Information Technology Institute

Restaurant Management System

By:

Ali Elsayed Sherif

Omar Ragi Fouaad

Fares Youssef Ahmed

Mohamed Hossam Eldin

Date Integration and Visualization

Under The Supervision of

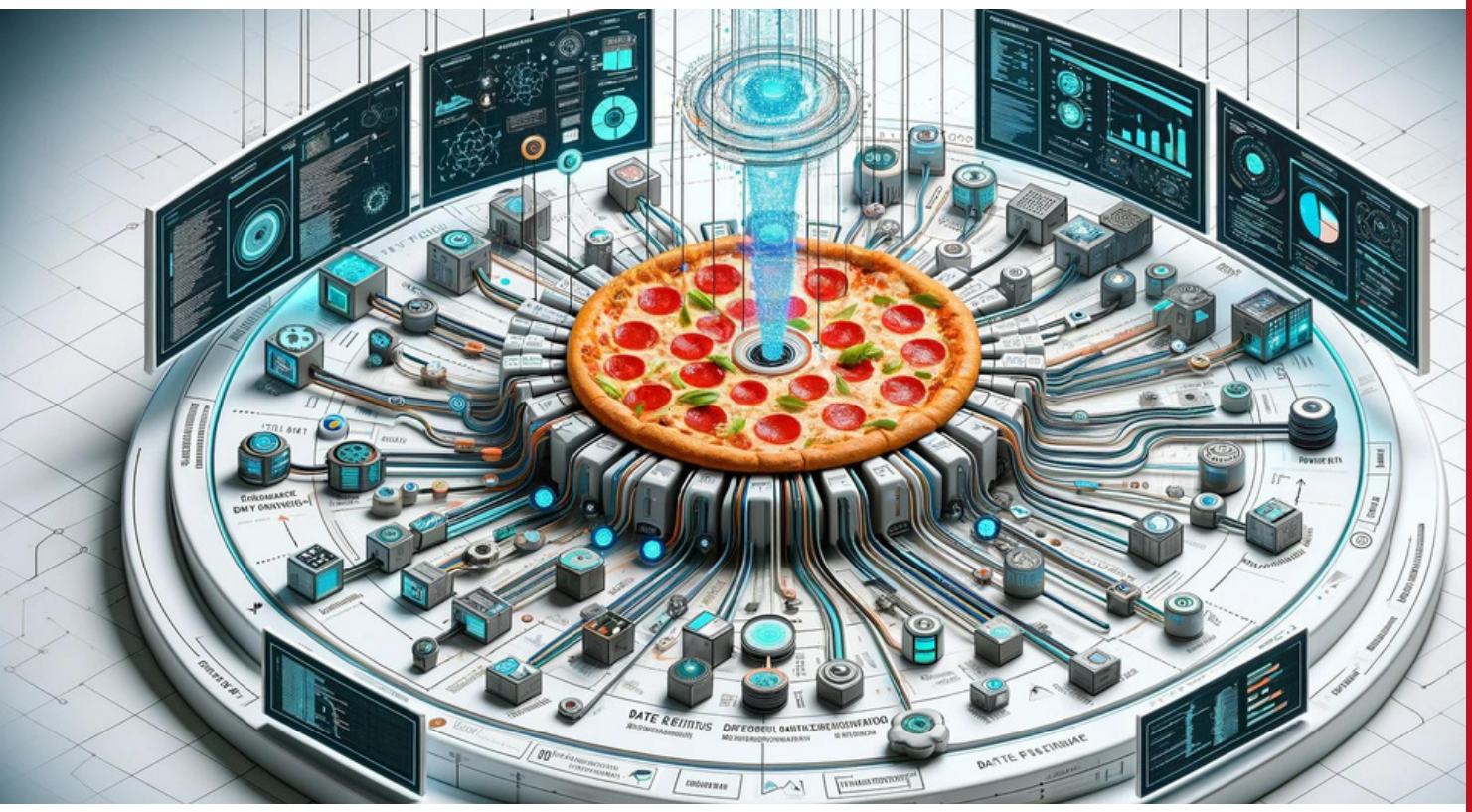
Eng. Rana Dahish

April, 2024

Information Technology Institute

APRIIL, 2024

Restaurant Management System



Documented By

Ali Elsayed Sherif

Omar Ragi Fouaad

Fares Youssef Ahmed

Mohamed Hossam Eldin

INTODUCTION
ABOUT OUR PROJECT

ABOUT THE PROJECT

The project is a restaurant management system designed to enhance customer service and streamline restaurant operations. It allows customers to reserve tables, place personalized or off-menu orders, choose between delivery and pickup, and assists restaurant staff in managing orders and table reservations to prevent overbooking. This system aims to optimize operations and improve customer satisfaction, ultimately increasing business efficiency and customer loyalty.

