DATABASE PROJECT

For Starbucks

TEAM 4
KEFU ZHU
ZEYANG GONG
RUIBING JI
HAILING LI

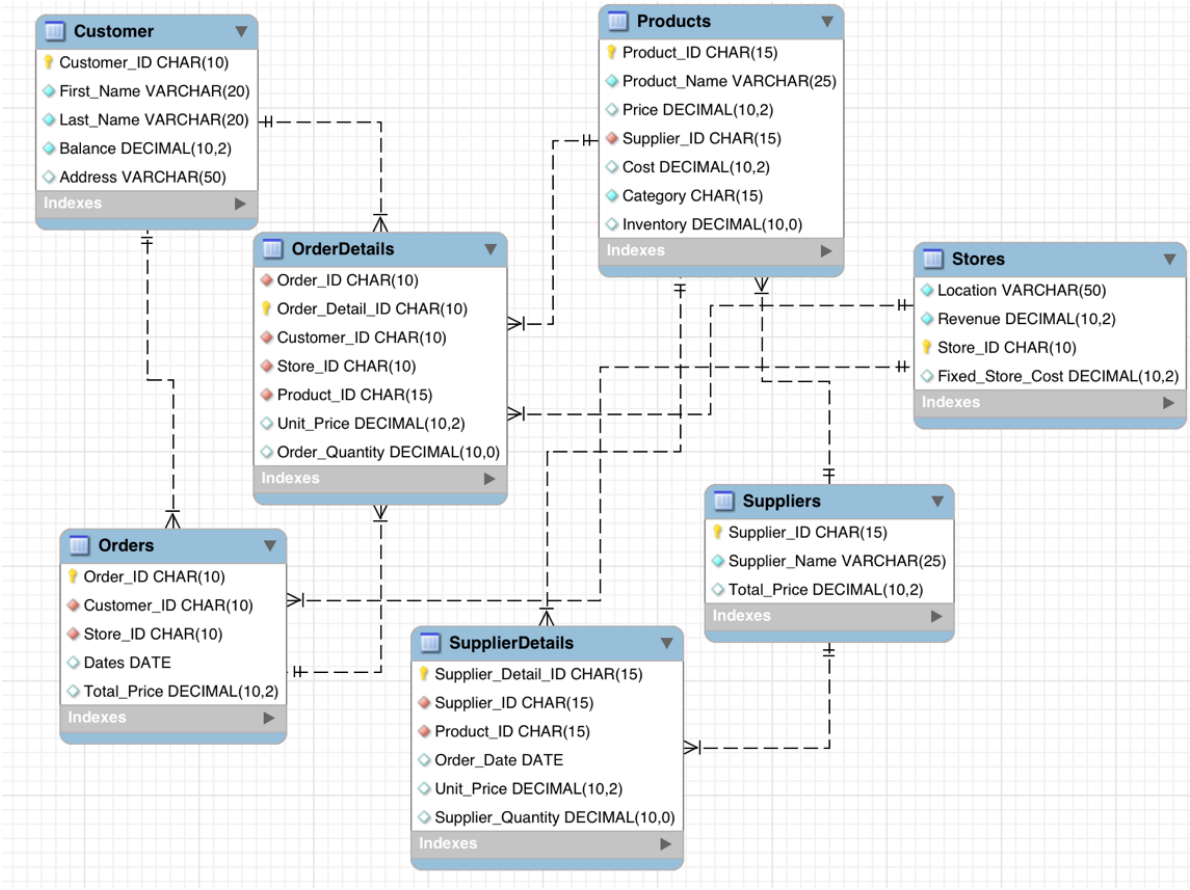
1. Introduction

Starbucks Corporation is an American coffee company and coffeehouse chain. The chain was founded in Seattle, Washington in 1971. It is the most successful coffeehouse brand which operates 23,450 stores all around the world. In addition to these convenient locations, Starbucks also provide its customers with a large product portfolio from various kinds of drinks, pre-packaged food items to mugs and drinkwares.

With its fast expansion, Starbucks has to balance the operating cost and monitor the revenues to keep its market leader position. We have built a mock Starbucks database with five dimensions: stores, suppliers, customer, products, and order information. Our database will enable Starbucks to retrieve data to answer business related queries. For instance, Starbucks can use data from Supplier, Products, and Stores tables to calculate the total cost of each store. They can also retrieve data from Orders table to see the total revenue for each store. Combining revenue and cost information, Starbucks can decide which stores are high performers and which are not. With the help of this database, Starbucks can better allocate its resources and make informative business decisions as well as map out effective marketing strategies.

2. Database Design

Figure 1



We built 7 tables in the database. The tables are Customer, Products, Stores, Orders, OrderDetails, Suppliers, and SupplierDetails.

