The Historic Bakery Shop Database Project

May, 11, 2023

Ciping Wu, Rupa Poddar, Wilson Cen Wu, Alessandro Sciorilli

The following materials document the design and development of a database application to support a local bakery shop. The project begins with a description of the business and proceeds through conceptual (ER) modeling, logical (Relational) modeling, Physical modeling and finally implementation of a database application. Notes (Commentary) are provided at the end of each section to explain some specific features of the steps being carried out.

I. Business Scenario

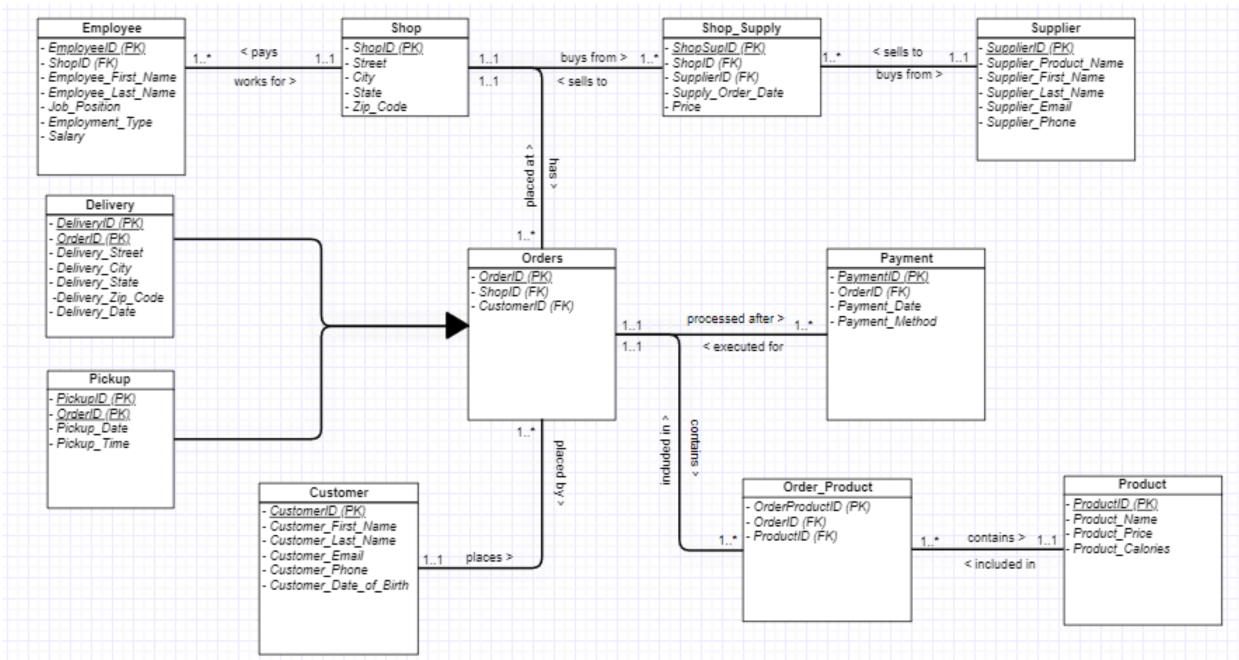
We are an historic local bakery shop in Manhattan, specializing in sweet pastries, breads and cakes. Despite our success and prime location, we have never mapped our operations and kept track of our customers. However, as we look to scale up our business, we have decided to build a database that will help us in several ways. Firstly, we can better keep track of our loyal customers and offer them rewards and discounts. Secondly, we can improve our operations management. And thirdly, we can gain better insights into our customers' favorite products.

Although there are expenses associated with building a database, we believe it is a sound investment as it will help us retain customers, improve operational efficiency, and increase our customer base by tailoring offers to the most popular products. Implementing a database is a crucial step in laying the foundation for our expansion.

As part of our operations, we work with suppliers who periodically deliver raw materials to us. Each day, we use these ingredients to create our products, which are specifically categorized by details such as retail price and calories. When customers place an order or visit one of our shops, we collect their data, including their email, phone number, and date of birth. This enables us to reach out to them with offers, news, and discounts and provide them with a special gift on their birthday. Once the payment is validated, we record the transaction details, such as the total amount spent and the date. We keep track of all order information, including whether it's for delivery or pick-up. For delivery orders, we also collect the delivery address, date and time, and the name of the rider. For pick-up orders, we only collect the pick-up date and time. Finally, we keep updated records with details of our employees.

II. ER Model using UML Notation

Based on the above description, we develop the following Entity Relationship model using UML notation.



Commentary:

- The ER diagram has relationship lines that are clearly drawn and do not overlap, resulting in a neat and organized layout.
- All attribute names are written without spaces or abbreviations, and no attributes have the same name.
- The primary key of each entity has been underlined in the diagram to make it easier to visualize.
- An **advanced feature**, an inheritance hierarchy, has been included in the ER diagram. The superclass "Orders" has been defined with two subclasses, "Pickup" and "Delivery".
- Having a many-to-many relationships between the entities “Shop” and “Supplier”, we needed to create an associative entity called “Shop Supply” to represent this relationship and allow multiplicity on both sides (a shop may have multiple suppliers, a supplier may serve multiple shops).
- Having a many-to-many relationships between the entities “Orders” and “Product”, we needed to create an associative entity called “Order_Product” to represent this relationship and allow multiplicity on both sides (an order may contain multiple products, and multiple orders may be included in the same type of products).