PROJECT VISION

The vision for the project is to create a unified system that speeds up restaurant operations while enhancing customer satisfaction and loyalty. It seeks to deliver a seamless customer experience through efficient table reservations, flexible order management, and timely delivery or pickup options. By providing a centralized repository for data analysis and reporting, the system aims to help restaurants make informed business decisions and foster customer engagement.

PROJECT MISSION

- Enhance Customer Satisfaction: Improve the quality of service to boost customer satisfaction and loyalty.
- Streamline Restaurant Operations: Implement a comprehensive system that manages table reservations, order handling, and other operational tasks.
- Provide Flexible Order Options: Allow customers to place personalized or off-menu orders, with delivery or pickup choices.
- Optimize Inventory Management: Maintain a balance between having enough stock to meet demand while avoiding excessive inventory levels.
- Support Employee Management: Create a positive work environment by setting clear goals, investing in employee growth, and providing the necessary tools for success.
- Utilize Data Warehousing: Employ a centralized data warehouse for complex analysis and reporting to inform business decisions.
- Foster Customer Engagement: Develop effective strategies to enhance customer relationships and increase customer loyalty through improved service and experience.

DATA SOURCES

GETTING DATA

These data sources are derived from a real-world restaurant, Pizza Pan. After analyzing the original dataset and understanding the relationships between various data points, these specific source tables were selected. This careful selection ensures that the data used in this integration project is both relevant and effective in representing Pizza Pan's operations. The chosen tables reflect key aspects of the restaurant's business model, including customer interactions, inventory management, order processing, and supplier relationships. By focusing on these essential areas, the project aims to create a robust integration framework tailored to Pizza Pan's unique operational needs

SOURCE TABLES

Our data sources for the project consist of various tables that capture key elements of a business's operational and customer-facing aspects.

Here's a brief overview of each table and its relevance to the integration project:

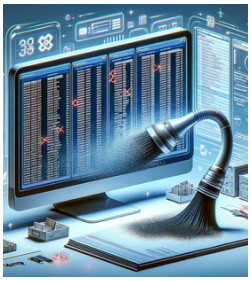
1. Category: Defines item types or groups.
2. Customer: Holds customer details.
3. Customer Title: Represents customer titles or designations.
4. Inventory: Tracks stock levels and item details.
5. Kitchen Type: Identifies different kitchen styles or sections.
6. Menu-Sales-Item: Lists items on a menu with prices.
7. Orders: Captures customer orders and related data.
8. Order-Details: Provides specifics on order items.
9. Store-Item: Represents stock in a store.
10. Supplier: Stores information about suppliers.
11. Supplier-Store-Item: Links suppliers to specific store items.
12. Void: Contains data on voided transactions.
13. Void-Details: Details about voided items or orders.
14. Zone: Defines operational areas or business sections.

DATA TRANSFORMATION

DENORMALIZATION

To build an effective data warehouse, I've denormalized several tables to streamline data processing and simplify analytics. Specifically:

- The Orders and Order-Details tables were merged to form a single table of orders.
- The Void and Void-Details tables were combined to create a unified dataset containing all information related to voided transactions.
- The tables for Supplier, Store-Item, and Supplier-Store-Item were integrated into a single source to provide a holistic view of the relationship between suppliers and store items.



DATA CLEANING

Customers: Phone-number

[Removing Uncompleted numbers and removing text inserted in “Phone-number”]

DATA FORMATTING

Generating date dimension and its all attributes from one input date-time such as date, year, Quarter, Month-num, month-name, Day-of-month, Day-of-week and Day-name

DATA DEDUPLICATION

Applied on all dimensions to get distinct PK values for each dimension

DWH MODELLING

DWH MODELLING TABLES

FACT TABLES

Orders_Fact: Contains information about orders, including date, customer, menu item, quantity, price, and total.
Foreign keys (FK): Date_Time_Key, Customer_ID, Zone_ID, Menu_Item_ID, Branch_ID.

Void_Fact: Contains information about voided orders, including menu item, customer, quantity, price, and total.
FK: Date_Time_Key, Menu_Item_ID, Customer_ID, Zone_ID, Branch_ID.

Inventory_Fact: Contains inventory information, including actual and expected quantities, item prices, and differences.
FK: Date_Time_Key, Inventory_ID, Branch_ID, Warehouse_ID, Storeitem_ID.

DIMENSIONS TABLES:

Dim_Date: Contains date-related information such as year, quarter, month, and day attributes.
Primary key (PK): Date_Time_Key.

Dim_Items: Contains information about menu items, including name, price, and category.
PK: Menu_Item_ID.