3. Table Schemes

- Customer Table

Figure 2 (Customer Table Field View)

Customer_ID	First_Name	Last_Name	Balance	Address
001	Jason	Braharam	20.15	50 Bratton street
002	Sally	Brown	76.35	39 Hallway Blvd
003	John	Ibrahim	15.10	356 Newark Avenue
004	Kyle	Costner	0.75	1993 Michigan Avenue
005	Tom	Hanks	750.65	738 J.J Drive
999	empty	empty	0.00	empty
NULL	NULL	NULL	NULL	NULL

Figure 3 (Customer Table Setting)

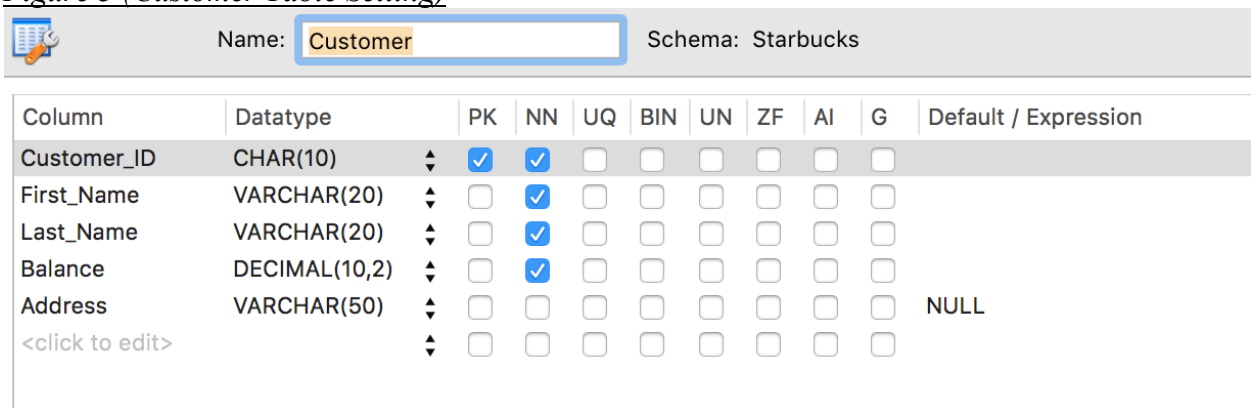
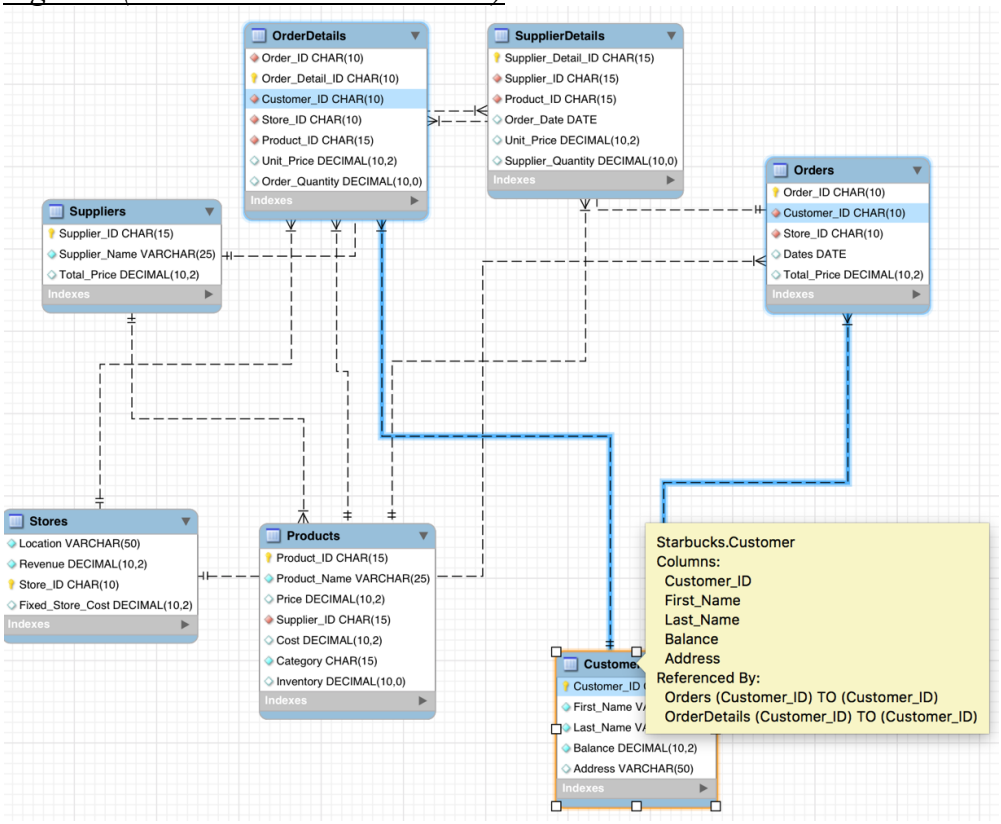


Figure 4 (Customer Table Connection)



We stored customers' information in the Customer Table. Each customer has only one CustomerID. We can observe each customer's CustomerID, first name, last name, balances in the membership cards and address. For instance, Sally Brown's CustomerID is 002 and has $76.35 in the card and he or she lives in 39 Hallway Blvd. For customers without a Starbucks membership, their ID is designated as 999.