Relationship sentences:

- One **Shop** may have one or more **Employee**
- One **Employee** can work for one and only one **Shop**

- One **Shop** may have one or multiple **Supplier**
- One **Supplier** may sell to one or more **Shop**

- One **Shop** may have zero or more **Customer**
- One **Customer** may buy from one or more **Shop**

- One **Shop** may have one or multiple **Orders**
- One **Order** may be placed at one and only one **Shop**

- One **Customer** may place one or more **Orders**
- One **Order** must be placed by one and only one **Customer**

- One **Order** must be processed after one or multiple **Payment**
- One **Payment** must be executed for one or multiple **Orders**

- One **Order** may contain one or multiple **Product**
- One **Product** may be included in one or multiple **Orders**

III. Conversion to Relational Model

The next step is to Convert the Entity Relationship diagram to a Relational Model. During this step, Identifiers in the Entities become Keys in the Relations. One-to-Many relationships result in a foreign key being copied from the One side to the Many sides of the relationship.

- **Shop** (*ShopID (PK), Street, City, State, Zip_Code*)
- **Employee** (*EmployeeID (PK), ShopID (FK), Employee_First_Name, Employee_Last_Name, Job_Position, Employment_Type, Salary*)
- **Shop_Supply** (*ShopSupID (PK), ShopID (FK), SupplierID (FK), Supply_Order_Date, Price*)
- **Supplier** (*SupplierID (PK), Supplier_Product_Name, Supplier_First_Name, Supplier_Last_Name, Supplier_Email, Supplier_Phone*)
- **Customer** (*CustomerID (PK), Customer_First_Name, Customer_Last_Name, Customer_Email, Customer_Phone, Customer_Date_Of_Birth*)
- **Payment** *(PaymentID (PK), OrderID (FK), Payment_Date, Payment_Method)*
- **Orders** (*OrderID (PK), ShopID (FK), CustomerID (FK))*
- **Order_Product** *(OrderProductID (PK), OrderID (FK), ProductID (FK))*
- **Product** (*ProductID (PK), Product_Name, Product_Price, Product_Calories*)
- **Pickup** (*PickupID (PK), OrderID (PK), Pickup_Date, Pickup_Time*)
- **Delivery** (*DeliveryID (PK), OrderID (PK), Delivery_Street, Delivery_City, Delivery_State, Delivery_Zip_Code, Delivery_Date*)

Commentary:

- Notice that the entities "Pickup" and "Delivery" are two subclasses that depend on the superclass "Order," forming an inheritance hierarchy
- Primary keys are indicated with (PK) designation and foreign keys are indicated with the (FK) designation.

IV. Normalization

The next step is to Normalize the Relations.

1) **Shop** (ShopID (PK), Street, City, State, Zip_Code)

ShopID	Street	City	State	Zip_Code
S101	123 Main street	New York	NY	1001
S102	456 Broadway	New York	NY	1002
S103	789 Fifth Avenue	New York	NY	1003
S104	246 Park Avenue South	New York	NY	1004
S105	135 Grand Street	New York	NY	1005

Primary Key: ShopID
FD1: ShopID (PK) -> Street, City, State, Zip_Code
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

2) **Employee** (EmployeeID (PK), ShopID (FK), Employee_First_Name, Employee_Last_Name, Job_Position, Employment_Type, Salary)

EmployeeID	ShopID	Employee_First_Name	Employee_Last_Name	Job_Position	Employment_Type	Salary
E16	S101	Wilson	Foster	Baker	Full-Time	50000
E17	S101	Ryan	Peterson	Pastry Chef	Part-Time	55000
E18	S101	Alessandro	Evans	Baker	Internship	47500
E19	S101	Ethan	Brown	Cake Decorator	Part-Time	48000
E20	S101	Ciping	Parker	Assistant Baker	Full-Time	45000
E21	S102	Rupa	Poddar	Baker	Internship	35000
E22	S102	Liam	Turner	Store Manager	Full-Time	70000
E23	S102	Samuel	Johnson	Bread Baker	Volunteer	56000
E24	S102	Ava	Anderson	Pastry Chef	Part-Time	56000
E25	S102	Abigail	Smith	Cake Designer	Temporary	75000
E26	S103	Emily	Campbell	Pastry Chef	Full-Time	60000
E27	S103	Tracy	Hayes	Inventory Manager	Full-Time	67500
E28	S103	Daniel	Bennett	Cake Decorator	Volunteer	52000
E29	S103	Victoria	Carter	Inventory Manager	Temporary	65000
E30	S103	Ryon	Reed	Cake Decorator	Part-Time	43000

Primary Key: EmployeeID
FD1: EmployeeID (PK) -> ShopID (FK), Employee_First_Name, Employee_Last_Name, Job_Position, Employment_Type, Salary

