Golden Ware Kitchenware Distributor

Yingxi Zhou, Sabrina Zhao, Minsheng Li, Sean Li, Won Park, Jui-Chen Pu, Sean Tsung, Mengfan Zhang

Our Journey

First Presentation

- Client description
- Database requirements
- EER diagram

DP Review I

DP Review II

Second Presentation

- Updated EER diagram
- Relational design (schema)
- Interesting queries

DP Review III

Project Report

- Executive summary
- MySQL tables and queries

Today

Final Presentation

- Client overview
- Simplified EER
- Relational design
- Database and queries
- Normalization

Agenda

1. Client Overview
2. Simplified EER
3. Relational Design
4. Database and Queries
5. Normalization

Client Overview

Simplified EER

Relational Design

Database and Queries

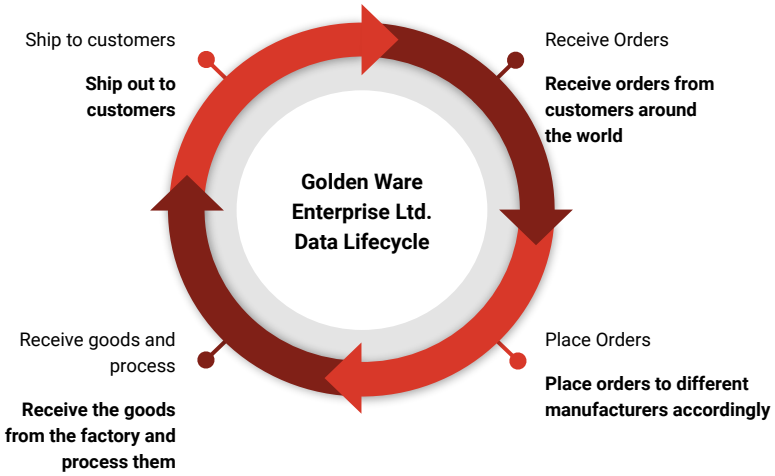
Normalization

Our Client: Golden Ware Enterprise Ltd.

- Located in Guangzhou, China
- Kitchenware supplier for supermarket chains
- Stainless steel kitchenware
- Products: vacuum flasks, cutlery sets, chafing dishes, knife sets...
- ~25 employees
- Products found all over Europe and the Middle East