Primary key: Customer_ID

- Orders Table

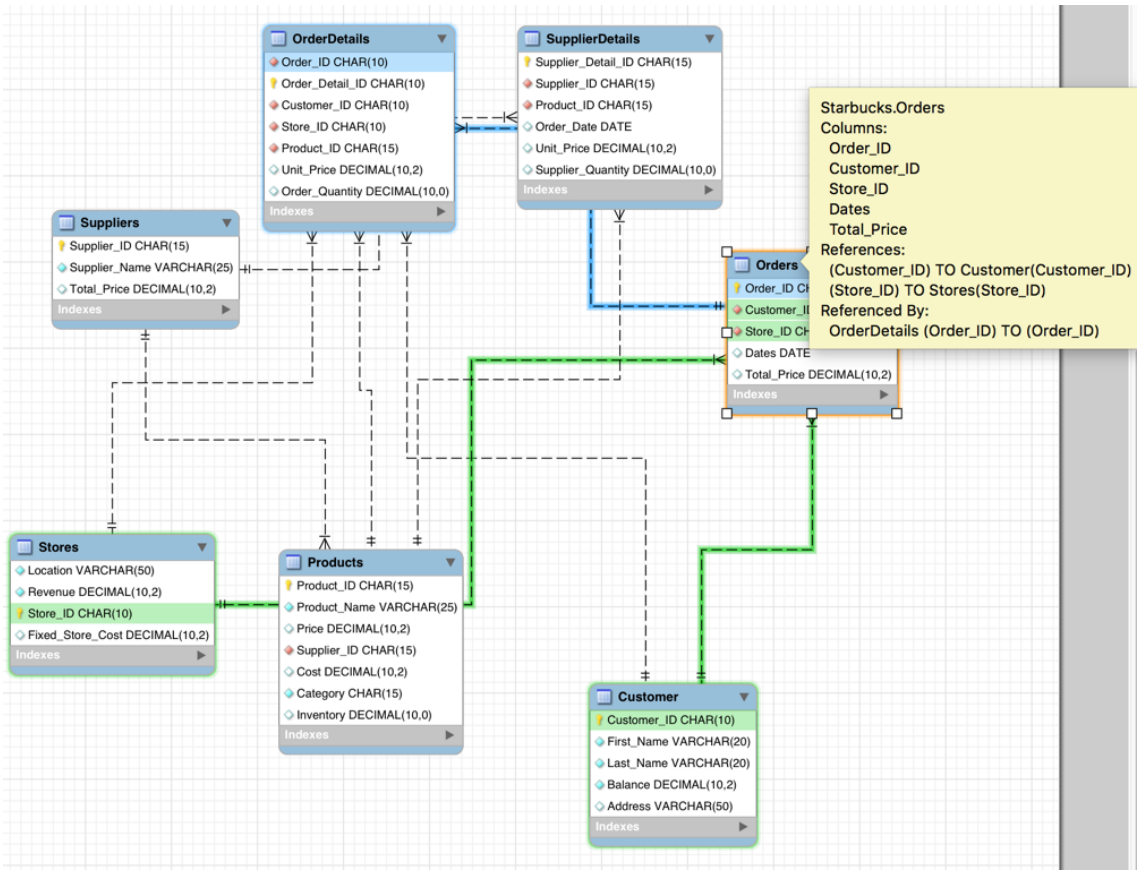
Figure 5 (Orders Table Field View)

Order_ID	Customer_ID	Store_ID	Dates	Total_Price
0001	001	ST1	2015-04-01	9.90
0002	002	ST2	2015-05-02	7.8
0003	999	ST3	2015-06-03	13.35
NULL	NULL	NULL	NULL	NULL

Figure 6 (Orders Table Setting)

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
Order_ID	CHAR(10)	☑	☑	☐	☐	☐	☐	☐	☐	
Customer_ID	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
Store_ID	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
Dates	DATE	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Total_Price	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 7 (Orders Table Connection)



We have orders' information in the Orders Table. OrderID is the primary key in this table. We can see each order's OrderID, CustomerID, StoreID, dates, and total Price. For instance, the Order 0001 is sold in the Store ST1 at April 1st, 2015. The total price is $9.9 and is placed by Customer 001.

Primary key: Order_ID

Foreign keys: Customer_ID (references Customer table)
Store_ID (references Stores table)

- OrderDetails Table

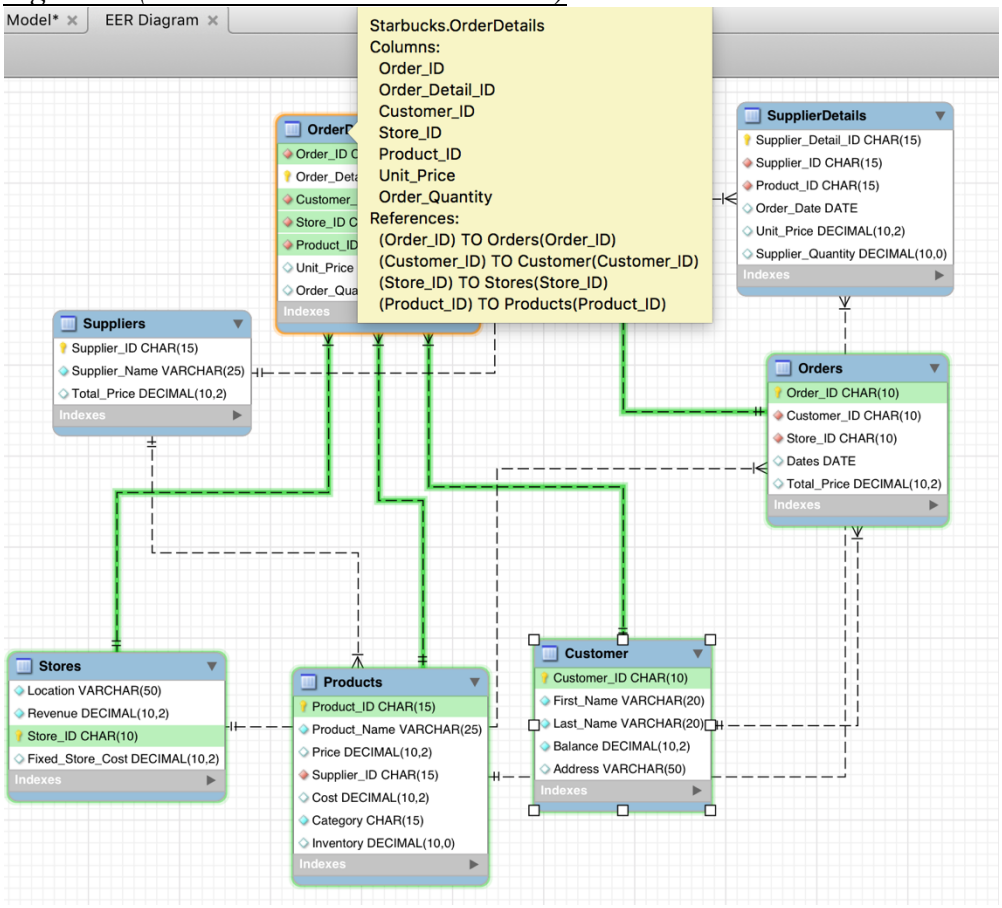
Figure 8 (OrderDetails Table Field View)

