

# **DATABASE SYSTEMS**

**BSSE Fall 21**

## **Semester Project Report**

**Project Title:**

**Food Express – An Online Food Delivery Solution**

**Submitted to**  
**Dr. Asif Sohail**

**Submitted By**  
**Group no 5**  
BSEF21M001 – Yeshal Khan  
BSEF21M008 – Zohaib Shahid  
BSEF21M0020 – Rabail Waseem



## **1- Introduction to the working of the system**

Food Express is a database-driven platform that facilitates the seamless operation and management of food ordering, processing, and delivery services. It primarily relies on a robust database architecture to store, manage, and retrieve various types of information. Here's a concise overview of its functionality from a database-oriented perspective:

### **1. Database Design:**

- The system's foundation lies in a well-structured database schema designed to efficiently store information related to customers, restaurants, orders, deliveries, items, branches, and more.
- It consists of multiple interrelated tables designed to maintain data integrity and support complex relationships between entities.

### **2. User Data Management:**

- Customer information, including profiles, addresses, contact details, and order history, is stored securely in the database.
- Authentication mechanisms manage user credentials and access privileges.

### **3. Restaurant and Menu Management:**

- The system's database maintains records of partnered restaurants, their menus, available items, prices, descriptions, and availability status.
- Each restaurant's menu data is stored and linked to ensure real-time updates and accurate listings.

### **4. Order Processing and Tracking:**

- Orders placed by customers are logged in the database, detailing itemized selections, quantities, total costs, delivery information, and status (e.g., 'pending,' 'in progress,' 'delivered').
- The database handles transactional data, capturing payment details, billing information, and order timestamps.

### **5. Delivery Management:**

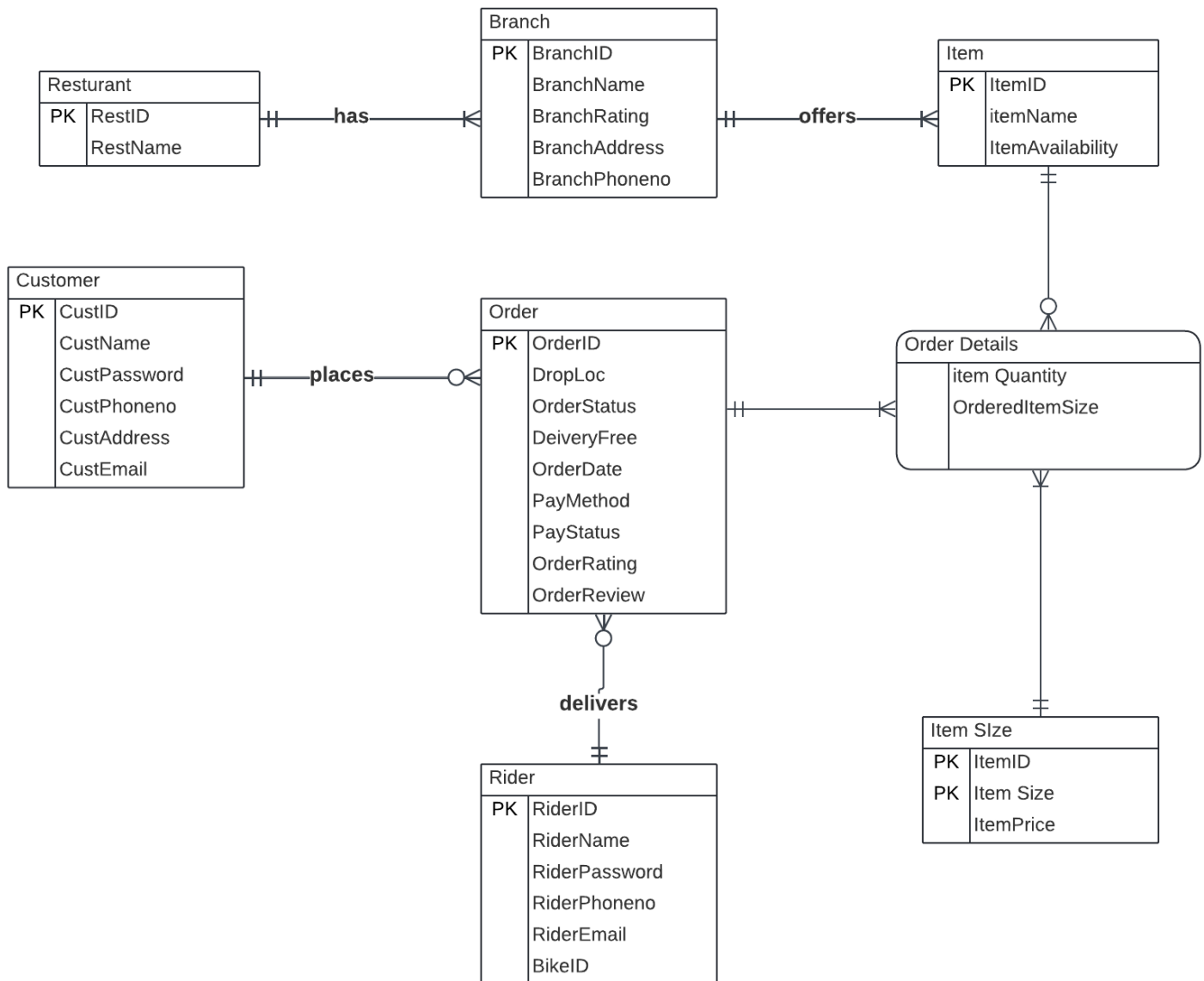
- Details of delivery personnel ('riders'), their availability, assigned orders, routes, and completion status are tracked and managed within the database.
- Real-time assignment and tracking of deliveries leverage database functionalities for efficient logistics management.

## **2- Problems in the existing system (optional)**

Before the introduction of online food delivery systems, traditional food delivery methods faced challenges such as limited food options, manual ordering processes via phone calls, inefficient delivery logistics, restricted payment methods, and a lack of real-time order tracking. These limitations led to

inconvenience for customers and operational inefficiencies for restaurants. The emergence of online food delivery systems addressed these issues by offering a digital platform that provided diverse food choices, streamlined ordering processes, improved delivery logistics, multiple payment options, real-time order tracking, and enhanced customer satisfaction.