Dim_Customers: Contains customer information like name, phone number, and gender.
PK: Customer_ID.

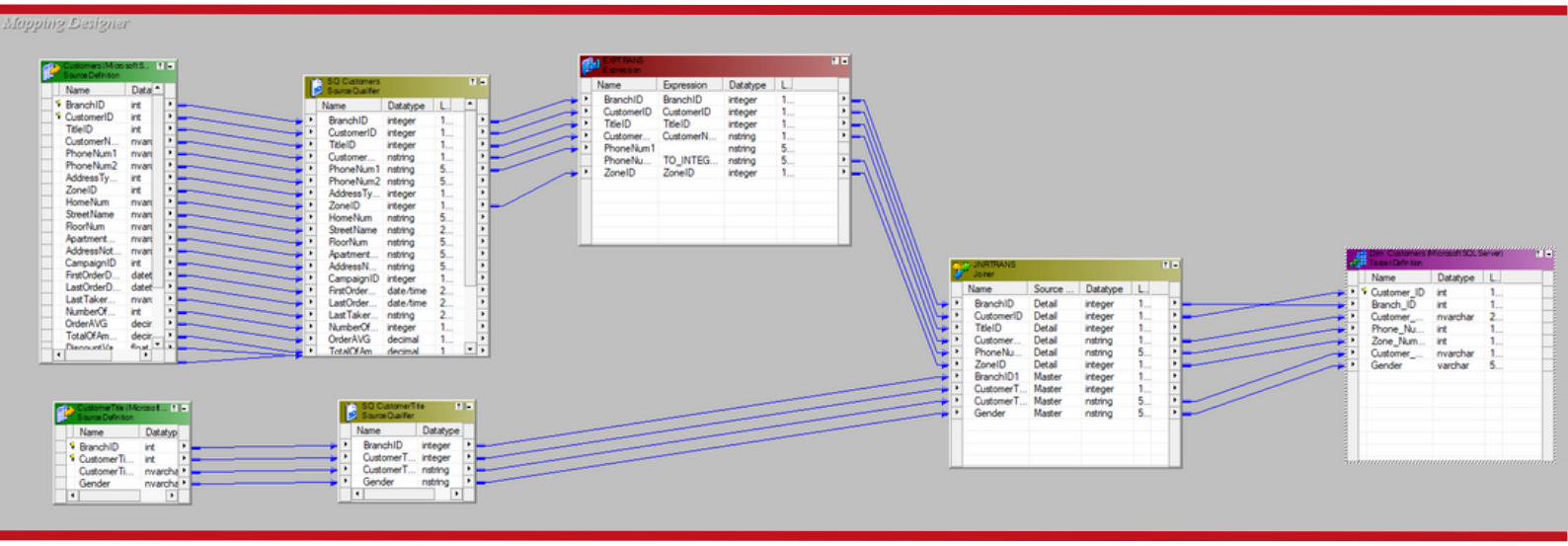
Dim_Zone: Contains information about delivery zones, including name and delivery fees.
PK: Zone_ID.