Order_ID	Order_Detail_ID	Customer_ID	Store_ID	Product_ID	Unit_Price	Order_Quantity
0001	01	001	ST1	P5	4.95	2
0002	02	002	ST2	P1	3.65	1
0002	03	002	ST2	P2	4.15	1
0003	04	999	ST3	P3	4.45	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 9 (OrderDetails Table Setting)

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
Order_ID	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
Order_Detail_ID	CHAR(10)	☑	☑	☐	☐	☐	☐	☐	☐	
Customer_ID	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
Store_ID	CHAR(10)	☐	☑	☐	☐	☐	☐	☐	☐	
Product_ID	CHAR(15)	☐	☑	☐	☐	☐	☐	☐	☐	
Unit_Price	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Order_Quantity	DECIMAL(10,0)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 10 (OrderDetails Table Connection)



In the OrderDetails Table, we include details of each placed order. OrderDetail ID is the primary key in this table. For instance, Customer002 bought 2 different products in one order. The unit price for P1 is $3.65 and the unit price for P2 is $4.15. Therefore, Customer002 has contributed $7.8 in total.

Primary key: Order_Detail_ID

Foreign keys: Order_ID references Orders table
Customer_ID references Customer table
Store_ID references Stores table
Product_ID references Products table

- Products Table

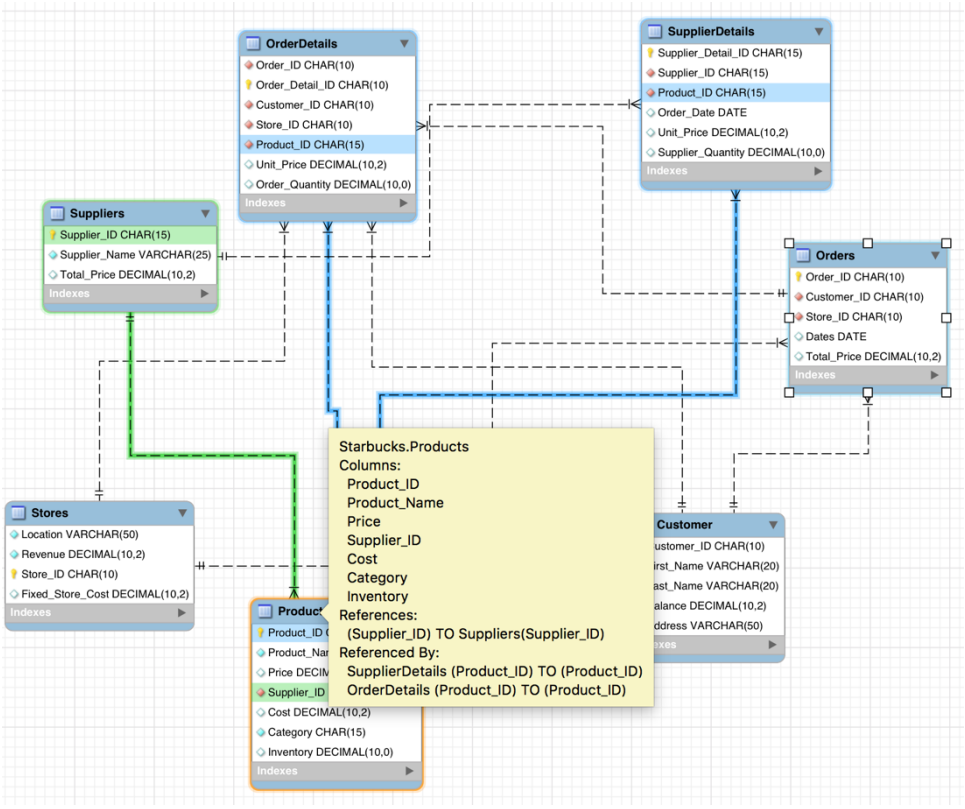
Figure 11 (Products Table Field View)

Product_ID	Product_Name	Price	Supplier_ID	Cost	Category	Inventory
P1	Caffe Latte	3.65	S1	1.00	C1	200000
P2	Caffe Mocha	4.15	S2	1.50	C2	150000
P3	White Chocolate Mocha	4.45	S2	2.00	C2	50000
P4	Skinny Peppermint Mocha	4.65	S2	2.10	C2	40000
P5	Toasted Graham Latte	4.95	S1	2.00	C1	2000
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 12 (Products Table Setting)