Database Requirements

Data Lifecycle



Benefits of a Database

- Logistical optimization
- Customer Analysis
- Data consistency

Goals and Stakeholders

- Customer behavior prediction
- Employee engagement

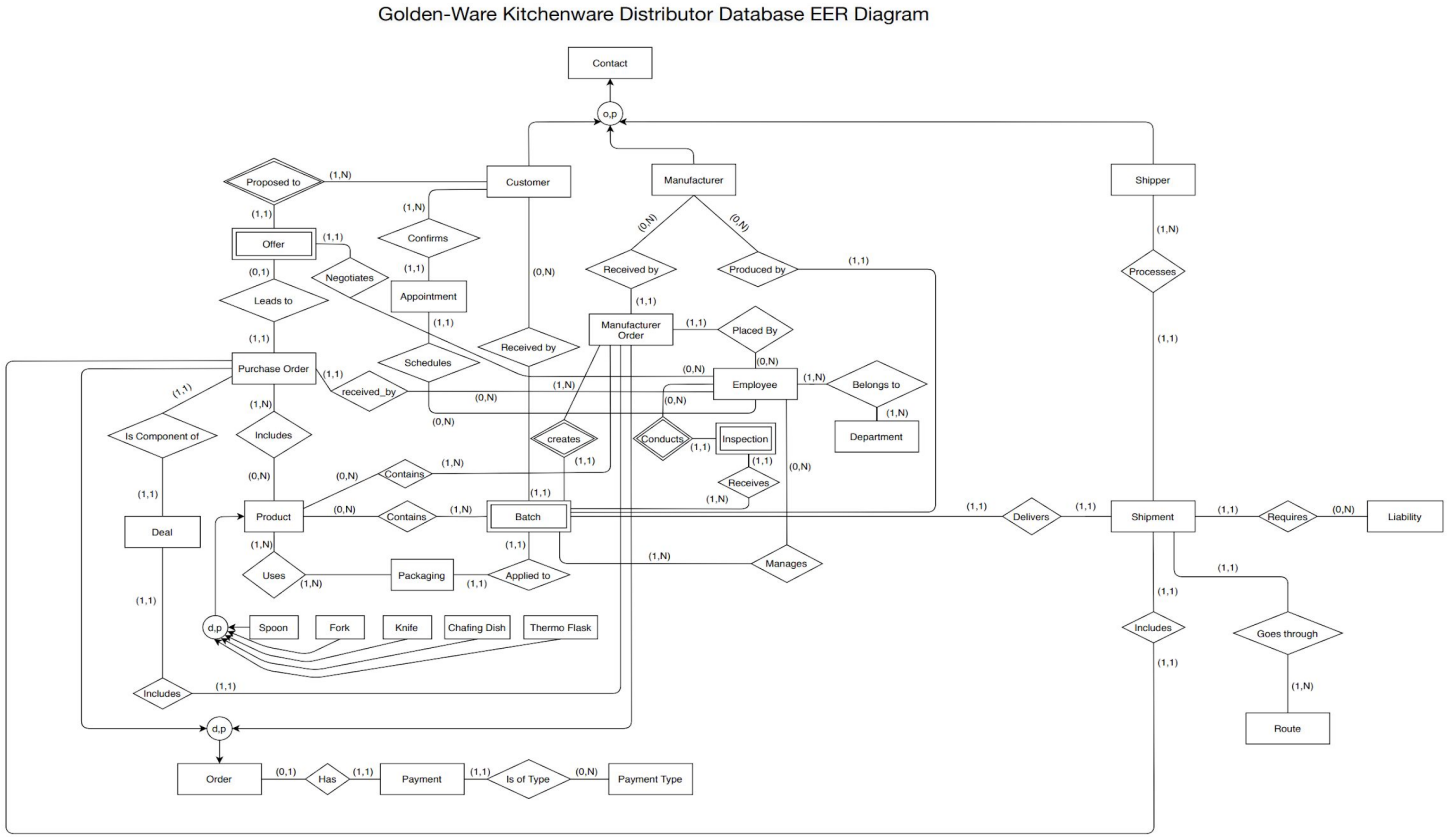
Client Overview

Simplified EER

Relational Design

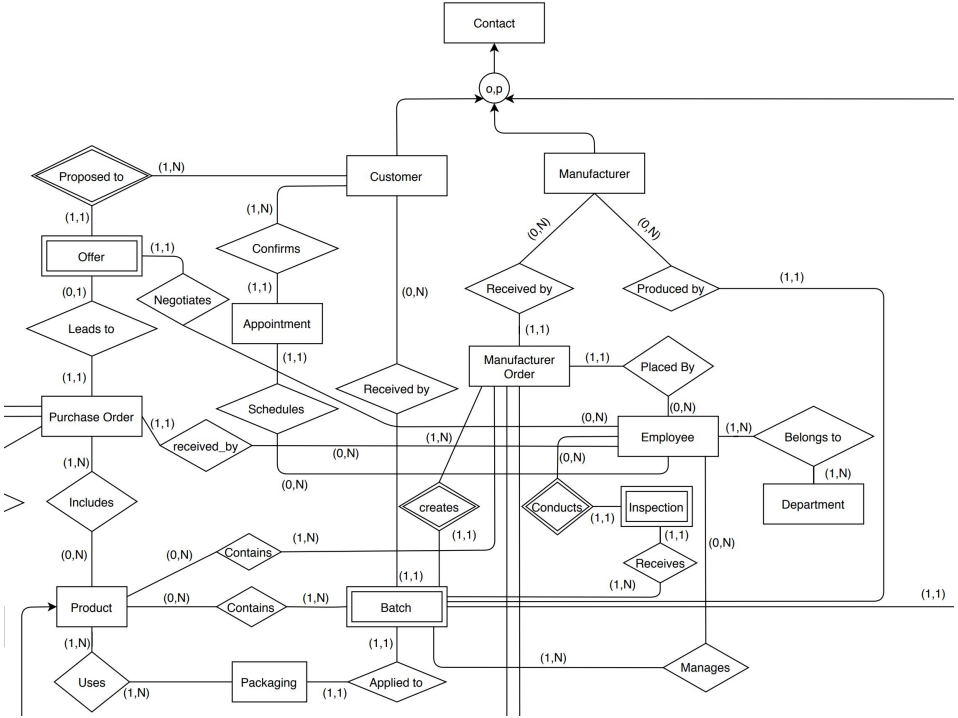
Database and Queries

Normalization



Simplified EER Diagram

Simplified EER Diagram



Client Overview

Simplified EER

Relational Design

Database and Queries

Normalization

Relational Design (Schema)

1. CONTACT (contact_ID, address, liaison_phone, email_Address, Location, liaison_name,
   1a. Customer(contact_ID[1], company_Size)
   1b. Manufacturer(contact_ID[1], plant_size, flexibility, portdist)
   1c. Shipper(contact_ID[1], shipping_Cost_Rate, shipping_Discount)
2. DEAL (deal_Id, net_Profit, tax_Return, actual_Shipping_Date, actual_Received_Date, customer_Satisfaction, revenue, manufactured_Costs, shipping_Cost, delay_Rate, payment_Received_date, inspection_Fees, Order_ID[3a], Order_ID[3b])
3. PLACED_ORDER (order_ID, order_date, shipping_Cost, exp_Shipping_Date, company_liaison)
   3a. PURCHASE_ORDER (order_ID[3], discount, recurrent, currency_Exchange_Rate, employee_Id[8], offer_Name[15], customer_Contact_Id[15])
   3b. MANUFACTURER_ORDER (order_ID[3], company_title, employee_Id[8], contact_ID[1b])
4. PRODUCT (product_Id, p_Name, ordered_Frequency_per_Season, p_Type, p_Design, baseline_Price, minimum_Ordered_Amount, p_material, p_Weight)
   4a. SPOON (product_Id[4], firmness)
   4b. FORK (product_Id[4], length)
5. BATCH (batch_Id, order_ID[3b], length, width, height, weight, inspector_Name, shipping_Method, shipping_Company, contact_ID[1a], contact_ID[1b], date[16], employee_Id[8], shipment_Id[10])
6. PAYMENT (payment_Id, amount_Paid, payment_Date, paid_To_Account_ID, paid_From_Account_ID, method_Name[7])
7. PAYMENT_TYPE (method_Name, transaction_Fees, bank_Title, trasnfer_Amount_Limit)

8. EMPLOYEE (employee_Id, first_Name, last_Name, middle_Initials, personal_ID, department, salary, title, region_In_Charge)
9. DEPARTMENT (department_Id, Department Name, number_Of_Employees, Location, revenue_created, utilities_Fee, department_Manager)
10. SHIPMENT (shipment_Id, ship_Date, volumn, Weight, Expected_arrival, Actual_arrival, shipment_cost, route_Name[12], liability_Id[11], contact_ID[1c])
11. LIABILITY (liability_Id, Is_free_on_board, liability_Type, liability_Rate)
12. ROUTE (route_Name, distance, destination, cargo_Origin, shipping_method, limitation)
13. APPOINTMENT (appointment_Id, date, appointment_Method, appointment_Feedback, employee_Id[8], contact_ID[1a])
14. PACKAGING (packing_Method_Name, weight, brand, cost, packaging_material, baling_material)
15. OFFER (offer_ID, customer_Contact_Id[1], proposed_date, product, deadline_to_Accept, Is_Accepted, employee_Id[8])
16. INSPECTION (date, employee_Id[8], inspection fees, requirement, batch_Id, order_ID[3b])
17. PURCHASE_ORDER_INCLUDES_PRODUCT (order_ID[3a], product_Id[4], amount, price)
18. PRODUCT_USES_PACKAGING (product_Id[4], packing_Method_Name[14])
19. BATCH_CONTAINS_PRODUCT (batch_Id, order_ID[3b], product_Id[4])
20. MANUFACTURER_ORDER_CONTAINS_PRODUCT (order_ID[3b], product_Id[4], amount, price)
21. EMPLOYEE_BELONGS_TO_DEPARTMENT (employee_Id[8], department_Id[9])
22. EMPLOYEE_CONDUCTS_INSPECTION (employee_Id[8],date[16], employee_Id[8])
23. EMPLOYEE_MANAGES_BATCH (employee_Id[8], batch_Id[5], order_ID[3b])