1NF: Meets the definition of a relation
2NF: No partial Key dependencies

3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

3) **Shop_Supply** (ShopSupID (PK), ShopID (FK), SupplierID (FK), Supply_Order_Date, Price)

ShopSupID	ShopID	SupplierID	Supply_Order_Date	Price
1001	S101	S31	02-mar-23	3000
1002	S101	S32	03-mar-23	4000
1003	S101	S33	02-mar-23	2500
1004	S101	S34	03-mar-23	1500
1005	S101	S42	02-mar-23	3500
1006	S101	S36	03-mar-23	2500
1007	S102	S37	02-mar-23	2700
1008	S102	S38	03-mar-23	2800
1009	S102	S39	02-mar-23	2450
1010	S102	S40	11-mar-23	3465
1011	S102	S31	11-mar-23	1750
1012	S102	S41	02-mar-23	1950
1013	S103	S41	03-mar-23	2000
1014	S103	S42	15-mar-23	2100
1015	S103	S39	16-mar-23	2250
1016	S103	S42	09-mar-23	3000
1017	S103	S40	09-mar-23	3100
1018	S103	S38	09-mar-23	1900

Primary Key: ShopSupID
FD1: ShopSupID (PK) -> ShopID (FK), SupplierID (FK), Supply_Order_Date, Price
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

4) **Supplier** (SupplierID (PK), Supplier_Product_Name, Supplier_First_Name, Supplier_Last_Name, Supplier_Email, Supplier_Phone)

SupplierID	Supply_Product_Name	Supplier_First_Name	Supplier_Last_Name	Supplier_Email	Supplier_Phone
S31	Flour A	Andy	Sarmiento	andy.sarmiento@gmail.com	(678) 513-4372
S32	Sugar A	Cecilia	Kim	ceciliakim@yahoo.com	(407) 692-8109
S33	Butter A	Pranto	Saha	pranto.saha@hotmail.com	(817) 874-5736
S34	Eggs A	Sheikh	Maglaya	sheikhmaglaya@gmail.com	(646) 912-4658
S35	Flour B	Amaka	Arroyo	amakaarroyo@hotmail.com	(901) 803-6849
S36	Sugar B	Ankita	Gabe	ankita.gabe@yahoo.com	(305) 625-0427
S37	Butter B	Nancy	Wei	nancy.wei@yahoo.com	(312) 731-9195
S38	Eggs B	Letty	Batra	letty.batra@gmail.com	(818) 874-9543
S39	Salt A	Jason	Yang	jasonyang@yahoo.com	(480) 539-8692
S40	Salt B	Kal	Galindo	kalgalindo@hotmail.com	(720) 876-3509
S41	Blueberries	Anirban	Bose	anirbanbose@gmail.com	(703) 561-7990
S42	Cinnamon	Abhi	Bairu	abhi.bairu@yahoo.com	(918) 965-8420

Primary Key: SupplierID
FD1: SupplierID (PK) -> Supplier_Product_Name, Supplier_First_Name, Supplier_Last_Name, Supplier_Email, Supplier_Phone
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

5) **Customer** (CustomerID (PK), Customer_First_Name, Customer_Last_Name, Customer_Email, Customer_Phone, Customer_Date_Of_Birth)

CustomerID	Customer_First_Name	Customer_Last_Name	Customer_Email	Customer_Phone	Customer_Date_of_Birth
C01	John	Smith	johnsmith@gmail.com	(555) 123-4567	30/08/1997
C02	Sarah	Johnson	sarah.johnson@hotmail.com	(555) 234-5678	10/07/1990
C03	Micahel	Williams	michael.williams@yahoo.com	(555) 345-6789	03/11/1988
C04	Jessica	Brown	jessicabrown@gmail.com	(555) 456-7890	22/04/1992
C05	David	Davis	david.davis@hotmail.com	(555) 567-8901	17/09/1987
C06	Jennifer	Rodriguez	jennifer.rodriguez@yahoo.com	(555) 678-9012	08/12/1993
C07	Christopher	Martinez	christophermartinez@gmail.com	(555) 789-0123	25/05/1989
C08	Lisa	Wilson	lisa.wilson@hotmail.com	(555) 890-1234	14/10/1991

Primary Key: CustomerID
FD1: CustomerID (PK) -> Customer_First_Name, Customer_Last_Name, Customer_Email, Customer_Phone, Customer_Date_Of_Birth

1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

6) **Payment** (PaymentID (PK), OrderID (FK), Payment_Date, Payment_Method)

PaymentID	OrderID	Payment_Date	Payment_Method
P501	O5850	05/04/2023	Credit Card
P502	O5851	06/04/2023	PayPal
P503	O5852	07/04/2023	Credit Card
P504	O5853	06/04/2023	Cash
P505	O5854	05/04/2023	PayPal
P506	O5855	07/04/2023	Credit Card
P507	O5856	11/04/2023	Cash
P508	O5857	12/04/2023	Debit Card
P509	O5858	11/04/2023	Debit Card
P510	O5859	12/04/2023	Cash
P511	O5860	15/04/2023	PayPal
P512	O5861	16/04/2023	Credit Card
P513	O5862	15/04/2023	PayPal
P514	O5863	16/04/2023	Credit Card
P515	O5863	16/04/2023	Cash
P516	O5864	16/04/2023	Credit Card