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
Product_ID	CHAR(15)	☑	☑	☐	☐	☐	☐	☐	☐	
Product_Name	VARCHAR(25)	☐	☑	☐	☐	☐	☐	☐	☐	
Price	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Supplier_ID	CHAR(15)	☐	☑	☐	☐	☐	☐	☐	☐	
Cost	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
Category	CHAR(15)	☐	☑	☐	☐	☐	☐	☐	☐	
Inventory	DECIMAL(10,0)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 13 (Products Table Connection)



In the Products Table, we can see product information. We include the product's name, the price of each product, the cost of each product that we pay to our suppliers. We also have the inventory information of each products and the product category. For instance, P1 is Caffe Latte in the C1 Category; we have an inventory of 200,000 Caffe Latte. It is sold at $3.65 per unit. We pay supplier1 $1 per unit.

Primary key: Product_ID

Foreign key: Supplier_ID references Suppliers table

- Stores Table

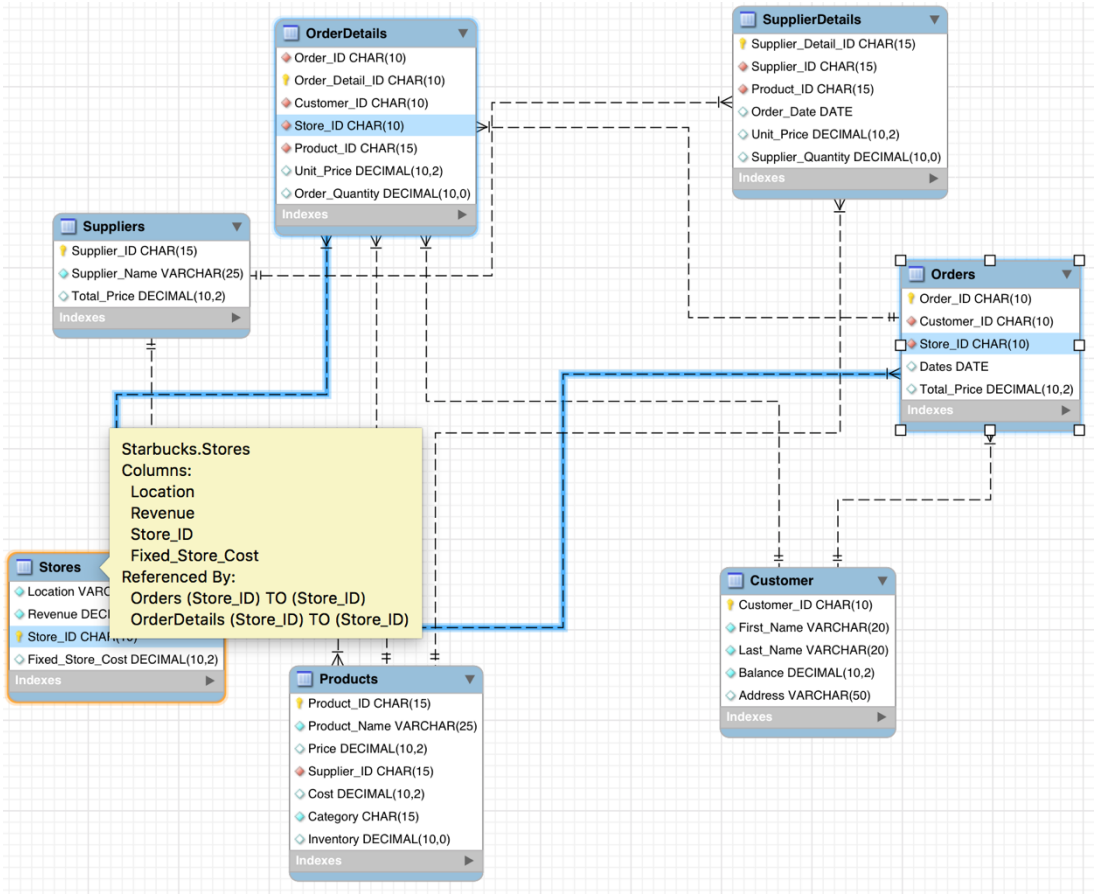
Figure 14 (Stores Table Field View)

Location	Revenue	Store_ID	Fixed_Store_Cost
100 10th St. New York NY 10003	5000.00	ST1	2000.00
50 60th St. New York NY 10023	8000.00	ST2	4000.00
35 22nd St. New York NY 10010	12000.00	ST3	6000.00
NULL	NULL	NULL	NULL

Figure 15 (Stores Table Setting)

Column	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
Location	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐	☐	
Revenue	DECIMAL(10,2)	☐	☑	☐	☐	☐	☐	☐	☐	
Store_ID	CHAR(10)	☑	☑	☐	☐	☐	☐	☐	☐	
Fixed_Store_Cost	DECIMAL(10,2)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
<click to edit>		☐	☐	☐	☐	☐	☐	☐	☐	

Figure 16 (Stores Table Connection)



In the Stores Table, we include the location information of each store, total revenue generated in each store and fixed cost of operating each store. For instance, the store ST1 in total generates $5,000 and its fixed cost is $2,000. STI's address is 100 10th St. New York NY 10003.