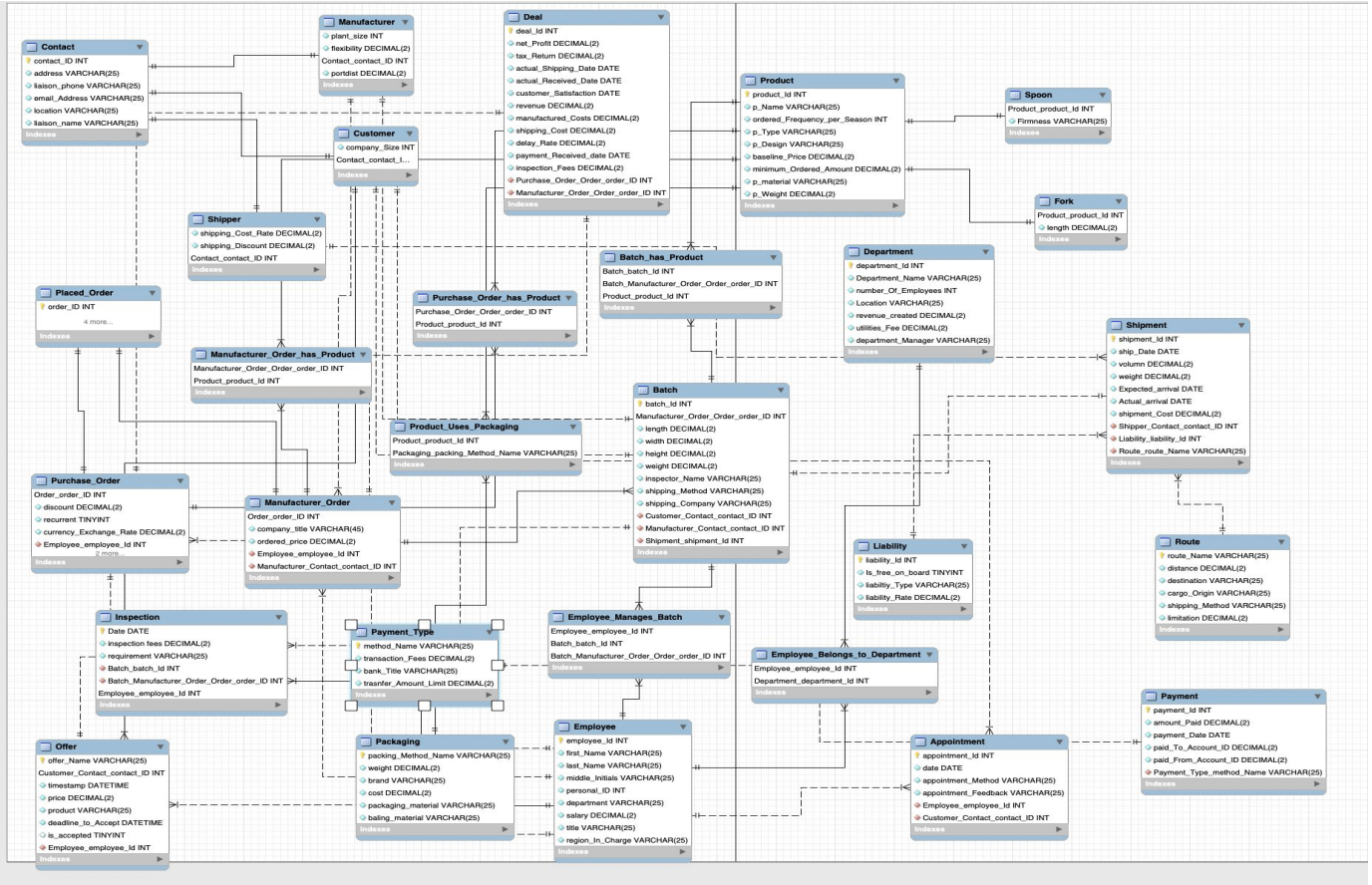
Client Overview

Simplified EER

Relational Design

Database and Queries

Normalization



Database Design - MySQL

Query 1: Customer Loyalty – Overview

Customers are evaluated based on three perspectives: economic contribution, stability of partnership and size of company. We will quantify these factors using a 'loyalty score' which is defined as follows:

Loyalty score =

θ_1 × contribution to annual revenue (% of total revenue)

+ θ_2 × frequency of orders (number of orders)

+ θ_3 × size of the customer company

Rationale:

- Foster long-term collaborations between the client and its customers
- Provide exclusive promotions/discounts for loyal customers

N.B.: The weights provided by the client are $\theta_1 = 0.2$, $\theta_2 = 0.7$, $\theta_3 = 0.1$;
see later slides for an evaluation of the provided weights.

Query 1: Customer Loyalty – SQL

MySQL Implementation

```sql
CREATE OR REPLACE VIEW totalRevenue AS (
    SELECT sum(revenue) as total
    FROM DEAL
);

CREATE OR REPLACE VIEW customerData AS (
    SELECT C.Contact_contact_ID AS CID, SUM(D.REVENUE) / (SELECT TOTAL FROM TOTALREVENUE) AS CRV, COUNT(*) AS FREQUENCY
    FROM DEAL AS D, PURCHASE_ORDER AS P, CUSTOMER AS C
    WHERE D.Purchase_Order_Order_order_ID = P.Order_order_ID
    AND P.Offer_Customer_Contact_contact_ID = C.Contact_contact_ID
    GROUP BY C.Contact_contact_ID
);

SELECT C.Contact_contact_ID AS CUSTOMER, CD.CRV * 0.2 + CD.frequency * 0.7 + C.company_Size * 0.1 AS LOYALTYSCORE
FROM CUSTOMER C, customerData CD
WHERE C.Contact_contact_ID = CD.CID
ORDER BY LOYALTYSCORE DESC;
```

Query 1: Customer Loyalty – Import Data

Synthetic Data - Table DEAL

```
DEAL = pd.DataFrame()
fake2 = Faker()
num_deals = 300

for i in range(num_deals):
    new_row = {# unique order ID
               'deal_Id': fake2.unique.random_int(min=401, max=800, step=1),
               # revenue, normal distribution
               'revenue': int(random.normalvariate(2000000, 200000)),
               # delay_Rate with extreme weather in mind
               'delay_Rate': (30 / 365) * 0.4 + (335 / 356) * math.floor(random.normalvariate(0.08, 0.01))
              }
    new_row['customer_Satisfaction'] = int(random.normalvariate(8.5, 0.5) - 0.15 * new_row['delay_Rate'])
    new_row['manufactured_Costs'] = int(random.uniform(new_row['revenue'] * 0.4, new_row['revenue'] * 0.5))
    new_row['shipping_Cost'] = int(random.uniform(new_row['revenue'] * 0.08, new_row['revenue'] * 0.2))
    new_row['inspection_Fees'] = int(random.uniform(new_row['revenue'] * 0.02, new_row['revenue'] * 0.03))
    new_row['Purchase_Order_Order_order_ID'] = random.randint(0, 300)
    new_row['Manufacturer_Order_Order_order_ID'] = random.randint(0, 300)

    DEAL = DEAL.append(new_row, ignore_index = True)
```