Primary Key: PaymentID
FD1: PaymentID (PK) -> OrderID (FK), Payment_Date, Payment_Method
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

7) **Orders** (OrderID (PK), ShopID (FK), CustomerID (FK)

OrderID	ShopID	CustomerID
O5850	S101	C01
O5851	S101	C02
O5852	S101	C03
O5853	S101	C04
O5854	S101	C05
O5855	S102	C06

O5856	S102	C07
O5857	S102	C08
O5858	S102	C01
O5859	S102	C02
O5860	S103	C04
O5861	S103	C08
O5862	S103	C06
O5863	S103	C05
O5864	S103	C04

Primary Key: OrderID
FD1: OrderID (PK) -> ShopID (FK), CustomerID (FK)
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

8) Order_Product (OrderProductID (PK), OrderID (FK), ProductID (FK))

OrderProductID	OrderID	ProductID
OP001	O5850	P61
OP002	O5850	P63
OP003	O5850	P65
OP004	O5851	P68
OP005	O5851	P70
OP006	O5852	P64
OP007	O5853	P62
OP008	O5853	P69
OP009	O5853	P65
OP010	O5854	P66
OP011	O5854	P75
OP012	O5855	P64
OP013	O5855	P74
OP014	O5856	P64
OP015	O5857	P61
OP016	O5857	P73
OP017	O5857	P68
OP018	O5858	P62
OP019	O5858	P69
OP020	O5859	P72
OP021	O5859	P68

OP022	O5860	P61
OP023	O5861	P71
OP024	O5863	P68
OP025	O5864	P65

Primary Key: OrderProductID
FD1: OrderProductID (PK) -> OrderID (FK), ProductID (FK)
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

9) **Product** (ProductID (PK), Product_Name, Product_Price, Product_Calories)

ProductID	Product_Name	Product_Price	Product_Calories
P61	Croissant	2	231
P62	Sourdough Bread	5	110
P63	Cinnamon Rolls	3	310
P64	Fruit Danish	2	265
P65	Baguette	4	120
P66	Blueberry Muffin	2	385
P67	Apple Pie	16	320
P68	Éclair	4	220
P69	Banana Bread	7	290
P70	Red Velvet Cupcake	3	240
P71	Chocolate Chip Cookies	1	150
P72	Cheese Straws	4	120
P73	Pecan Tart	8	460
P74	Lemon Bars	3	210
P75	Carrot Cake	19	400

Primary Key: ProductID
FD1: ProductID (PK) -> Product_Name, Product_Price, Product_Calories
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

10) **Pickup** (PickupID (PK), OrderID (PK), Pickup_Date, Pickup_Time)

PickupID	OrderID	Pickup_Date	Pickup_Time
P5698	O5850	05/04/2023	8:30:00
P5699	O5851	05/04/2023	18:15:00
P5700	O5852	06/04/2023	8:30:00
P5701	O5853	06/04/2023	8:30:00
P5702	O5854	05/04/2023	8:30:00
P5703	O5855	06/04/2023	16:15:00
P5704	O5856	11/04/2023	8:30:00
P5705	O5857	11/04/2023	18:15:00

Primary Key: PickupID, OrderID
FD1: PickupID (PK), OrderID (PK) -> Pickup_Date, Pickup_Time
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: No Transitive dependencies
BCNF: All determinants are candidate keys

11) **Delivery** (DeliveryID (Key), OrderID (Key), Delivery_Street, Delivery_City, Delivery_State, Delivery_Zip_Code, Delivery_Date)

DeliveryID	OrderID	Delivery_Street	Delivery_City	Delivery_State	Delivery_Zip_Code	Delivery_Date
D267	O5858	3118 Broadway	New York	NY	10027	07/04/2023
D268	O5859	174 East 75th Street	New York	NY	10021	07/04/2023
D269	O5860	3118 Broadway	New York	NY	10027	07/04/2023
D270	O5861	3118 Broadway	New York	NY	10027	07/04/2023
D271	O5862	174 East 75th Street	New York	NY	10021	07/04/2023
D272	O5863	3118 Broadway	New York	NY	10027	20/04/2023
D273	O5864	174 East 75th Street	New York	NY	10021	07/04/2023

Primary Key: DeliveryID, OrderID
FD1: DeliveryID (PK), OrderID (PK) -> Delivery_Street, Delivery_City, Delivery_State, Delivery_Zip_Code, Delivery_Date
1NF: Meets the definition of a relation
2NF: No partial Key dependencies
3NF: All determinants are candidate keys

V. Structured Query Language (SQL) to Create the Schema

Create a table in the database for each of the relations in the final set of relations.