Primary key: Store_ID

- SupplierDetails Table

Figure 17 (SupplierDetails Table Field View)

Supplier_Detail_ID	Supplier_ID	Product_ID	Order_Date	Unit_Price	Supplier_Quantity
01	S1	P1	2015-02-15	1.00	200000
02	S2	P2	2015-03-16	1.50	150000
03	S2	P3	2015-04-17	2.00	50000
04	S2	P4	2015-05-18	2.10	40000
05	S1	P5	2015-06-19	2.00	2000
NULL	NULL	NULL	NULL	NULL	NULL

Figure 18 (SupplierDetails Table Setting)

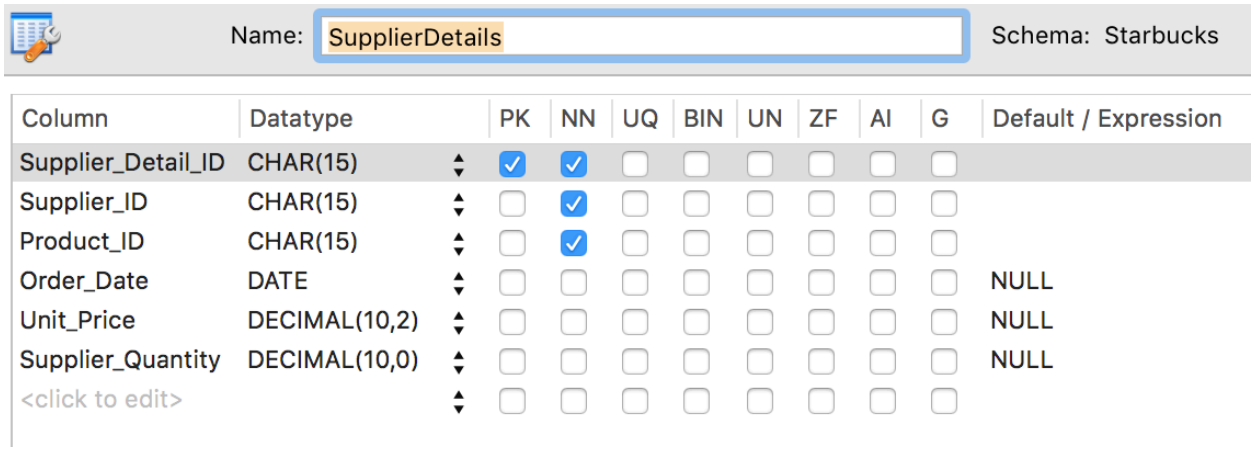
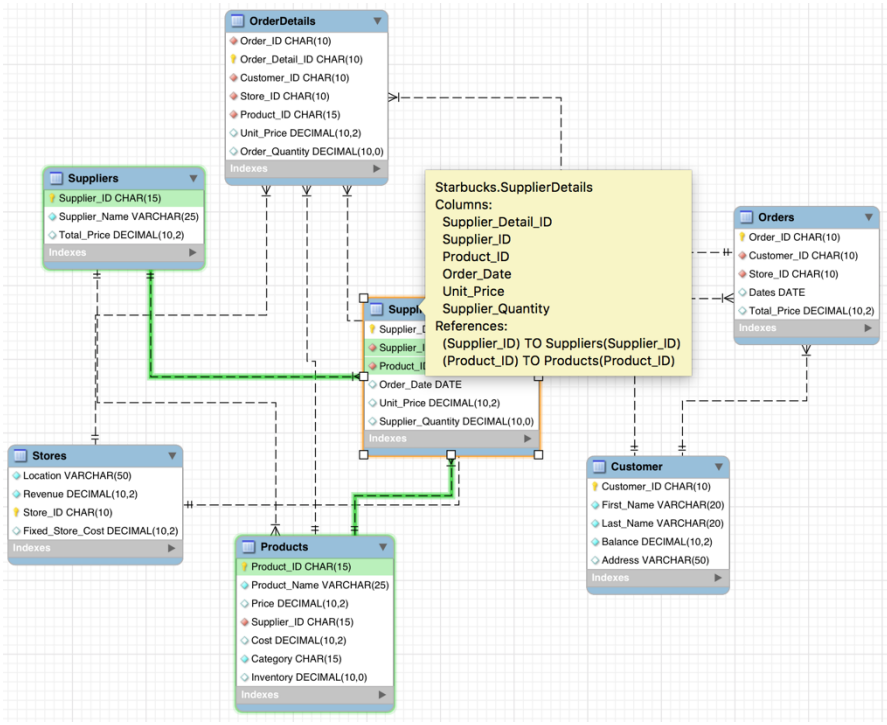


Figure 19 (SupplierDetails Table Connection)



In the SupplierDetails Table, we include suppliers' identification information and their transactions with Starbucks. We have listed SupplierID, ProductID, OrderDate, UnitPrice and Quantity information. For example: Supplier S1 sold 200,000 P1 at $1 per unit at Feb, 15th, 2015. Supplier S1 also sold 2,000 P5 at $2 per unit at Jun, 19th, 2015.