### 3- ERD of the system



#### 4- Construction of the Relational Schema by using both bottom-up approach and top-down approach.

##### a) Top-Down Approach:

In the top-down approach, we will convert the ERD to the relational schemas by applying the conversion rules.

##### Step-1

Each regular entity will be converted into a relation.

**Customer** (CustID, CustName, CustPassword, CustEmail, CustPhone, CustAddress)

**Order** (OrderID, DropLoc, OrderStatus, DeliveryFee, OrderDate, OrderReview, OrderRating, PayMethod, PayStatus)

**Rider** (RiderID, RiderPhone, RiderName, RiderEmail, RiderPassword, RiderBikeID)

**Item** (ItemID, ItemName, ItemAvailibilty)

**Branch** (BranchID, BranchName, BranchRating, BranchAddress, BranchPhoneno)

**Restaurant** (RestID, RestName)

**ItemSize** (ItemID, ItemSize, ItemPrice)

##### Step-2

Mapping Binary one to many relationships.

For each binary 1:M relationship, the primary key attribute (or attributes) of the entity on the one-side of the relationship will be included as a foreign key in the relation that is on the many-side of the relationship.

- The PK of Customer and Rider will be FK in Order.
- The PK of Restaurant relation will be included in Branch.
- The PK of Branch will be included in the Item Relation.

**Order** (OrderID, DropLoc, OrderStatus, DeliveryFee, OrderDate, OrderReview, OrderRating, PayMethod, PayStatus, RiderID, CustID)

**Rider** (RiderID, RiderPhone, RiderName, RiderEmail, RiderPassword, RiderBikeID)

**Item** (ItemID, ItemName, ItemAvailibilty, BranchID)

**Branch (BranchID, BranchName, BranchRating, BranchAddress, BranchPhoneno, RestID)**

#### Step-4 Mapping Associative Entities

We'll create a separate table for the Order\_Details entity and the PKs of the entities which are in relationship with the associative entity will become part of the PK of the Order\_Details along with the Ordered\_Item\_Size and each of them will act as FK as well for their respective relation.

**OrderDetails (OrderID, ItemID, OrderedItemSize, ItemQuantity)**

So, the relations as the result of the Top-Down Approach are as follow:

**Customer (CustID, CustName, CustPassword, CustEmail, CustPhone, CustAddress)**

**Order (OrderID, DropLoc, OrderStatus, DeliveryFee, OrderDate, OrderReview, OrderRating, PayMethod, PayStatus, RiderID, CustID)**

CustID references Customer (CustID)

RiderID reference Rider(RiderID)

**OrderDetails (OrderID, ItemID, OrderedItemSize, ItemQuantity)**

OrderID references Order(OrderID)

ItemID, OrderedItemSize references ItemSize (ItemID, ItemSize)

**Rider (RiderID, RiderPhone, RiderName, RiderEmail, RiderPassword, RiderBikeID)**

**Item (ItemID, ItemName, ItemAvailibilty, BranchID)**

BranchID references Branch(BranchID)

**Branch (BranchID, BranchName, BranchRating, BranchAddress, BranchPhoneno, RestID)**

RestID references Restaurant(RestID)

**Restaurant (RestID, RestName)**

**ItemSize (ItemID, ItemSize, ItemPrice)**

#### **b) Bottom-Up Approach:**

Now we are converting the following large, complex, and unstable dataset of our system to the set of small and stable relations.

**Order** (OrderID, CustID, CustName, CustPassword, CustEmail, CustPhone, CustAddress, DropLoc, OrderStatus, DeliveryFee, OrderDate, OrderReview, OrderRating, RiderID, RiderPhone, RiderName, RiderEmail, RiderPassword, RiderBikeID, RestID, RestName, BranchID, BranchName, BranchRating, Branch\_Address, BranchPhoneno, ItemID, ItemName, ItemPrice, ItemQuantity, OrderedItemSize, ItemSize, PayMethod, PayStatus, ItemAvailability)

We will achieve our goal through the process of normalization up to 3NF.

## 1NF:

In 1NF we check for the repeating groups/multivalued attributes. We have multiple repeating groups. We will eliminate these in each step of 1NF. One Customer can have multiple orders.

Customer (CustID, CustName, CustPassword, CustEmail, CustPhone, CustAddress) in 1NF.

Order (OrderID, DropLoc, OrderStatus, DeliveryFee, OrderDate, OrderReview, OrderRating, RiderID, RiderPhone, RiderName, RiderEmail, RiderPassword, RiderBikeID, RestID, RestName, BranchID, BranchName, BranchRating, BranchAddress, ItemID, ItemName, ItemPrice, ItemQuantity, OrderedItemSize, ItemSize, PayMethod, PayStatus, ItemAvailability, CustID) not in 1NF.

Order relation has repeating group, one order can have multiple items.

Order (OrderID, CustID, DropLoc, OrderStatus, DeliveryFee, OrderDate, OrderReview, OrderRating, RiderID, RiderPhone, RiderName, RiderEmail, PayMethod, PayStatus, RiderPassword, RiderBikeID) in 1NF.

OrderDetails (OrderID, ItemID, RestID, RestName, BranchID, BranchName, BranchRating, BranchAddress, ItemName, ItemPrice, ItemQuantity, OrderedItemSize, ItemSize, ItemAvailability) not in 1NF.

The OrderDetails table has a repeating as well so, we'll remove it.

1NF on OrderDetails:

OrderDetails (OrderID, ItemID, OrderedItemSize, ItemQuantity) in 1NF.

Item (ItemID, ItemSize, ItemName, ItemPrice, ItemAvailability, RestID, RestName, BranchID, BranchName, BranchRating, BranchAddress, BranchPhoneno) in 1NF.

## 2NF:

We have separated the ItemSize relation because the rest of the attributes in the Item table were determined by the ItemID.

Relations in 2NF:

Customer (CustID, CustName, CustPassword, CustEmail, CustPhone, CustAddress)

Order (OrderID, DropLoc, OrderStatus, DeliveryFee, OrderDate, OrderReview, OrderRating, PayMethod, PaymentStatus, RiderID, RiderPhone, RiderName, RiderEmail, RiderPassword, RiderBikeID, CustID)

OrderDetails (OrderID, ItemID, OrderedItemSize, ItemQuantity)

Item (ItemID, ItemName, ItemAvailability, RestID, RestName, BranchID, BranchName, BranchRating, BranchAddress, BranchPhoneno)

Item Size (ItemID, ItemSize, ItemPrice)

### 3NF:

The Rider relation will be separated from the order relation. The Branch Relation will be separated from the Item relation and Restaurant will be separated from the Branch due to transitive dependency.