The following SQL code creates the eleven tables and adds the PRIMARY KEY and FOREIGN KEY constraint to each one:

```
CREATE TABLE SHOP (
ShopID Char(35) NOT NULL,
Street Char(35) NOT NULL,
City Char(35) NOT NULL,
State Char(35) NOT NULL,
Zip_Code INT NOT NULL,
CONSTRAINT SHOP_PK PRIMARY KEY(ShopID)
);
```

```
CREATE TABLE EMPLOYEE (
EmployeeID Char(35) NOT NULL,
ShopID Char(35) NOT NULL,
Employee_First_Name Char(35) NOT NULL,
Employee_Last_Name Char(35) NOT NULL,
Job_Position Char(35) NOT NULL,
Employment_Type Char(35) NOT NULL,
Salary numeric NOT NULL,
CONSTRAINT EMPLOYEE_PK PRIMARY KEY(EmployeeID),
CONSTRAINT EMP_SHOP_FK FOREIGN KEY(ShopID)
REFERENCES SHOP(ShopID)
ON DELETE NO ACTION ON UPDATE CASCADE
);
```

```
CREATE TABLE SHOP_SUPPLY (
ShopSupID Char(35) NOT NULL,
ShopID Char(35) NOT NULL,
SupplierID Char(35) NOT NULL,
Supply_Order_Date Date NOT NULL,
Price numeric NOT NULL,
CONSTRAINT SHOP_SUPPLY_PK PRIMARY KEY(ShopSupID),
CONSTRAINT SHOPSUPPLY_SHOP_FK FOREIGN KEY(ShopID)
REFERENCES SHOP(ShopID)
ON DELETE NO ACTION ON UPDATE CASCADE,
CONSTRAINT SHOPSUPPLY_SUPPLIER_FK FOREIGN KEY(SupplierID)
```

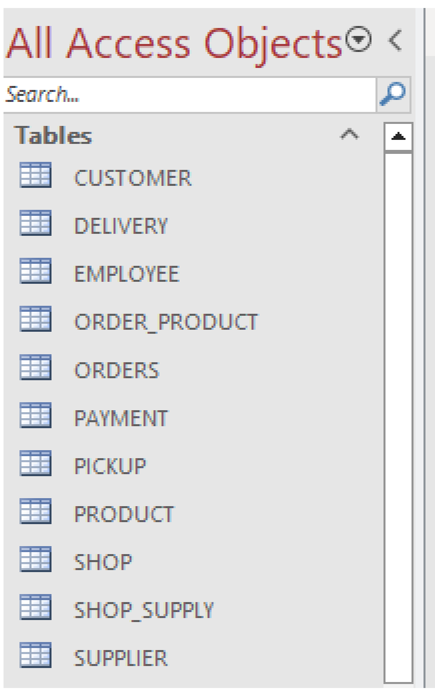
```
REFERENCES SUPPLIER(SupplierID)
ON DELETE NO ACTION ON UPDATE CASCADE
);



CREATE TABLE SUPPLIER (
SupplierID Char(35) NOT NULL,
Supply_Product_Name Char(35) NOT NULL,
Supplier_First_Name Char(35) NOT NULL,
Supplier_Last_Name Char(35) NOT NULL,
Supplier_Email Char(35) UNIQUE NOT NULL,
Supplier_Phone Char(35) NOT NULL,
CONSTRAINT SUPPLIER_PK PRIMARY KEY(SupplierID)
);



CREATE TABLE CUSTOMER (
CustomerID Char(35) NOT NULL,
Customer_First_Name Char(35) NOT NULL,
Customer_Last_Name Char(35) NOT NULL,
Customer_Email Char(35) NOT NULL,
Customer_Phone Char(35) NOT NULL,
Customer_Date_of_Birth Date NOT NULL,
CONSTRAINT CUSTOMER_PK PRIMARY KEY(CustomerID)
);


CREATE TABLE PAYMENT (
PaymentID Char(35) NOT NULL,
OrderID char(35) NOT NULL,
Payment_Date Date NOT NULL,
Payment_Method Char(35) NOT NULL,
CONSTRAINT PAYMENT_PK PRIMARY KEY(PaymentID),
CONSTRAINT PAY_ORDER_FK FOREIGN KEY(OrderID)
REFERENCES ORDERS(OrderID)
ON DELETE NO ACTION ON UPDATE CASCADE
);


CREATE TABLE ORDERS(
OrderID Char(35) NOT NULL,
ShopID Char(35) NOT NULL,
CustomerID Char(35) NOT NULL,
CONSTRAINT ORDER_PK PRIMARY KEY(OrderID),
CONSTRAINT ORD_SHOP_FK FOREIGN KEY(ShopID)
```

```
REFERENCES SHOP(ShopID)
ON DELETE NO ACTION ON UPDATE CASCADE,
CONSTRAINT ORD_CUSTOMER_FK FOREIGN KEY(CustomerID)
REFERENCES CUSTOMER(CustomerID)
ON DELETE NO ACTION ON UPDATE CASCADE
);


CREATE TABLE PRODUCT (
ProductID Char(35) NOT NULL,
Product_Name Char(35) NOT NULL,
Product_Price NUMERIC(10) NOT NULL,
Product_Calories INT NOT NULL,
CONSTRAINT PRODUCT_PK PRIMARY KEY(ProductID)
);


CREATE TABLE PICKUP (
PickupID Char(35) NOT NULL,
OrderID Char(35) NOT NULL,
Pickup_Date Date NOT NULL,
Pickup_Time Time NOT NULL,
CONSTRAINT PICKUP_PK PRIMARY KEY(PickupID, OrderID),
CONSTRAINT PICKUP_ORDER_FK FOREIGN KEY(OrderID)
REFERENCES ORDERS(OrderID)
ON DELETE NO ACTION ON UPDATE CASCADE
);


CREATE TABLE DELIVERY (
DeliveryID Char(35) NOT NULL,
OrderID Char(35) NOT NULL,
Delivery_Street Char(35) NOT NULL,
Delivery_City Char(35) NOT NULL,
Delivery_State Char(35) NOT NULL,
Delivery_Zip_Code INT NOT NULL,
Delivery_Date Date NOT NULL,
CONSTRAINT DELIVERY_PK PRIMARY KEY(DeliveryID, OrderID),
CONSTRAINT DELI_ORDER_FK FOREIGN KEY(OrderID)
REFERENCES ORDERS(OrderID)
ON DELETE NO ACTION ON UPDATE CASCADE
);
```