Query 1: Customer Loyalty – Import Data

Synthetic Data - Table DEAL

```
DEAL = pd.DataFrame()
fake2 = Faker()
num_deals = 300

for i in range(num_deals):
    new_row = {# unique order ID
               'deal_Id': fake2.unique.random_int(min=401, max=800, step=1),
               # revenue, normal distribution
               'revenue': int(random.normalvariate(2000000, 200000)),
               # delay_Rate with extreme weather in mind
               'delay_Rate': (30 / 365) * 0.4 + (335 / 356) * math.floor(random.normalvariate(0.08, 0.01))
              }
    new_row['customer_Satisfaction'] = int(random.normalvariate(8.5, 0.5) - 0.15 * new_row['delay_Rate'])
    new_row['manufactured_Costs'] = int(random.uniform(new_row['revenue'] * 0.4, new_row['revenue'] * 0.5))
    new_row['shipping_Cost'] = int(random.uniform(new_row['revenue'] * 0.08, new_row['revenue'] * 0.2))
    new_row['inspection_Fees'] = int(random.uniform(new_row['revenue'] * 0.02, new_row['revenue'] * 0.03))
    new_row['Purchase_Order_Order_order_ID'] = random.randint(0, 300)
    new_row['Manufacturer_Order_Order_order_ID'] = random.randint(0, 300)

    DEAL = DEAL.append(new_row, ignore_index = True)
```

Query 1: Customer Loyalty – Import Data

Synthetic Data - Table MANUFACTURER

```
from collections import OrderedDict
for i in range(40, 101):
    m = {
        'plant_size':
        fake.random_element(elements=OrderedDict
                                ([("S", 0.2), ("M", 0.35), ("L", 0.45)])),
        'flexibility': random.normalvariate(0.7, 0.08),
        'Contact_contact_ID': i,
        'portdist': math.floor(random.normalvariate(800, 80)),
        'manufacturer_name': fake.company()
    }

    MANUFACTURER = MANUFACTURER.append(m,ignore_index = True)
DataFrame_to_MySQL(MANUFACTURER,"MANUFACTURER")
```

Query 1: Customer Loyalty – Import Data

Synthetic Data - Table MANUFACTURER

```
from collections import OrderedDict
for i in range(40, 101):
    m = {
        'plant_size':
        fake.random_element(elements=OrderedDict
                                ([("S", 0.2), ("M", 0.35), ("L", 0.45)])),
        'flexibility': random.normalvariate(0.7, 0.08),
        'Contact_contact_ID': i,
        'portdist': math.floor(random.normalvariate(800, 80)),
        'manufacturer_name': fake.company()
    }

    MANUFACTURER = MANUFACTURER.append(m,ignore_index = True)
DataFrame_to_MySQL(MANUFACTURER,"MANUFACTURER")
```

Query 1: Customer Loyalty – Import Data

Synthetic Data - Table CONTACT

```
shipper = {
                'contact_ID': 30,
                'address': fake.address(),
                'email_Address': fake.email(),
               }
CONTACT = CONTACT.append(shipper,ignore_index = True)
```

```
CONTACT = pd.DataFrame()
for i in range(21):
    customer = {
                'contact_ID': i,
                'address': fake.address(),
                'email_Address': fake.email(),
               }

    CONTACT = CONTACT.append(customer,ignore_index = True)
```

```
for i in range(40, 101):
    manufacture = {
                'contact_ID': i,
                'address': fake.address(),
                'email_Address': fake.email(),
               }

    CONTACT = CONTACT.append(manufacture,ignore_index = True)
```

Synthetic Data - Table CUSTOMER

```
CUSTOMER = pd.read_sql_query('SELECT * FROM CUSTOMER', connection)
for i in range(0, 21):
    c = {
    'company_Size':
     random.randint(50, 400),
    'Contact_contact_ID': i,
    'customer_name': fake.company(),
    }

    CUSTOMER = CUSTOMER.append(c,ignore_index = True)
```

Query 1: Customer Loyalty – Import Data

Synthetic Data - Table CONTACT

```
shipper = {
                'contact_ID': 30,
                'address': fake.address(),
                'email_Address': fake.email(),
               }
CONTACT = CONTACT.append(shipper,ignore_index = True)
```

Synthetic Data - Table CUSTOMER

```
CUSTOMER = pd.read_sql_query('SELECT * FROM CUSTOMER', connection)
for i in range(0, 21):
    c = {
    'company_Size':
     random.randint(50, 400),
    'Contact_contact_ID': i,
    'customer_name': fake.company(),
    }

    CUSTOMER = CUSTOMER.append(c,ignore_index = True)
```

```
CONTACT = pd.DataFrame()
for i in range(21):
    customer = {
                'contact_ID': i,
                'address': fake.address(),
                'email_Address': fake.email(),
               }

    CONTACT = CONTACT.append(customer,ignore_index = True)
```

```
for i in range(40, 101):
    manufacture = {
                'contact_ID': i,
                'address': fake.address(),
                'email_Address': fake.email(),
               }

    CONTACT = CONTACT.append(manufacture,ignore_index = True)
```

Query 1: Customer Loyalty – Import Data

Synthetic Data - Table PURCHASE_ORDER

Highlights: skewed recurrent rate

```
num_pos = 300

for i in range(num_pos):
    new_row = {
        'Order_order_ID': i * 2 + 1,
        'discount': random.normalvariate(0.1, 0.02),
        'recurrent': np.random.choice([0, 1], 1, [0.3, 0.7])[0],
        'Employee_employee_Id': randind(0, 30),
        'Customer_ID': randind(0, 21)
    }

    PURCHASE_ORDER = PURCHASE_ORDER.append(new_row,ignore_index = True)
```

```
num_mos = 300
MANUFACTURER_ORDER = pd.read_sql_query('SELECT * FROM MANUFACTURER_ORDER', connection)
for i in range(num_mos):
    mo = {
        'Order_order_ID': i * 2,
        'Employee_employee_Id': random.randint(0, 30),
        'Manufacturer_Contact_contact_ID': random.randint(40, 101)
    }

    MANUFACTURER_ORDER = MANUFACTURER_ORDER.append(mo,ignore_index = True)
```