**Relational Schemas as result of 3NF will be:**

**Customer (CustID, CustName, CustPassword, CustEmail, CustPhone, CustAddress)**

**Order (OrderID, DropLoc, OrderStatus, DeliveryFee, OrderDate, OrderReview, OrderRating, PayMethod, PayStatus, RiderID, CustID)**

CustID references Customer (CustID)

RiderID reference Rider(RiderID)

**OrderDetails (OrderID, ItemID, OrderedItemSize, ItemQuantity)**

OrderID references Order(OrderID)

ItemID, OrderedItemSize references ItemSize (ItemID, ItemSize)

**Rider (RiderID, RiderPhone, RiderName, RiderEmail, RiderPassword, RiderBikeID)**

**Item (ItemID, ItemName, ItemAvailability, BranchID)**

BranchID references Branch(BranchID)

**Branch (BranchID, BranchName, BranchRating, BranchAddress, BranchPhoneno, RestID)**

RestID references Restaurant(RestID)

**Restaurant (RestID, RestName)**

ItemSize (ItemID, ItemSize, ItemPrice)

## 5- Description of the relations

RIDER		
ATTRIBUTES	DATA TYPES	CONSTRAINTS
<b>RiderID</b>	Varchar2(10)	PRIMARY KEY
<b>RiderName</b>	Varchar2(30)	
<b>RiderPassword</b>	varchar2(10),	NOT NULL, LENGTH>=8
<b>RiderPhoneNo</b>	number (11),	NOT NULL, UNIQUE, LENGTH=11
<b>RiderEmail</b>	varchar2(30),	NOT NULL
<b>BikeID</b>	Varchar2(10)	

ITEMSIZE		
ATTRIBUTES	DATA TYPES	CONSTRAINTS
<b>ItemID</b>	Varchar2(10)	PRIMARY KEY
<b>ItemSize</b>	Varchar2(25)	PRIMARY KEY
<b>ItemPrice</b>	varchar2(10)	NOT NULL

CUSTOMER		
ATTRIBUTES	DATA TYPES	CONSTRAINTS
<b>CustID</b>	Varchar2(10)	PRIMARY KEY
<b>CustName</b>	Varchar2(30)	NOT NULL
<b>CustPassword</b>	varchar2(10)	NOT NULL, UNIQUE LENGTH>=8
<b>CustPhoneNo</b>	number (11)	NOT NULL, LENGTH=11
<b>CustAddress</b>	varchar2(50)	
<b>CustEmail</b>	Varchar2(30)	NOT NULL



RESTAURANT		
ATTRIBUTES	DATA TYPES	CONSTRAINTS
<b>RestID</b>	Varchar2(10)	PRIMARY KEY
<b>RestName</b>	Varchar2(30)	NOT NULL

BRANCH		
ATTRIBUTES	DATA TYPES	CONSTRAINTS
<b>BranchID</b>	Varchar2(10)	PRIMARY KEY
<b>BranchName</b>	Varchar2(30)	NOT NULL
<b>BranchReview</b>	varchar2(100)	
<b>BranchAddress</b>	Varchar2(50)	NOT NULL
<b>RestaurantID</b>	varchar2(10)	FOREIGN KEY (RESTAURANT)

ITEM		
ATTRIBUTES	DATA TYPES	CONSTRAINTS
<b>ItemID</b>	Varchar2(10)	PRIMARY KEY
<b>ItemName</b>	Varchar2(30)	NOT NULL
<b>BranchID</b>	varchar2(10)	FOREIGN KEY(BRANCH)
<b>ItemAvailability</b>	char(1)	CAN BE 'Y' OR 'N'

ORDERDETAILS		
ATTRIBUTES	DATA TYPES	CONSTRAINTS
<b>ItemQuantity</b>	Number(3)	NOT NULL
<b>OrderedItemSize</b>	Varchar2(50)	PRIMARY KEY, FOREIGN KEY(ITEM_SIZE)
<b>ItemID</b>	varchar2(10)	PRIMARY KEY, FOREIGN KEY(ITEM_SIZE)

<b>ORDERS</b>		
<b>ATTRIBUTES</b>	<b>DATA TYPES</b>	<b>CONSTRAINTS</b>
	Varchar2(10)	PRIMARY KEY
<b>DropLoc</b>	Varchar2(50)	NOT NULL
<b>Status</b>	varchar2(10)	CAN BE 'PENDING', 'IN PROGRESS', 'DELIVERED'
<b>DelieveryFee</b>	number(4)	
<b>OrderDate</b>	date	NOT NULL
<b>CustID</b>	Varchar2(10)	FOREIGN KEY(CUSTOMER)
<b>RiderID</b>	Varchar2(10)	FOREIGN KEY(RIDER)
<b>PayMethod</b>	Varchar2(10)	CAN BE 'BANK TRANSFER', 'COD', 'CREDIT/DEBIT CARD'
<b>OrderStatus</b>	Varchar2(11)	CAN BE 'IN MAKING', 'IN PROGRESS', 'DELIVERED'
<b>PayStatus</b>	varchar2(10)	NOT NULL, CAN BE 'PENDING' OR 'COMPLETED'
<b>OrderReview</b>	Varchar2(100)	
<b>OrderRating</b>	Number(1)	BETWEEN 1 AND 5