Commentary on SQL:

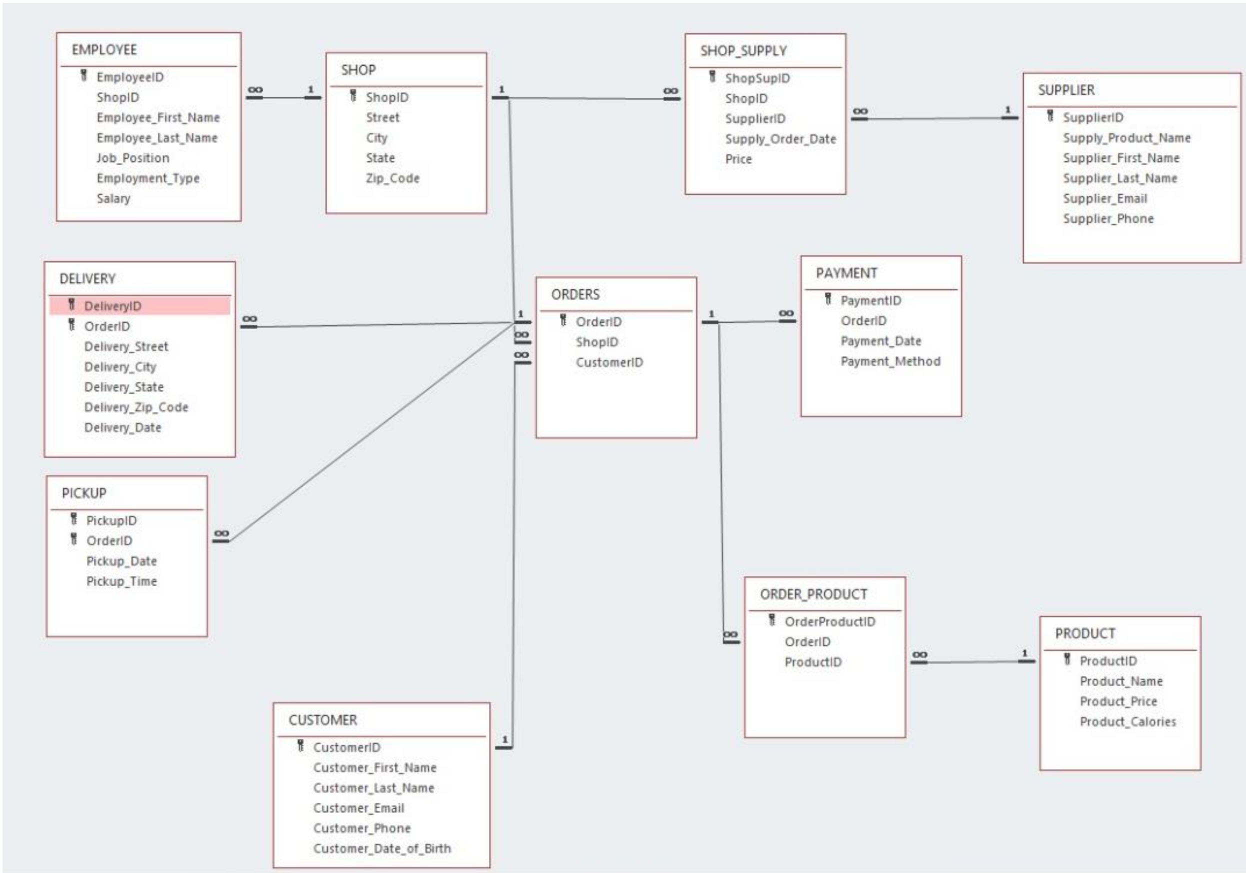
- Constraints are given meaningful names such as EMPLOYEE_PK for a Primary Key and EMP_SHOP_FK for a Foreign Key.
- Supplier_Email in Supplier is a candidate key/alternate key.
- Keys and Foreign Keys should have the same exact name and data type. For example, the Key OrderID is CHAR(35) in the Orders table and also CHAR(35) in the Payment table, Pickup Table and Delivery Table.
- DDL statements include Referential integrity for FKs.
- If a NUMBER or INTEGER data type is used, then the leading zeros will be missing.