Primary key: Supplier_Detail_ID

Foreign keys: Supplier_ID (references Suppliers table)
Product_ID (references Products table)

- Suppliers Table

Figure 20 (Suppliers Table Field View)

Supplier_ID	Supplier_Name	Total_Price
S1	Latte Industry	204000.00
S2	Mocha Industry	409000.00
NULL	NULL	NULL

Figure 21 (Suppliers Table Setting)

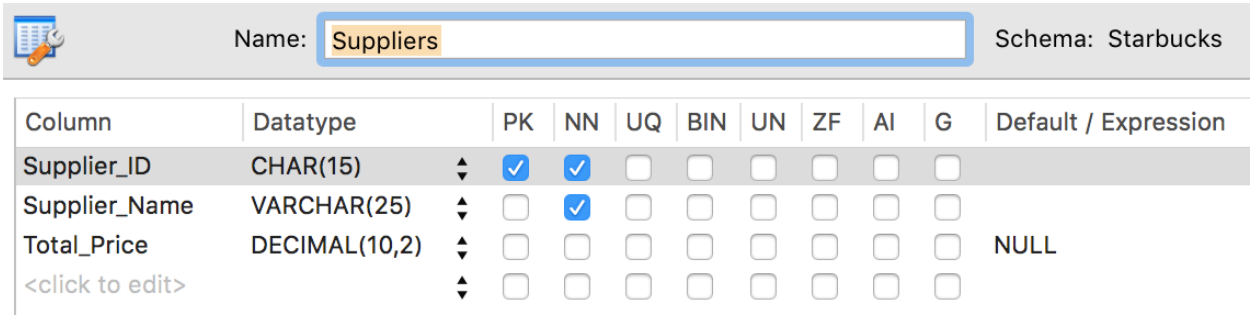
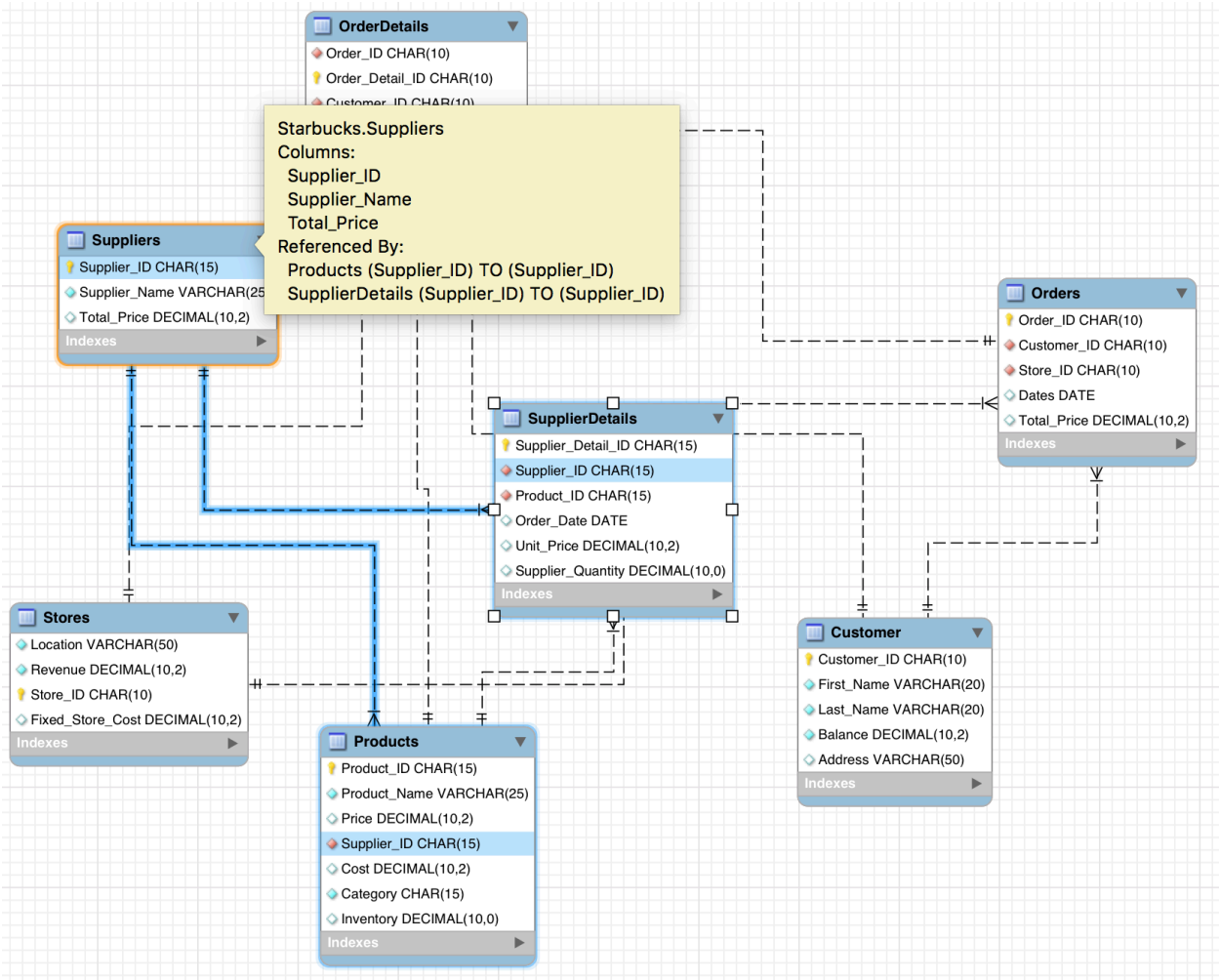


Figure 22 (Suppliers Table Connection)



In the Suppliers Table, we include all the supplier information: SupplierID, supplier's name, and the total amont of money paid to each supplier. For instance, Supplier S1, whose name is Latte Industry, sold products to Starbucks that added up to $204,000.