## 6- Create Statements for Relations:

CREATE TABLE Customer (

CustID VARCHAR2(10) CONSTRAINT pk\_Customer PRIMARY KEY,

CustName VARCHAR2(30) CONSTRAINT nn\_CustName NOT NULL,

CustPassword VARCHAR2(10) CONSTRAINT nn\_CustPassword NOT NULL,

CustPhone NUMBER(11) CONSTRAINT nn\_CustPhone NOT NULL,

CustAddress VARCHAR2(50) CONSTRAINT nn\_CustAddress NOT NULL,

CustEmail VARCHAR2(30) CONSTRAINT uk\_CustEmail UNIQUE,

CONSTRAINT chk\_CustPasswordLength CHECK (LENGTH(CustPassword) >= 8),

```
CONSTRAINT uk_CustPhone UNIQUE (CustPhone)
);

CREATE TABLE Rider (
    RiderID VARCHAR2(10) CONSTRAINT pk_Rider PRIMARY KEY,
    RiderName VARCHAR2(30) CONSTRAINT nn_RiderName NOT NULL,
    RiderPassword VARCHAR2(10) CONSTRAINT nn_RiderPassword NOT NULL,
    RiderPhone NUMBER(11) CONSTRAINT nn_RiderPhone NOT NULL,
    RiderEmail VARCHAR2(30) CONSTRAINT uk_RiderEmail UNIQUE,
    RiderBikeID VARCHAR2(10),
    CONSTRAINT chk_RiderPasswordLength CHECK (LENGTH(RiderPassword) >= 8),
    CONSTRAINT uk_RiderPhone UNIQUE (RiderPhone)
);
```

```
CREATE TABLE Restaurant (
    RestID VARCHAR2(10) CONSTRAINT pk_Restaurant PRIMARY KEY,
    RestName VARCHAR2(30) CONSTRAINT nn_RestName NOT NULL
);
```

```
CREATE TABLE Branch (
    BranchID VARCHAR2(10) CONSTRAINT pk_Branch PRIMARY KEY,
    BranchName VARCHAR2(30) CONSTRAINT nn_BranchName NOT NULL,
    BranchAddress VARCHAR2(50) CONSTRAINT nn_BranchAddress NOT NULL,
    BranchRating NUMBER CHECK (BranchRating BETWEEN 1 AND 5),
    RestID VARCHAR2(10) CONSTRAINT fk_Branch_Restaurant REFERENCES Restaurant(RestID)
);
```

```
CREATE TABLE Item (
    ItemID VARCHAR2(10) CONSTRAINT pk_Item PRIMARY KEY,
```

```
ItemName VARCHAR2(30) CONSTRAINT nn_ItemName NOT NULL,  
BranchID VARCHAR2(10) CONSTRAINT fk_Item_Branch REFERENCES Branch(BranchID),  
ItemAvailability CHAR(1) CHECK (ItemAvailability IN ('Y', 'N'))  
);
```

```
CREATE TABLE ItemSize (  
    ItemID VARCHAR2(10),  
    ItemSize VARCHAR2(25),  
    ItemPrice NUMBER(5) CONSTRAINT nn_ItemPrice NOT NULL,  
    CONSTRAINT pk_ItemSize PRIMARY KEY (ItemID, ItemSize)  
);
```

```
CREATE TABLE Orders (  
    OrderID VARCHAR2(10) CONSTRAINT pk_Orders PRIMARY KEY,  
    DropLoc VARCHAR2(50) CONSTRAINT nn_DropLoc NOT NULL,  
    OrderStatus VARCHAR2(11) CONSTRAINT chk_OrderStatus CHECK (OrderStatus IN ('IN MAKING', 'IN  
PROGRESS', 'DELIVERED')),  
    DeliveryFee NUMBER(4),  
    PayMethod VARCHAR2(17) CONSTRAINT chk_PayMethod CHECK (PayMethod IN ('BANK TRANSFER',  
'COD', 'CREDIT/DEBIT CARD')),  
    PayStatus VARCHAR2(9) CONSTRAINT chk_PayStatus CHECK (PayStatus IN ('PENDING', 'COMPLETED')),  
    OrderDate DATE DEFAULT SYSDATE,  
    CustID VARCHAR2(10) CONSTRAINT fk_Order_Customer REFERENCES Customer(CustID),  
    RiderID VARCHAR2(10) CONSTRAINT fk_Order_Rider REFERENCES Rider(RiderID),  
    OrderReview VARCHAR2(100),  
    OrderRating NUMBER CONSTRAINT chk_OrderRating CHECK (OrderRating BETWEEN 1 AND 5)  
);
```

```

CREATE TABLE OrderDetails (
    ItemQuantity NUMBER(3) DEFAULT 1 CONSTRAINT nn_ItemQuantity NOT NULL,
    OrderedItemSize VARCHAR2(50),
    ItemID VARCHAR2(10),
    OrderID VARCHAR2(10) CONSTRAINT fk_OrderDetails_Order REFERENCES Orders(OrderID),
    CONSTRAINT pk_OrderDetails PRIMARY KEY (OrderedItemSize, ItemID, OrderID),
    CONSTRAINT fk_OrderDetails FOREIGN KEY (ItemID, OrderedItemSize) REFERENCES ItemSize
(ItemID,ItemSize)
);

```

## 7- Views

### 1) To display menu items of a restaurant's branch:

```

CREATE OR REPLACE VIEW Menu AS
SELECT I.ItemId, I.ItemName, I.ItemAvailability,
B.BranchId, B.BranchName, B.BranchAddress, B.BranchRating, B.RestID
FROM Item I JOIN Branch B
ON I.BranchId = B.BranchId;

```

#### Working:

```

SELECT * FROM Menu
WHERE BranchName = 'Jalal Sons Karachi';

```

Results Explain Describe Saved SQL History

ITEMID	ITEMNAME	ITEMAVAILABILITY	BRANCHID	BRANCHNAME	BRANCHADDRESS	BRANCHRATING	RESTID
808	Cheeseburger	N	704	Jalal Sons Karachi	101 PQR Lane, Karachi	3.9	602
810	Chicken Tenders	Y	704	Jalal Sons Karachi	101 PQR Lane, Karachi	3.9	602
809	Fries	Y	704	Jalal Sons Karachi	101 PQR Lane, Karachi	3.9	602
811	Pepperoni Pizza	Y	704	Jalal Sons Karachi	101 PQR Lane, Karachi	3.9	602

## 2) To display order history of a customer:

```
CREATE OR REPLACE VIEW OrderHistory AS
SELECT C.CustId, C.CustName, O.OrderID, O.OrderDate,
       D.ItemId, I.ItemName, D.ItemQuantity, D.OrderedItemSize
FROM Customer C JOIN OrderS O ON C.CustId = O.CustId
JOIN OrderDetails D ON O.OrderId = D.OrderId
JOIN Item I ON I.ItemId = D.ItemId;
```