Synthetic Data - Table MANUFACTUER_ORDER

Query 1: Customer Loyalty - Data Analysis

Major Metrics:

- Contribution to annual revenue (% of total revenue)
- Frequency of orders (number of orders)
- Size of the customer company

What we did:

- Data Cleaning
- Normalization: 0-1 scale
- 3D plot - visualization
- Dimensional Reduction using PCA (linear)
- Normalization - Loyalty Score Simulation

Query 1: Customer Loyalty - Data Analysis

What we can do with this analysis later on:

- Compare the loyalty score of a company throughout the years to measure its stability
- Calculate the difference between our pre-assumed coefficients & simulated ones

Query 1: Customer Loyalty - Data Analysis

```
[187]
from dateutil import parser
merged = deal_purchase_df.merge(placed_df,left_on='Order_ID', right_on='order_ID')
merged['order_date'] = merged['order_date'].apply(parser.parse)
merged['year'] = merged['order_date'].dt.to_period('Y')
merged.head().T
```

	0	1	2	3	4
deal_Id	1	2	3	4	5
revenue	1757589	2156797	1695264	1859044	1780035
delay_Rate	0.4	0.4	0.4	0.2	0.4
customer_Satisfaction	6.4014	6.1296	6.11447	9.4783	6.03976
manufactured_Costs	872094	1050060	724097	798404	802437
shipping_Cost	285463	421575	317792	192966	354275
inspection_Fees	40912	52472	35803	39354	36433
Purchase_Order_ID	1	3	5	7	9
Manufacturer_Order_ID	2	4	6	8	10
Order_ID	1	3	5	7	9
discount	0.104401	0.0649553	0.0818077	0.118894	0.102388
recurrent	0	0	1	1	1
Employee_Id	1	1	1	1	1
Customer_ID	1	2	3	4	5
order_ID	1	3	5	7	9
order_date	2020-03-30 00:00:00	2019-10-03 00:00:00	2020-06-13 00:00:00	2020-11-02 00:00:00	2020-06-24 00:00:00
year	2020	2019	2020	2020	2020

Data Analysis

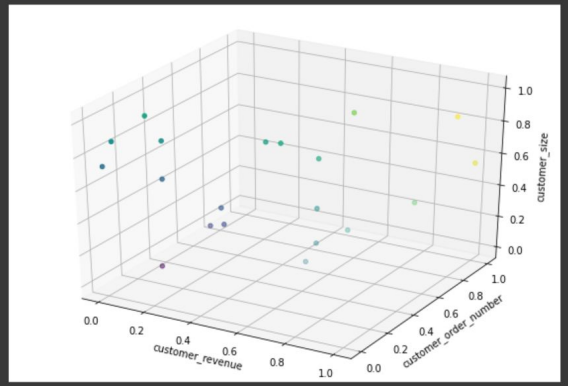
Query 1: Customer Loyalty – Data Analysis

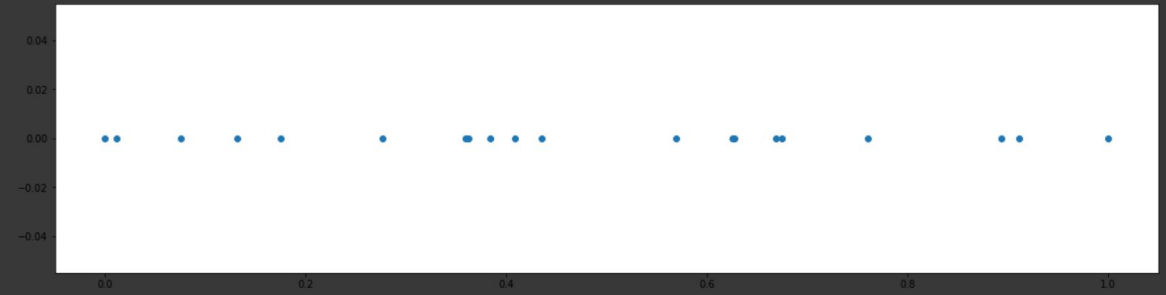
```python
for year in merged['year'].unique():
  print('In year ', year)
  temp = merged[merged['year'] == year]
  customer_revenue = temp.groupby('Customer_ID').sum()['revenue'] / sum(temp['revenue'])
  customer_order_number = temp.groupby('Customer_ID').count()['deal_Id']
  customer_size = customer_df.groupby('Contact_ID').sum()['company_Size']
  d = {'customer_revenue': customer_revenue, 'customer_order_number': customer_order_number, 'customer_size': customer_size}
  df = pd.DataFrame(data=d).dropna()
  df = MinMaxScaler().fit_transform(df)
  df = pd.DataFrame(data=df, columns=('customer_revenue', 'customer_order_number', 'customer_size'))
  df['dist'] = np.sqrt(df['customer_revenue'] **2 + df['customer_order_number'] ** 2 + df['customer_size'] ** 2)

  fig = plt.figure(figsize = (10, 7))
  ax = fig.add_subplot(111, projection='3d')
  ax.scatter(df['customer_revenue'], df['customer_order_number'], df['customer_size'], c=df['dist'])
  #ax.set_xlim(0,  0.04)
  #ax.set_zlim(0,  350)
  ax.set_xlabel('customer_revenue')
  ax.set_ylabel('customer_order_number')
  ax.set_zlabel('customer_size')
  plt.show()
  pca = PCA(n_components=1)
  principalComponents = pca.fit_transform(df.iloc[:, :3])
  principalComponents = [component[0] for component in principalComponents]
  principalComponents = sklearn.preprocessing.minmax_scale(principalComponents, feature_range=(0, 1), axis=0, copy=True)
  fig = plt.figure(figsize = (20, 5))
  plt.scatter(x=principalComponents, y=np.zeros(len(principalComponents)))
  data = {'Customer_ID': customer_revenue.index, 'Revenue': customer_revenue, 'Loyalty Score': principalComponents}
  df = pd.DataFrame(data).reset_index(drop=True)
  print(df)
  #print(principalComponents)