After creating the tables, the database schema now looks like the following:



Relationship View

Using the Relationship View under Database Tools, we can see the relationships (Foreign Keys) between the tables:



Example of adding Data to the Tables using SQL INSERT Statements:

```
INSERT INTO SHOP (ShopID, Street, City, State, Zip_Code)
VALUES ('S105', '135 Grand Street', 'New York', 'NY', '10005');
```

```
INSERT INTO SHOP (ShopID, Street, City, State, Zip_Code)
VALUES ('S105', '135 Grand Street', 'New York', 'NY', '10005');
```

(…)

Commentary on Data Samples:

- We add just enough data to be able to test out the relationships between the tables and to give the application developers something to work with.
- At this point the database schema is ready for the application developers to get to work designing forms, reports and queries.

IV. Creation of DML Queries

After creating our database, we recognized its potential as a powerful tool for analyzing our business and making data-driven decisions. To fully leverage this potential, we developed a list of DML (Data Manipulation Language) statements that enable us to extract meaningful insights from our data. These DML statements allow us to manipulate and query the data in various ways, enabling us to answer important business questions and make informed decisions. By tailoring our products and services to better meet the needs and expectations of our customers, we can ultimately increase their satisfaction and loyalty. Using this set of queries to analyze our database, we are able to transform raw data into actionable insights that allow us to optimize our business operations and make better decisions. This, in turn, helps us to stay competitive in our industry and achieve long-term success.

1) AverageSalaryEmployee

```
SELECT EMPLOYEE.Employment_Type, EMPLOYEE.Job_Position,
Avg(EMPLOYEE.Salary) AS AvergeSalary
FROM EMPLOYEE
GROUP BY EMPLOYEE.Employment_Type, EMPLOYEE.Job_Position;
```

AverageSalaryEmployee

Employmen	Job_Position	AvergeSalary
Full-Time	Assistant Baker	45000
Full-Time	Baker	50000
Full-Time	Bread Baker	56000
Full-Time	Cake Decorator	52000
Full-Time	Inventory Manager	67500
Full-Time	Pastry Chef	60000
Full-Time	Store Manager	70000
Internship	Baker	41250
Part-Time	Cake Decorator	45500
Part-Time	Pastry Chef	55500
Temporary	Cake Designer	75000
Temporary	Inventory Manager	65000

2) CalculateCostPerOrder-Join

```
SELECT OrderID, SUM(Product_Price) AS Cost
FROM ORDER_PRODUCT, PRODUCT
WHERE ORDER_PRODUCT.ProductID = PRODUCT.ProductID
GROUP BY ORDER_PRODUCT.OrderID;
```

CalculateCostPerOrder-Join

OrderID	Cost
O5850	9
O5851	7
O5852	2
O5853	16
O5854	21
O5855	5
O5856	2
O5857	14
O5858	12
O5859	8
O5860	2
O5861	1
O5863	4
O5864	4

3) FilerLowCaloriesProducts

```
SELECT *
FROM PRODUCT
WHERE (((PRODUCT.Product_Calories)<250));
```

FilterLowCaloriesProduct

ProductID	Product_Name	Product_Price	Product_Cal
P61	Croissant	2	231
P62	Sourdough Bread	5	110
P65	Baguette	4	120
P68	Éclair	4	220
P70	Red Velvet Cupcake	3	240
P71	Chocolate Chip Cookies	1	150
P72	Cheese Straws	4	120
P74	Lemon Bars	3	210
*			

4) MoneyEarned-Suppliers-Join

```
SELECT SUPPLIER.SupplierID, Sum(SHOP_SUPPLY.Price) AS SumOfPrice
FROM SUPPLIER, SHOP_SUPPLY
WHERE SUPPLIER.SupplierID = SHOP_SUPPLY.SupplierID
GROUP BY SUPPLIER.SupplierID;
```

MoneyEarned-Suppliers-join

SupplierID	SumOfPrice
S31	4750
S32	4000
S33	2500
S34	1500
S36	2500
S37	2700
S38	4700
S39	4700
S40	6565
S41	3950
S42	8600

5) PayMethodPreferred-DescOrder

```
SELECT PAYMENT.Payment_Method, Count(PAYMENT.Payment_Method) AS
CountOfPayment_Method
FROM PAYMENT
GROUP BY PAYMENT.Payment_Method
ORDER BY Count(PAYMENT.Payment_Method) DESC;
```

PayMethodPreferred-DescOrder

Payment_Method	CountOfPayment_Method
Credit Card	6
PayPal	4
Cash	4
Debit Card	2

6) **Popular_Products-join**

```
SELECT Product_Name, Count(ORDER_PRODUCT.ProductID) AS
NumberOfProducts
FROM PRODUCT, ORDER_PRODUCT
WHERE PRODUCT.ProductID = ORDER_PRODUCT.ProductID
GROUP BY PRODUCT.Product_Name;
```