### Working:

```
SELECT * FROM OrderHistory
WHERE CustId = 401
ORDER BY OrderId;
```

Results Explain Describe Saved SQL History

CUSTID	CUSTNAME	ORDERID	ORDERDATE	ITEMID	ITEMNAME	ITEMQUANTITY	ORDEREDITEMSIZE
401	Ali Khan	901	12/18/2023	802	Chocolate Cake	2	Small
401	Ali Khan	901	12/18/2023	801	Chicken Sandwich	1	Large
401	Ali Khan	901	12/18/2023	804	Chocolate Cake	1	Small (1 Pound)
401	Ali Khan	905	12/18/2023	810	Chicken Tenders	1	Large Pack (15 pieces)
401	Ali Khan	905	12/18/2023	809	Fries	2	Large Pack
401	Ali Khan	911	12/18/2023	801	Chicken Sandwich	1	Large
401	Ali Khan	911	12/18/2023	802	Chocolate Cake	2	Small
401	Ali Khan	911	12/18/2023	803	Chicken Sandwich	1	Small
401	Ali Khan	912	12/18/2023	804	Chocolate Cake	2	Large (2 Pounds)
401	Ali Khan	912	12/18/2023	805	Chicken Wings	1	Small Pack (10 wings)
401	Ali Khan	920	12/18/2023	802	Chocolate Cake	2	Small
401	Ali Khan	920	12/18/2023	801	Chicken Sandwich	2	Small
401	Ali Khan	920	12/18/2023	803	Chicken Sandwich	1	Large

## 9- Five Common Reports

### i. Branch-wise Revenue Report 2023

```
SELECT Branch.BranchID, Branch.BranchName, Restaurant.RestName, NVL(SUM(ItemSize.ItemPrice *
OrderDetails.ItemQuantity), 0) AS BranchRevenue FROM Branch JOIN Restaurant ON Branch.RestID =
Restaurant.RestID JOIN Item ON Branch.BranchID = Item.BranchID JOIN OrderDetails ON Item.ItemID =
OrderDetails.ItemID JOIN ItemSize ON OrderDetails.ItemID = ItemSize.ItemID and
OrderDetails.ordereditemsize=ItemSize.itemsize JOIN Order ON OrderDetails.OrderID = Order.OrderID
```

WHERE TO\_CHAR(Order.OrderDate, 'YYYY') = '2023' GROUP BY Branch.BranchID, Branch.BranchName, Restaurant.RestName ORDER BY BranchRevenue DESC;

BRANCHID	BRANCHNAME	RESTNAME	BRANCHREVENUE
704	Jalal Sons Karachi	Jalal Sons	30050
703	Jalal Sons Lahore	Jalal Sons	19400
701	Cafe Euphoria Lahore 1	Cafe Euphoria	11100
702	Cafe Euphoria Lahore 2	Cafe Euphoria	10100
710	Malmo Karachi	Malmo Bakers & Sweets	5400
708	Malmo Lahore 1	Malmo Bakers & Sweets	2800
706	Malmo Islamabad 1	Malmo Bakers & Sweets	2000
709	Malmo Lahore 2	Malmo Bakers & Sweets	750
705	Makbiz Peshawar	Makbiz Kebab Corner	310

9 rows returned in 0.01 seconds [Download](#)

## ii. Rider Performance Report

SELECT Rider.RiderID, Rider.RiderName, COUNT(Orders.OrderID) AS TotalDeliveries

FROM Rider

JOIN Orders ON Rider.RiderID = Orders.RiderID

WHERE Orders.OrderStatus = 'DELIVERED'

GROUP BY Rider.RiderID, Rider.RiderName

ORDER BY TotalDeliveries DESC;

RIDERID	RIDERNAME	TOTALDELIVERIES
505	Faisal Khan	2
508	Saima Malik	2
504	Zaviyar Ahmed	1
507	Tariq Ahmed	1
503	Arif Malik	1
509	Noman Hassan	1
501	Hassan Ali	1

### iii. Most Ordered items Report

```
SELECT * FROM ( SELECT Item.ItemName, Branch.BranchName, Restaurant.RestName,  
SUM(OrderDetails.ItemQuantity) AS TotalQuantityOrdered FROM OrderDetails  
JOIN Item ON OrderDetails.ItemID = Item.ItemID  
JOIN Branch ON Item.BranchID = Branch.BranchID JOIN Restaurant ON Branch.RestID =  
Restaurant.RestID GROUP BY Item.ItemName, Branch.BranchName, Restaurant.RestName  
ORDER BY SUM(OrderDetails.ItemQuantity) DESC  
)  
WHERE ROWNUM <= 5;
```

**Results** Explain Describe Saved SQL History

ITEMNAME	BRANCHNAME	RESTNAME	TOTALQUANTITYORDERED
Chocolate Cake	Cafe Euphoria Lahore 1	Cafe Euphoria	6
Chicken Sandwich	Cafe Euphoria Lahore 1	Cafe Euphoria	6
Fries	Jalal Sons Karachi	Jalal Sons	6
Pepperoni Pizza	Jalal Sons Karachi	Jalal Sons	5
Chicken Wings	Jalal Sons Lahore	Jalal Sons	5

5 rows returned in 0.01 seconds [Download](#)

### iv. Top Customers by order count

```
SELECT c.CustID, c.CustName, c.CustEmail, order_counts.TotalOrdersPlaced FROM Customer c JOIN (  
SELECT CustID, COUNT(OrderID) AS TotalOrdersPlaced FROM Orders GROUP BY CustID HAVING  
COUNT(OrderID) = (SELECT MAX(OrderCount) FROM ( SELECT COUNT(OrderID) AS OrderCount FROM  
Orders GROUP BY CustID ) )  
) order_counts ON c.CustID = order_counts.CustID;
```

CUSTID	CUSTNAME	CUSTEMAIL	TOTALORDERSPLACED
401	Ali Khan	alikh123@gmail.com	5



## v. "Orders in Progress with Customer Details Report"

```
SELECT o.OrderID, o.OrderDate, o.OrderStatus, o.DROPLOC, c.CustID, c.CustName, c.CustPhone
FROM Orders o JOIN Customer c ON o.CustID = c.CustID WHERE o.OrderStatus = 'IN PROGRESS';
```