Primary key: Supplier_ID

4. Queries

1)

- **Store Performance Query**

```
SELECT Store_ID, Location, Revenue-Fixed_Store_Cost As StoreProfit
FROM Stores
ORDER BY (Revenue-Fixed_Store_Cost) Desc;
```

Figure 23

Store_ID	Location	StoreProfit
ST3	35 22nd St. New York NY 10010	6000.00
ST2	50 60th St. New York NY 10023	4000.00
ST1	100 10th St. New York NY 10003	3000.00

Management wants to rank the stores and find out which store is the most profitable. The profit of a store is calculated by store revenue minus fixed store cost. After finding out which store is the most profitable, management can seek opportunities to open new stores in that area. Same for the least profitable store, management can implement proper strategic adjustment accordingly for that store and possibly area.

- **Low Performance Customers Query**

```
SELECT Customer_ID, First_Name, Last_Name, Balance
FROM Customer
WHERE Balance<20 AND Balance>0
ORDER BY Balance;
```

Figure 24

Customer_ID	First_Name	Last_Name	Balance
004	Kyle	Costner	0.75
003	John	Ibrahim	15.10
NULL	NULL	NULL	NULL

This query shows low performance customers. The low performance customer is defined as someone who has a balance lower than $20. Management team can design promotions for this group of customers because they have the ability to contribute more. We only want to choose Starbucks members and give members promotion, so customers who have $0 in the balance will be eliminated.

2)

- **Inner join query-High Performance Product**

```
SELECT P.Product_ID, P.Product_Name,
Sum((P.Price-P.Cost)*O.Order_Quantity)As ProductProfit
FROM OrderDetails O, Products P
WHERE O.Product_ID=P.Product_ID
GROUP BY P.Product_ID, P.Product_Name
ORDER BY Sum((P.Price-P.Cost)*O.Order_Quantity) Desc;
```

Figure 25

Product_ID	Product_Name	ProductPro...
P3	White Chocolate Mocha	7.35
P5	Toasted Graham Latte	5.90
P1	Caffe Latte	2.65
P2	Caffe Mocha	2.65

This query is designed to find the highest performing product, defined as high profit product. The product profit is calculated by the difference value of Price in the Products Table and Cost in the Products Table times the quantity of the products that have been sold. Management team can provide more "White Chocolate Mocha" in the stores and give more promotions o this product to generate more profits.

- **Outer Join Query-High Performance Supplier**

```
SELECT S.Supplier_ID, S.Supplier_Name,
  Sum(O.Order_Quantity)/Sum(P.Inventory)*Sum(P.Price-P.Cost)/Sum(P.Price)
  As SupplierContribution
FROM Suppliers S LEFT JOIN (Products P LEFT JOIN OrderDetails O ON O.Product_ID=P.
Product_ID) ON S.Supplier_ID=P.Supplier_ID
GROUP BY S.Supplier_ID, S.Supplier_Name
ORDER BY Sum(O.Order_Quantity)/Sum(P.Inventory)*Sum(P.Price-P.Cost)/Sum(P.Price)
Desc;
```

Figure 26

Supplier_ID	Supplier_Name	SupplierContribution
S1	Latte Industry	0.0000096704
S2	Mocha Industry	0.0000096223

This query is designed to find out the performance of suppliers. High performance supplier is defined by Supplier Contribution.
Supplier Contribution=Sold product rate*Product profit rate
Sold product rate=Sold product quantity/Product Inventory

Product profit rate=Product profit/Product price
A good supplier is someone who can provide fast sold product and also a high profit product. For the low performing supplier, management team should take certain actions such as reduce purchasing from supplier S2 as well as increase the price of products from supplier S2.

3)

- **Filtered Data Before Aggregation with "Where" Query**

```
SELECT
Orders.Total_Price, Stores.Revenue, Stores.Store_ID
FROM Orders, Stores
WHERE Orders.Store_ID = Stores.Store_ID
ORDER BY Stores.Revenue Desc
LIMIT 1;
```

Figure 27

Total_Price	Revenue	Store_ID
13.35	12000.00	ST3

A manager can easily view the order with the highest price and also the store with the highest revenue using this query.
Where statement is used here for before aggregation happens.

- **Filtered Data After Aggregation with "Having" Query**

```
SELECT
Sum(Orders.Total_Price) AS OrdersTotal,
Sum(Stores.Revenue) AS StoreRevenue
FROM Orders, Stores
HAVING Sum(Stores.Revenue)>5000;
```

Figure 28

OrdersTotal	StoreRevenue
93.15	75000.00

A manager can see the aggregated sum of all orders and revenue for all stores (on the condition that the revenue displayed are above 5000) using this query.
Having statement is used here due to the aggregation, different scenario from the Where statement.

4)

- **Create a View Using Low Performance Customer Query**

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `starbucks`.`low_performance_customer` AS
    SELECT
    `starbucks`.`customer`.`Customer_ID` AS `Customer_ID`,
    `starbucks`.`customer`.`First_Name` AS `First_Name`,
    `starbucks`.`customer`.`Last_Name` AS `Last_Name`,
    `starbucks`.`customer`.`Balance` AS `Balance`
    FROM
    `starbucks`.`customer`
    WHERE
    ((`starbucks`.`customer`.`Balance` < 20)
     AND (`starbucks`.`customer`.`Balance` > 0))
ORDER BY `starbucks`.`customer`.`Balance`
```

Figure 29

Customer_ID	First_Name	Last_Name	Balance
004	Kyle	Costner	0.75
003	John	Ibrahim	15.10