```

Data Analysis

Query 1: Customer Loyalty - Data Analysis

In year 2020



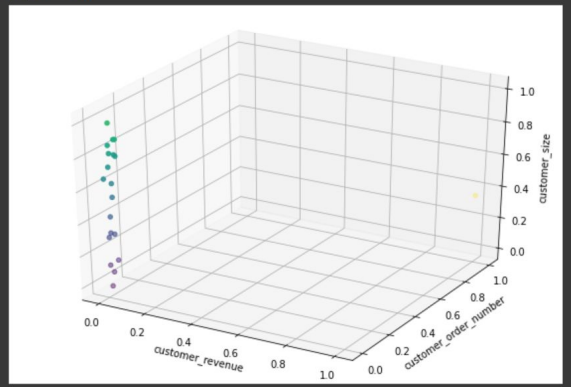


	Customer_ID	Revenue	Loyalty Score
0	1	0.034798	0.276254
1	2	0.032046	0.075314
2	3	0.054423	0.674488
3	4	0.059804	0.760428
4	5	0.080093	1.000000

Data Analysis

Query 1: Customer Loyalty - Data Analysis

```
[191] In year  2019
```

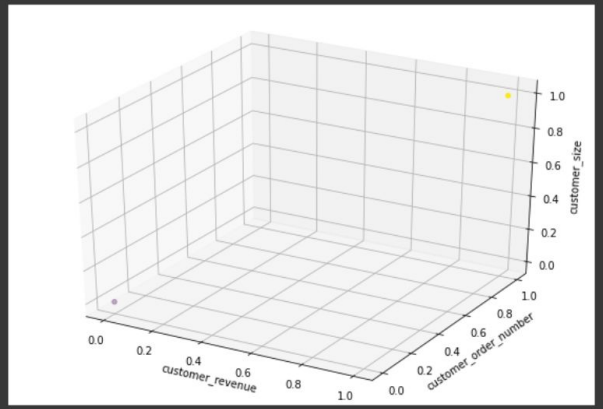


	Customer_ID	Revenue	Loyalty Score
0	1	0.015441	0.460466
1	2	0.015887	0.000000
2	3	0.011770	0.501028
3	4	0.023608	0.432821
4	5	0.018039	0.227710

Data Analysis

Query 1: Customer Loyalty - Data Analysis

In year 2018



	Customer_ID	Revenue	Loyalty Score
0	9	0.4446	1.0
1	12	0.5554	0.0

Data Analysis

Query 2: Offer Success Rate – Overview

Golden Ware records all offers proposed to customers along with information about the employee who negotiates the offer and the customer who receives the offer.

The output from this query is used to calculate/predict the probability that an offer is accepted.

Rationale

- Allows client to pay extra attention to offers that are predicted to be accepted
- Allows client to give a bonus to employees who have a high offer signing rate

Query 2: Offer Success Rate – SQL

```sql
-- Find total # of offers proposed by each employee
CREATE OR REPLACE VIEW employeeTotal AS (
    SELECT Employee_ID, COUNT(*) AS TOTAL
    FROM OFFER
    GROUP BY Employee_ID
);
-- Find total # of successful offers by each employee
CREATE OR REPLACE VIEW employeeS AS (
    SELECT Employee_ID, COUNT(*) AS TOTAL
    FROM OFFER
    WHERE OFFER.ACCEPTED = 1
    GROUP BY Employee_ID
);
CREATE OR REPLACE VIEW employeeRate AS (
    SELECT ES.TOTAL / ET.TOTAL
    FROM ((employeeTotal AS ET)
    INNER JOIN (employeeS AS ES)
    ON ET.Employee_ID)
);
```

```sql
-- Find total # of offers proposed to each customer
CREATE OR REPLACE VIEW customerTotal AS (
    SELECT Customer_ID, COUNT(*) AS TOTAL
    FROM OFFER
    GROUP BY Customer_ID
);
-- Find total # of successful offers to each customer
CREATE OR REPLACE VIEW customerS AS (
    SELECT Customer_ID, COUNT(*) AS TOTAL
    FROM OFFER
    WHERE OFFER.ACCEPTED = 1
    GROUP BY Customer_ID
);
CREATE OR REPLACE VIEW customerRate AS (
    SELECT CS.TOTAL / CT.TOTAL
    FROM ((customerTotal AS CT)
    INNER JOIN (customerS AS CS)
    ON CS.Customer_ID)
);
```

Query 2: Offer Success Rate – SQL

```sql
-- Find total # of offers proposed in each quarter
CREATE OR REPLACE VIEW quarterTotal AS (
    SELECT O_Quarter, COUNT(*) AS TOTAL
    FROM OFFER
    GROUP BY O_Quarter
);
-- Find total # of successful offers in each quarter
CREATE OR REPLACE VIEW quarterS AS (
    SELECT O_Quarter, COUNT(*) AS TOTAL
    FROM OFFER
    WHERE OFFER.ACCEPTED = 1
    GROUP BY O_Quarter
);
CREATE OR REPLACE VIEW quarterrRate AS (
    SELECT QS.TOTAL / QT.TOTAL
    FROM ((quarterTotal AS QT)
    INNER JOIN (quarterS AS QS)
    ON QS.O_Quarter)
);
```

```sql
-- Find total # of offers proposed for each offer window
CREATE OR REPLACE VIEW dateTotal AS (
    SELECT O.Deadline - O.Proposed_Date AS W, COUNT(*) AS TOTAL
    FROM OFFER AS O
    GROUP BY O.Deadline - O.Proposed_Date
);
-- Find total # of successful offers for each offer window
CREATE OR REPLACE VIEW dateS AS (
    SELECT O.Deadline - O.Proposed_Date AS W, COUNT(*) AS TOTAL
    FROM OFFER AS O
    WHERE OFFER.ACCEPTED = 1
    GROUP BY O.Deadline - O.Proposed_Date
);
CREATE OR REPLACE VIEW dateRate AS (
    SELECT DS.TOTAL / DT.TOTAL
    FROM ((dateTotal AS DT)
    INNER JOIN (dateS AS DS)
    ON DS.W)
);
```