ORDERID	ORDERDATE	ORDERSTATUS	DROPLOC	CUSTID	CUSTNAME	CUSTPHONE
902	12/21/2023	IN PROGRESS	456 Park Avenue, Islamabad	402	Sara Ahmed	3111234567
913	12/21/2023	IN PROGRESS	333 Beach Avenue, Karachi	403	Ahmed Hassan	3221234567
906	12/21/2023	IN PROGRESS	303 Skyline Avenue, Lahore	406	Nida Ali	3551234567
917	12/21/2023	IN PROGRESS	777 Seaside Street, Karachi	407	Imran Ahmed	3661234567
910	12/21/2023	IN PROGRESS	707 Galaxy Avenue, Peshawar	410	Kamran Ali	3991234567

## 8- Procedures, Functions and Triggers

### Procedures:

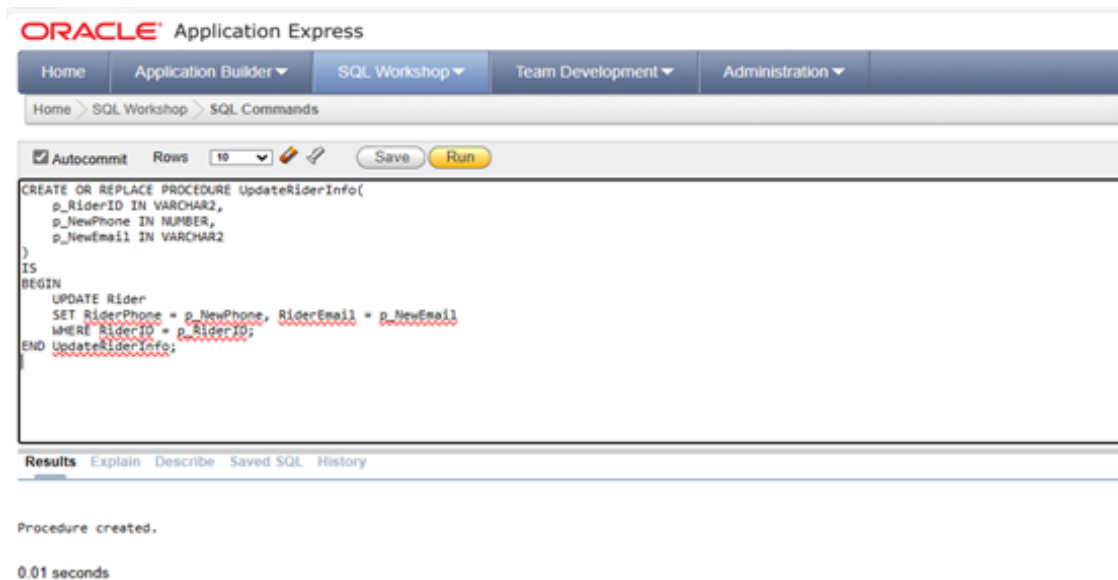
#### 1) Procedure to Update Rider Information

The screenshot displays the Oracle Application Express interface. The top navigation bar includes links for Home, Application Builder, SQL Workshop (selected), Team Development, and Administration. Below this, a breadcrumb trail shows Home > SQL Workshop > SQL Commands. The main workspace contains a SQL editor with the following code:

```
CREATE OR REPLACE PROCEDURE UpdateRiderInfo(
  p_RiderID IN VARCHAR2,
  p_NewPhone IN NUMBER,
  p_NewEmail IN VARCHAR2
)
IS
BEGIN
  UPDATE Rider
  SET RiderPhone = p_NewPhone, RiderEmail = p_NewEmail
  WHERE RiderID = p_RiderID;
END UpdateRiderInfo;
```

Below the editor, the 'Results' tab is active, showing the message 'Procedure created.' and the execution time '0.01 seconds'.

Working:

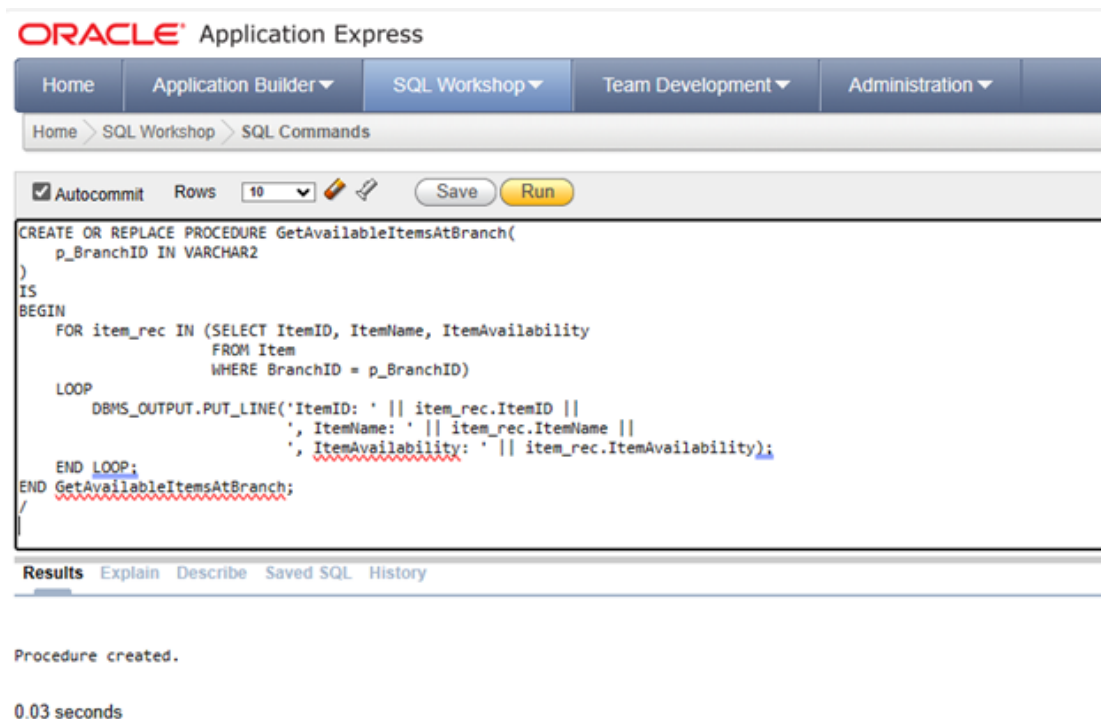


The screenshot shows the Oracle Application Express interface. The top navigation bar includes links for Home, Application Builder, SQL Workshop (selected), Team Development, and Administration. Below this, a breadcrumb trail shows Home > SQL Workshop > SQL Commands. The main editor area contains the following SQL code:

```
CREATE OR REPLACE PROCEDURE UpdateRiderInfo(  
  p_RiderID IN VARCHAR2,  
  p_NewPhone IN NUMBER,  
  p_NewEmail IN VARCHAR2  
)  
IS  
BEGIN  
  UPDATE Rider  
  SET RiderPhone = p_NewPhone, RiderEmail = p_NewEmail  
  WHERE RiderID = p_RiderID;  
END UpdateRiderInfo;
```

Below the editor, there are tabs for Results, Explain, Describe, Saved SQL, and History. The Results tab is active, displaying the message "Procedure created." and the execution time "0.01 seconds".