Popular_Products-join

Product_Name	NumberOfP
Baguette	3
Banana Bread	2
Blueberry Muffin	1
Carrot Cake	1
Cheese Straws	1
Chocolate Chip Cookies	1
Cinnamon Rolls	1
Croissant	3
Éclair	4
Fruit Danish	3
Lemon Bars	1
Pecan Tart	1
Red Velvet Cupcake	1
Sourdough Bread	2

7) SalesAmountPerShop= SalesAmountPerShop-Join4Tables + subquery

```
SELECT SHOP.ShopID, SUM(c.Product_Price) AS SalesAmount
FROM (SELECT * FROM (SELECT * FROM (SELECT * FROM (SELECT * FROM
ORDERS INNER JOIN SHOP ON SHOP.ShopID = ORDERS.ShopID)  AS a INNER
JOIN ORDER_PRODUCT ON ORDER_PRODUCT.OrderID = a.OrderID)  AS b INNER
JOIN PRODUCT ON PRODUCT.ProductID = b.ProductID)  AS c)  AS
SalesAmountPerShop
GROUP BY SHOP.ShopID;
```

SalesAmountPerShop-Join4Tables+subquery

ShopID	SalesAmount
S101	55
S102	41
S103	11

8) TotalAmountCustomerPurchased-Join4Tables+subquery

```
SELECT ORDERS.CustomerID, SUM(g.Product_Price) AS TotalCost
FROM (SELECT *
FROM(SELECT *
FROM (SELECT *
FROM
(SELECT * FROM CUSTOMER INNER JOIN ORDERS ON CUSTOMER.CustomerID =
```

```
ORDERS.CustomerID) AS e
INNER JOIN ORDER_PRODUCT ON ORDER_PRODUCT.OrderID = e.OrderID) AS f
INNER JOIN PRODUCT ON PRODUCT.ProductID = f.ProductID) AS g)  AS
[TotalAmountCustomer]
GROUP BY ORDERS.CustomerID;
```

TotalAmountCustomerPurchased-join4tables+subquery

CustomerID	TotalCost
C01	21
C02	15
C03	2
C04	22
C05	25
C06	5
C07	2
C08	15

9) TotalCaloriesPerOrder-Join

```
SELECT ORDER_PRODUCT.OrderID, SUM(Product_Calories) AS TotalCalories
FROM PRODUCT, ORDER_PRODUCT
WHERE PRODUCT.ProductID = ORDER_PRODUCT.ProductID
GROUP BY ORDER_PRODUCT.OrderID;
```

TotalCaloriesPerOrder-join

OrderID	TotalCalories
O5850	661
O5851	460
O5852	265
O5853	520
O5854	785
O5855	475
O5856	265
O5857	911
O5858	400
O5859	340
O5860	231
O5861	150
O5863	220
O5864	120

10) VIPCustomer-join+subquery

```
SELECT CustomerID, NewTable.TotalCost AS VIP
FROM (SELECT ORDERS.CustomerID, SUM(g.Product_Price) AS TotalCost
FROM
(SELECT *
FROM(SELECT *
FROM(SELECT *
FROM
(SELECT * FROM CUSTOMER INNER JOIN ORDERS ON CUSTOMER.CustomerID =
ORDERS.CustomerID) AS e
INNER JOIN ORDER_PRODUCT ON ORDER_PRODUCT.OrderID = e.OrderID) AS f
INNER JOIN PRODUCT ON PRODUCT.ProductID = f.ProductID) AS g) AS
newTable

GROUP BY ORDERS.CustomerID) AS NewTable
WHERE (NewTable.TotalCost>20);
```

VIPCustomer-join+subquery

CustomerID	VIP
C01	21
C04	22
C05	25

11) FilterSalaryAboveAvg(Salary)-subquery

```
SELECT EMPLOYEE.EmployeeID, EMPLOYEE.Employee_First_Name,
EMPLOYEE.Employee_Last_Name, EMPLOYEE.Salary
FROM EMPLOYEE
WHERE (((EMPLOYEE.[Salary])>(SELECT Avg(Salary) FROM EMPLOYEE)));
```

FilterSalaryAboveAvg(Salary)-subquery

EmployeeID	Employee_First_Name	Employee_Last_Name	Salary
E25	Abigail	Smith	75000
E26	Emily	Campbell	60000
E27	Tracy	Hayes	67500
E29	Victoria	Carter	65000
E22	Liam	Turner	70000
E23	Samuel	Johnson	56000
E24	Ava	Anderson	56000

12) 5%OffPriceAbove15 – subquery+join

Our objective with this query is to retrieve informati

```
SELECT OrderID, Cost, (0.95*Cost) AS Discounted
FROM (SELECT OrderID, SUM(Product_Price) AS Cost FROM ORDER_PRODUCT,
PRODUCT WHERE ORDER_PRODUCT.ProductID = PRODUCT.ProductID GROUP BY
ORDER_PRODUCT.OrderID) AS TABLE1
WHERE (Cost > 15);
```

5%OffPriceAbove15-subquery+join

OrderID	Cost	Discounted
O5853	16	15,2
O5854	21	19,95