Query 2: Offer Success Rate – Import Data

Synthetic Data - Table OFFER

```
num_offer = 800
regions = ['Asia', 'Africa', 'North America', 'South America', 'Europe', 'Australia']
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
for i in range(num_offer):
    new_offer = {
                'Offer_Id': i,
                'Region': np.random.choice(regions, 1, p=[0.6, 0.02, 0.2, 0.04, 0.1, 0.04])[0],
                'Employee_ID': random.randint(1, 30),
                'Customer_ID': random.randint(1, 20),
                'Accepted': np.random.choice([1, 0], 1, p=[0.85, 0.15])[0],
                'Quarter': np.random.choice(quarters, 1, p=[0.1, 0.2, 0.6, 0.1])[0]
               }
    proposedd = fake.date_between(start_date='-2y', end_date='today')
    difference = np.random.choice([3, 5, 7, 10], 1, p=[0.1, 0.1, 0.5, 0.3])[0]
    ddl = proposedd + timedelta(days= int(difference))
    new_offer['Proposed_Date']= proposedd.strftime("%d-%b-%Y")
    new_offer['Deadline']= ddl.strftime("%d-%b-%Y")

    OFFER = OFFER.append(new_offer,ignore_index = True)
DataFrame_to_MySQL(OFFER,"OFFER")
```

Query 2: Offer Success Rate – Import Data

Synthetic Data - Table OFFER

```
num_offer = 800
regions = ['Asia', 'Africa', 'North America', 'South America', 'Europe', 'Australia']
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
for i in range(num_offer):
    new_offer = {
                'Offer_Id': i,
                'Region': np.random.choice(regions, 1, p=[0.6, 0.02, 0.2, 0.04, 0.1, 0.04])[0],
                'Employee_ID': random.randint(1, 30),
                'Customer_ID': random.randint(1, 20),
                'Accepted': np.random.choice([1, 0], 1, p=[0.85, 0.15])[0],
                'Quarter': np.random.choice(quarters, 1, p=[0.1, 0.2, 0.6, 0.1])[0]
               }
    proposedd = fake.date_between(start_date='-2y', end_date='today')
    difference = np.random.choice([3, 5, 7, 10], 1, p=[0.1, 0.1, 0.5, 0.3])[0]
    ddl = proposedd + timedelta(days= int(difference))
    new_offer['Proposed_Date']= proposedd.strftime("%d-%b-%Y")
    new_offer['Deadline']= ddl.strftime("%d-%b-%Y")

    OFFER = OFFER.append(new_offer,ignore_index = True)
DataFrame_to_MySQL(OFFER,"OFFER")
```

Query 2: Offer Success Rate – Data Analysis

Features:

Region: customer based region (North America, Europe…)

Employee_Id: employee keeping track of this particular offer

Quarter: offer purposed in which quarter

Offer_window: time given to customer for considering this offer

Company_size: size of the customer's company

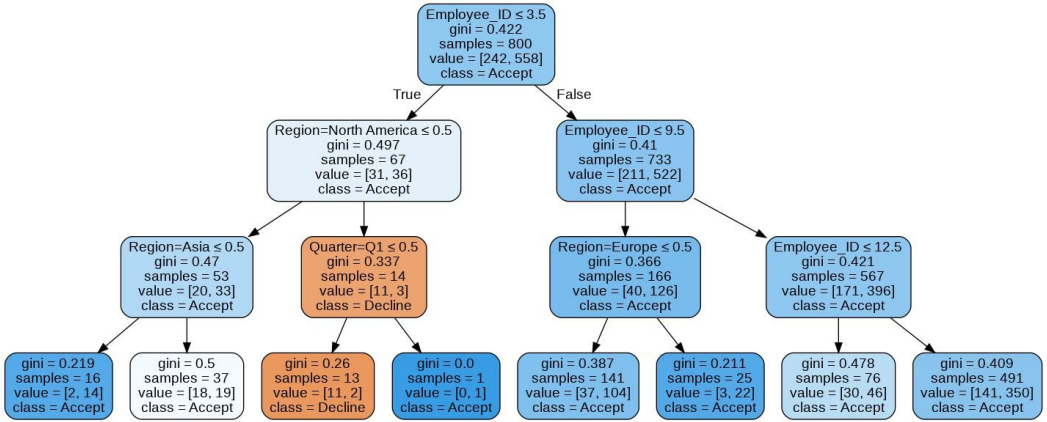
Method: Decision Tree with depth = 3

```
# vectorize label values
target = offer_df['Accepted']
le = preprocessing.LabelEncoder()
le.fit(target.unique())
target_dict = le.transform(target)


# vectorize features values
features = offer_df[['Region','Employee_ID','Quarter','company_Size','offer_length']]
vec = DictVectorizer()
features_dict = vec.fit_transform(features.to_dict(orient='records')).toarray()

# decision-tree classfiler
clf = DecisionTreeClassifier(max_depth=3)

clf = clf.fit(features_dict, target_dict)
y_pred = clf.predict(features_dict)

accuracy_score(target_dict, y_pred)
```

Query 2: Offer Success Rate – Data Analysis

Findings (Exploratory Data Analysis + Decision-tree Modeling)

1. Decision Tree model successfully predicts offer success 70.88% of the time
2. Offers that employees 29, 26, 14 are in charge of are more likely to be accepted
3. Offers purposed in Quarter 3 are more likely to be accepted
4. Customers from Asia are more likely to accept offers



Query 3: Recommended Manufacturers

Manufacturers are considered to be 'highly recommended' if they have a high 'manufacturer score', defined as follows:

Manufacturer score =

- ϕ_1 × average delay rate *(dependability, credibility, and timeliness)*
+ ϕ_2 × Number of types of products produced (*flexibility*)
+ ϕ_3 × Average customer rating *(quality)*
- ϕ_4 × Distance to the nearest port *(location)*

Rationale

- Allows manufacturers with a 'good' combination of time efficiency, quality, and other factors to be identified
- Allows the client to strategically encourage manufacturers to be timely, responsible, or produce higher quality products

```
SELECT M.Contact_contact_ID AS MID, -0.25 * avg(D.delay_Rate) + 0.15 * M.flexibility
+ 0.6 * AVG(D.customer_Satisfaction) + 0.1 * M.PORTDIST AS MSCORE
FROM DEAL AS D, Manufacturer_ORDER AS MO, Manufacturer as M
WHERE D.Manufacturer_Order_Order_order_ID = MO.Order_order_ID
AND MO.Manufacturer_Contact_contact_ID = M.Contact_contact_ID
GROUP BY M.Contact_contact_ID
ORDER BY MSCORE DESC;
```

Query 4: Employee Activities

For each employee, track their offer negotiation success rate, their appointments, and which batch inspections they were involved in.

