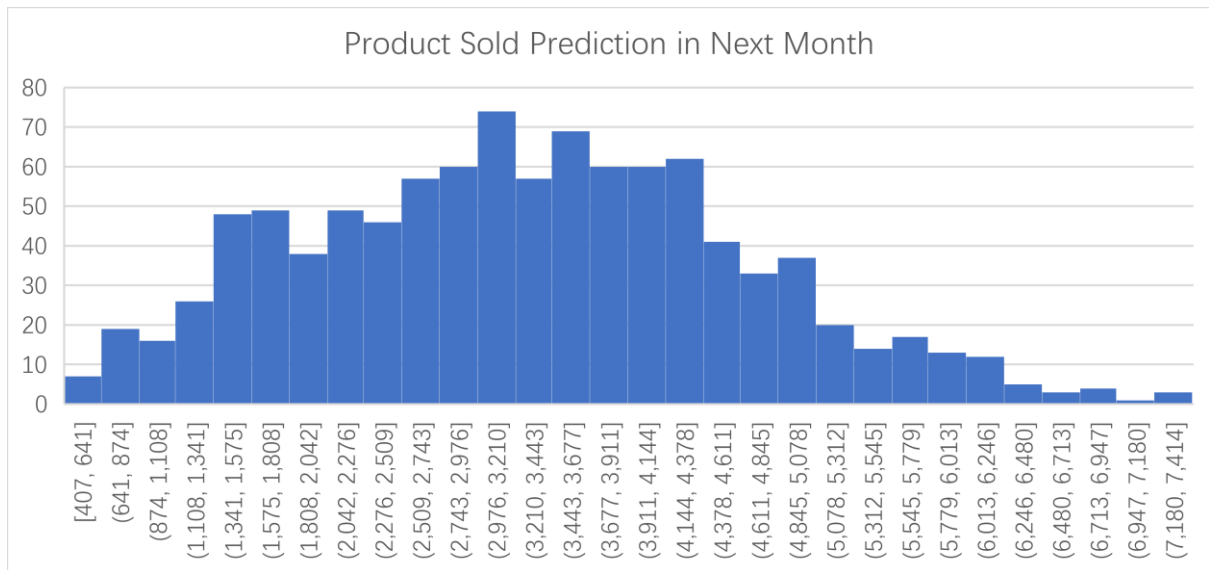


Appendix 8. 12



Appendix 8. 13





Appendix 8. 14

Source	SS	df	MS	Number of obs	=	6,605
Model	403062222	10	40306222.2	F(10, 6594)	=	375.84
Residual	707153560	6,594	107241.971	Prob > F	=	0.0000
				R-squared	=	0.3630
				Adj R-squared	=	0.3621
Total	1.1102e+09	6,604	168112.626	Root MSE	=	327.48

salary	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
age	.2000208	.4512685	0.44	0.658	-.6846116	1.084653
isMale	-8.089494	8.323564	-0.97	0.331	-24.40637	8.227387
isBlack	-13.02173	11.66721	-1.12	0.264	-35.89324	9.849773
isWhite	1.673103	10.41647	0.16	0.872	-18.74656	22.09276
isBrown	1.784479	11.66551	0.15	0.878	-21.08369	24.65265
inWS1	367.3181	13.8979	26.43	0.000	340.0737	394.5625
inWS2	2.806892	13.71164	0.20	0.838	-24.07236	29.68615
inWS3	-16.49909	13.80149	-1.20	0.232	-43.55448	10.55629
inWS4	-3.347355	13.93094	-0.24	0.810	-30.65651	23.9618
inWS5	-494.7215	13.9521	-35.46	0.000	-522.0721	-467.3708
_cons	6139.34	21.01467	292.15	0.000	6098.144	6180.535

Appendix 8. 15