## 2) Procedure to Retrieve Available Items at a Branch:

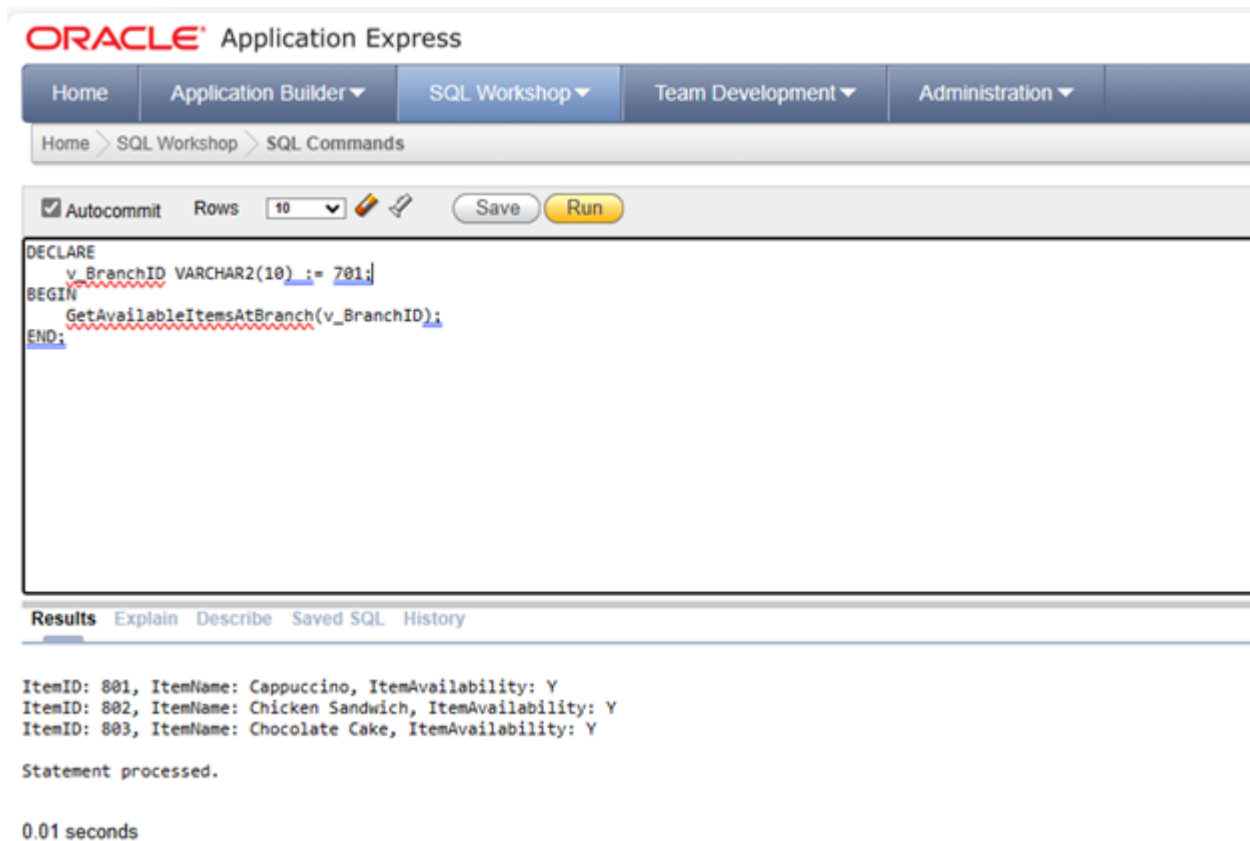


The screenshot shows the Oracle Application Express interface. The top navigation bar includes links for Home, Application Builder, SQL Workshop (selected), Team Development, and Administration. Below this, a breadcrumb trail shows Home > SQL Workshop > SQL Commands. The main editor area contains the following SQL code:

```
CREATE OR REPLACE PROCEDURE GetAvailableItemsAtBranch(  
  p_BranchID IN VARCHAR2  
)  
IS  
BEGIN  
  FOR item_rec IN (SELECT ItemID, ItemName, ItemAvailability  
    FROM Item  
    WHERE BranchID = p_BranchID)  
  LOOP  
    DBMS_OUTPUT.PUT_LINE('ItemID: ' || item_rec.ItemID ||  
      ', ItemName: ' || item_rec.ItemName ||  
      ', ItemAvailability: ' || item_rec.ItemAvailability);  
  END LOOP;  
END GetAvailableItemsAtBranch;
```

Below the editor, there are tabs for Results, Explain, Describe, Saved SQL, and History. The Results tab is active, displaying the message "Procedure created." and the execution time "0.03 seconds".