Rationale

- Keep a record of Employee responsibilities and rate their performance
    - Bonus for employees who facilitated the most offers
- Keep a record of employee collaboration and facilitate employee cooperation

SQL

```sql
CREATE OR REPLACE VIEW E_SUCCESS_O AS (
    SELECT O.Employee_employee_Id AS EID, COUNT(*) AS NUM_SUCCESS
    FROM OFFER AS O
    WHERE O.is_accepted = TRUE
    GROUP BY O.Employee_employee_Id
);
CREATE OR REPLACE VIEW E_FAIL_O AS (
    SELECT O.Employee_employee_Id AS EID, COUNT(*) AS NUM_FAILURE
    FROM OFFER AS O
    WHERE O.is_accepted = FALSE
    GROUP BY O.Employee_employee_Id
);
CREATE OR REPLACE VIEW E_O_R AS (
SELECT S.EID AS EID, S.NUM_SUCCESS / (F.NUM_FAILURE + S.NUM_SUCCESS) AS OFFERSCORE
FROM E_SUCCESS_O AS S, E_FAIL_O AS F
WHERE S.EID = F.EID
);
CREATE OR REPLACE VIEW E_A AS (
    SELECT A.Employee_employee_Id AS EID, COUNT(*) AS NUM_APP
    FROM APPOINTMENT AS A
    GROUP BY A.Employee_employee_Id
);
CREATE OR REPLACE VIEW E_I AS (
    SELECT I.Employee_employee_Id AS EID, COUNT(*) AS NUM_INSPECTION
    FROM Inspection AS I
    GROUP BY I.Employee_employee_Id
);

SELECT E_O_R.EID AS EID, E_O_R.OFFERSCORE AS OFFERSCORE,
E_A.NUM_APP AS APPOINT_PERFORMANCE,
E_I.NUM_INSPECTION AS INSPECTION_PERFORMANCE
FROM E_O_R, E_A, E_I
WHERE E_O_R.EID = E_A.EID AND E_A.EID = E_I.EID
```

Query 5: Shipping Logistics

For each order, arrange the shipping route and logistics for better shipping and cost efficiency.
The idea is to find pairs of manufacturer orders with the same manufacturer and the same shipping destination (port) and compatible expected shipping dates so as to ship different orders together in an efficient manner.

Rationale

- Keep a record of orders with same shipping destination (port)
- Reduce shipping time and cost by shipping in groups
- Keep a record of shipping logistics for future shipping analysis
- Keep in mind shipping rates from different shipping companies
- Enhance shipping efficiency and improve system organization

```sql
SELECT MO1.Order_order_ID AS ORDER1, MO2.Order_order_ID AS ORDER2
FROM Batch AS B1, Batch AS B2, Shipment AS S1, Shipment AS S2, Manufacturer_Order AS MO1, Manufacturer_Order AS MO2
WHERE B1.Shipment_shipment_Id = S1.shipment_Id AND
B2.Shipment_shipment_Id = S2.shipment_Id AND
B1.Manufacturer_Order_Order_order_ID = MO1.Order_order_ID AND
B2.Manufacturer_Order_Order_order_ID = MO2.Order_order_ID AND
MO1.Order_order_ID < MO2.Order_order_ID AND
S1.ship_Date = S2.ship_Date;
```

Client Overview

Simplified EER

Relational Design

Database and Queries

Normalization

Normalization

8. EMPLOYEE (employee_Id, first_Name, last_Name, middle_Initials, personal_ID, department, salary, title, region_In_Charge)

This is in first and second normal form but not in third normal form because employee_ID determines personal_ID, and personal_ID also determines department, salary, title, region_in_charge, as well as names.

EMPLOYEE_PERSONAL_ID (employee_Id, personal_ID)

EMPLOYEE (employee_Id, first_Name, last_Name, middle_Initials, department, salary, title, region_In_Charge)

Now, everything is in third normal form since there are no transitive dependencies.

Normalization

13. APPOINTMENT (appointment_Id, date, appointment_Method, appointment_Feedback, employee_Id[8], contact_IDs[1a])

This is not in first normal form because appointment_feedback and contact_IDs are multi-valued attributes.

APPOINTMENT_FEEDBACK (appointment_Id, appointment_Feedback)

APPOINTMENT_CONTACT_IDS(appointment_Id, contact_ID[1a])

APPOINTMENT (appointment_Id, date, appointment_Method, employee_Id[8])

Now, everything is in third normal form since there are no transitive dependencies.

Normalization

16. INSPECTION (date, employee_Id[8], inspection fee, requirements, batch_Id, order_ID[3b])

This is not in first normal form because the requirements attribute is a multi-valued attribute.

INSPECTION_FEE (date, employee_Id[8], inspection fee)

INSPECTION_REQUIREMENTS (date, employee_Id[8], requirement)

INSPECTION_BATCHES (date, employee_Id, batch_Id)

INSPECTION_ORDERS(date, employee_Id, order_ID[3b])

Normalization

17. PURCHASE_ORDER_INCLUDES_PRODUCT (order_ID[3a], product_Ids[4], amount, price)

This is not in first normal form because product_IDs attribute is a multi-value attribute. Moreover, this is not in second normal form because amount attribute and price attribute are not fully functionally dependent on order_ID[3a] and product_Ids[4].

PURCHASE_ORDER_INCLUDES_PRODUCTS (order_ID[3a], product_Id[4])

PURCHASE_ORDER_INCLUDES_AMOUNT_PRICE (order_ID[3a] , amount, price)

Now, everything is in third normal form since there are no transitive dependencies.

Normalization

20. MANUFACTURER_ORDER_CONTAINS_PRODUCT (order_ID[3b], product_Ids[4], amount, price)

This is not in first normal form because product_Ids attribute is a multi-value attribute. In addition, this is also not in the second normal form because both amount and price are not fully functionally dependent on the two primary attributes order_ID[3b] and product_Ids[4].

MANUFACTURER_ORDER_CONTAINS_PRODUCT_IDS(order_ID[3b], product_Id[4])

MANUFACTURER_ORDER_CONTAINS_AMOUNT_PRICE(order_ID[3b], amount, price)

Now, this is in third normal form because there are no transitive dependencies.

Thank you for listening!