## Working:



The screenshot displays the Oracle Application Express interface. The top navigation bar includes links for Home, Application Builder, SQL Workshop (selected), Team Development, and Administration. Below this, a breadcrumb trail shows Home > SQL Workshop > SQL Commands. The main workspace contains a SQL editor with the following code:

```
DECLARE
  v_BranchID VARCHAR2(10) := 701;
BEGIN
  GetAvailableItemsAtBranch(v_BranchID);
END;
```

Below the editor, the 'Results' tab is active, showing the output of the SQL statement:

```
ItemID: 801, ItemName: Cappuccino, ItemAvailability: Y
ItemID: 802, ItemName: Chicken Sandwich, ItemAvailability: Y
ItemID: 803, ItemName: Chocolate Cake, ItemAvailability: Y

Statement processed.

0.01 seconds
```

## Triggers:

### 1) To check if an item is available before placing an order

```
CREATE OR REPLACE TRIGGER CheckItemAvailability BEFORE INSERT ON OrderDetails
FOR EACH ROW
DECLARE
  Avail Item.ItemAvailability%type;
BEGIN
  SELECT ItemAvailability INTO Avail FROM Item WHERE ItemId = :NEW.ItemId;
  IF (Avail = 'N') THEN
    RAISE_APPLICATION_ERROR (-20003,'Item is not currently available');
  END IF;
END;
```

## Working:

```
INSERT INTO OrderDetails  
VALUES (1, 'Small (1 Pound)', 804, 921);
```

Results Explain Describe Saved SQL History

```
ORA-20003: Item is not currently available  
ORA-06512: at "BSEF21M001.CHECKITEMAVAILABILITY", line 6  
ORA-04088: error during execution of trigger 'BSEF21M001.CHECKITEMAVAILABILITY'
```

## 2) To update a restaurant branch's rating based on the rating of its orders

```
CREATE OR REPLACE TRIGGER UpdateBranchRating  
AFTER UPDATE OF OrderRating ON Orders  
FOR EACH ROW  
DECLARE  
    PRAGMA AUTONOMOUS_TRANSACTION;  
    AvgRating Branch.BranchRating%TYPE;  
    BrId      Branch.BranchId%TYPE;  
    OldRating Branch.BranchRating%TYPE;  
BEGIN  
    -- Retrieve information for the updated order  
    SELECT BranchId INTO BrId FROM Item WHERE ItemId = (SELECT ItemId FROM OrderDetails  
WHERE OrderId = : NEW.OrderId AND ROWNUM = 1);  
    SELECT BranchRating INTO OldRating FROM Branch WHERE BranchId = BrId;  
    -- Calculate the new average rating  
    SELECT AVG(O.OrderRating) INTO AvgRating  
    FROM OrderDetails D JOIN Orders O ON D.OrderId = O.OrderId  
    WHERE D.ItemId IN (SELECT ItemId FROM Item WHERE BranchId = BrId);  
    -- Update the branch rating using an autonomous transaction  
    COMMIT; -- Commit the current transaction  
    BEGIN  
        -- Start a new transaction  
        UPDATE Branch  
        SET BranchRating = AvgRating  
        WHERE BranchId = BrId;  
        DBMS_OUTPUT.PUT_LINE('Old Rating : ' || OldRating);  
        DBMS_OUTPUT.PUT_LINE('New Rating : ' || AvgRating);
```

```
COMMIT;  
END;  
END;
```

## Working:

```
UPDATE Orders  
SET OrderRating = 4.1  
WHERE OrderId = 915
```

Results Explain Describe Saved SQL

Old Rating : 4.5  
New Rating : 4.3

1 row(s) updated.

## Function:

### 1- calculate the total cost of an order

```
CREATE OR REPLACE FUNCTION CalculateTotalOrderCost(p_order_id Orders.OrderID%TYPE)  
RETURN NUMBER IS  
v_total_cost NUMBER := 0;  
delivery_fee NUMBER:=0;  
BEGIN  
SELECT SUM(ItemSize.ItemPrice * OrderDetails.ItemQuantity) INTO v_total_cost FROM OrderDetails  
JOIN ItemSize ON OrderDetails.ItemID = ItemSize.ItemID and OrderDetails.ordereditemsz = ItemSize.itemsize  
|| WHERE OrderDetails.OrderID = p_order_id;  
SELECT orders.deliveryfee INTO delivery_fee FROM Orders where orderid= p_order_id;  
RETURN v_total_cost + delivery_fee;  
END;
```

Results Explain Describe Saved SQL History

Function created.

0.00 seconds

## Working:

```
select orderid,'Rs ' || CalculateTotalOrderCost(orderid) totalordercost from orders;
```

**Results** Explain Describe Saved SQL History

ORDERID	TOTALORDERCOST
901	Rs 1950
902	Rs 3600
903	Rs 1300
904	Rs 2030
905	Rs 1570
906	Rs 1210
907	Rs 690
908	Rs 810
909	Rs 2470
910	Rs 2780
More than 10 rows available. Increase rows selector to view more rows.	

10 rows returned in 0.01 seconds [Download](#)

## 2- Function that calculates the number of orders in a branch of a restaurant.

```
CREATE OR REPLACE FUNCTION GetTotalOrdersAtBranch(p_branch_id IN Branch.BranchID%TYPE)
RETURN NUMBER
IS
    v_total_orders NUMBER := 0;
BEGIN
    SELECT COUNT(*)
    INTO v_total_orders
    FROM Orders
    WHERE OrderID IN (
        SELECT OrderID
        FROM OrderDetails
        WHERE ItemID IN (
            SELECT ItemID
            FROM Item
            WHERE BranchID = p_branch_id
        )
    );
    RETURN v_total_orders;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN NULL;
END;
```

**Results** Explain Describe Saved SQL History

Function created.

## Working:

Autocommit Rows 10 Save Run

```
select branchid,branchname,GetTotalOrdersAtBranch(branchid) totalorders,branch.restid,restname resturantname from branch join restaurant on branch.restid=restaurant.restid;
```

Results Explain Describe Saved SQL History

BRANCHID	BRANCHNAME	TOTALORDERS	RESTID	RESTURANTNAME
701	Cafe Euphoria Lahore 1	4	601	Cafe Euphoria
702	Cafe Euphoria Lahore 2	4	601	Cafe Euphoria
703	Jalal Sons Lahore	5	602	Jalal Sons
704	Jalal Sons Karachi	7	602	Jalal Sons
705	Makhibiz Peshawar	1	603	Makhibiz Kebab Corner
706	Malmo Islamabad 1	1	605	Malmo Bakers & Sweets
707	Malmo Islamabad 2	0	605	Malmo Bakers & Sweets
708	Malmo Lahore 1	3	605	Malmo Bakers & Sweets
709	Malmo Lahore 2	1	605	Malmo Bakers & Sweets
710	Malmo Karachi	1	605	Malmo Bakers & Sweets
More than 10 rows available. Increase rows selector to view more rows.				

10 rows returned in 0.02 seconds [Download](#)

## 8- Relational data model