RELATIONSHIPS

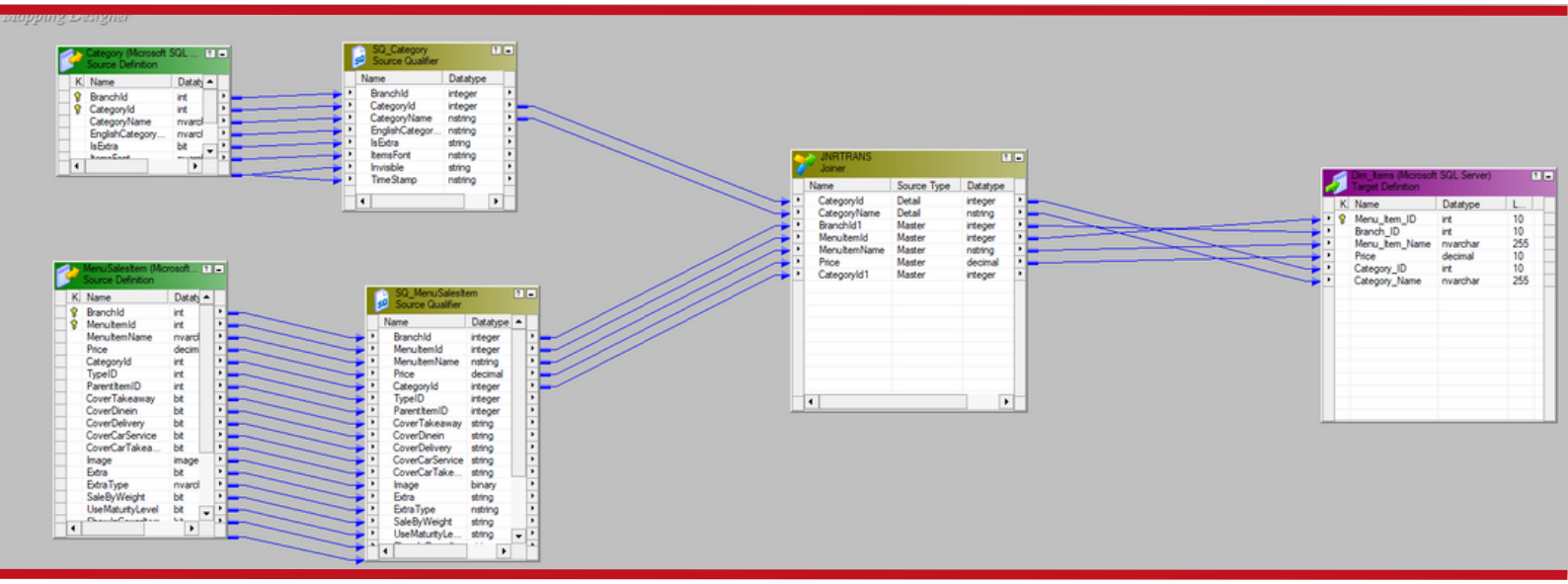
- Orders_Fact is related to Dim_Date through Date_Time_Key.
- Orders_Fact is related to Dim_Items through Menu_Item_ID.
- Orders_Fact is related to Dim_Customers through Customer_ID.
- Orders_Fact is related to Dim_Zone through Zone_ID.
- Void_Fact is related to Dim_Date through Date_Time_Key.
- Void_Fact is related to Dim_Items through Menu_Item_ID.
- Void_Fact is related to Dim_Customers through Customer_ID.
- Void_Fact is related to Dim_Zone through Zone_ID.
- Inventory_Fact is related to Dim_Date through Date_Time_Key.
- Inventory_Fact is related to Dim_Items through Storeitem_ID.

ETL INFORMATICA
DATA PIPELINE

DIM-CUSTOMERS

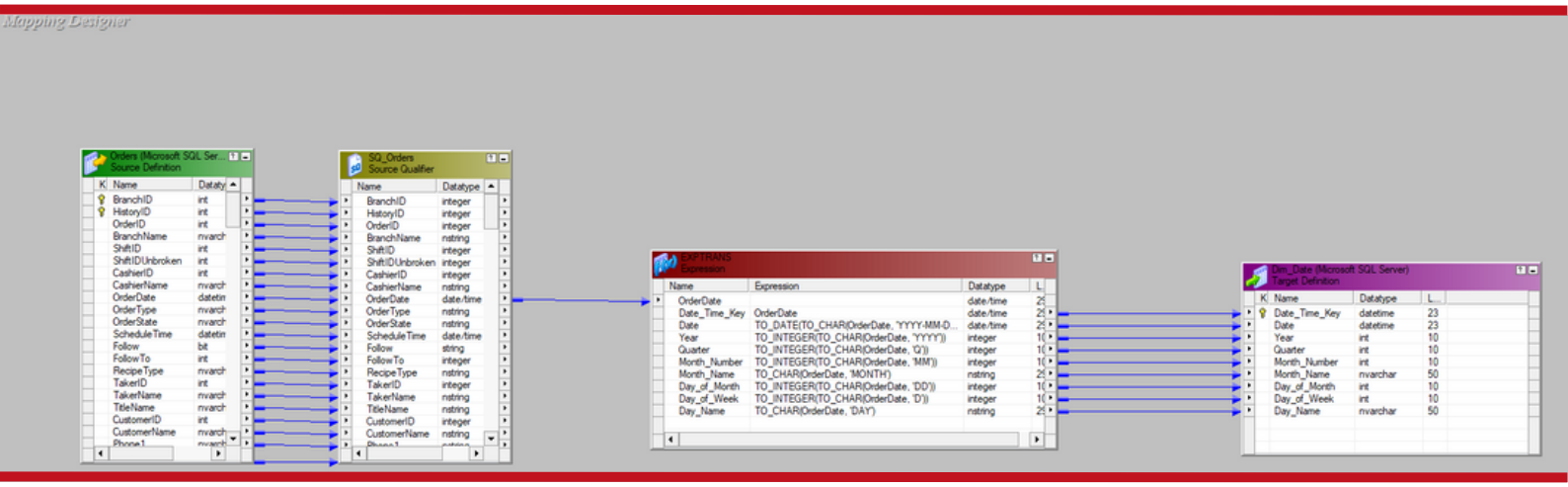


DIM-ITEMS

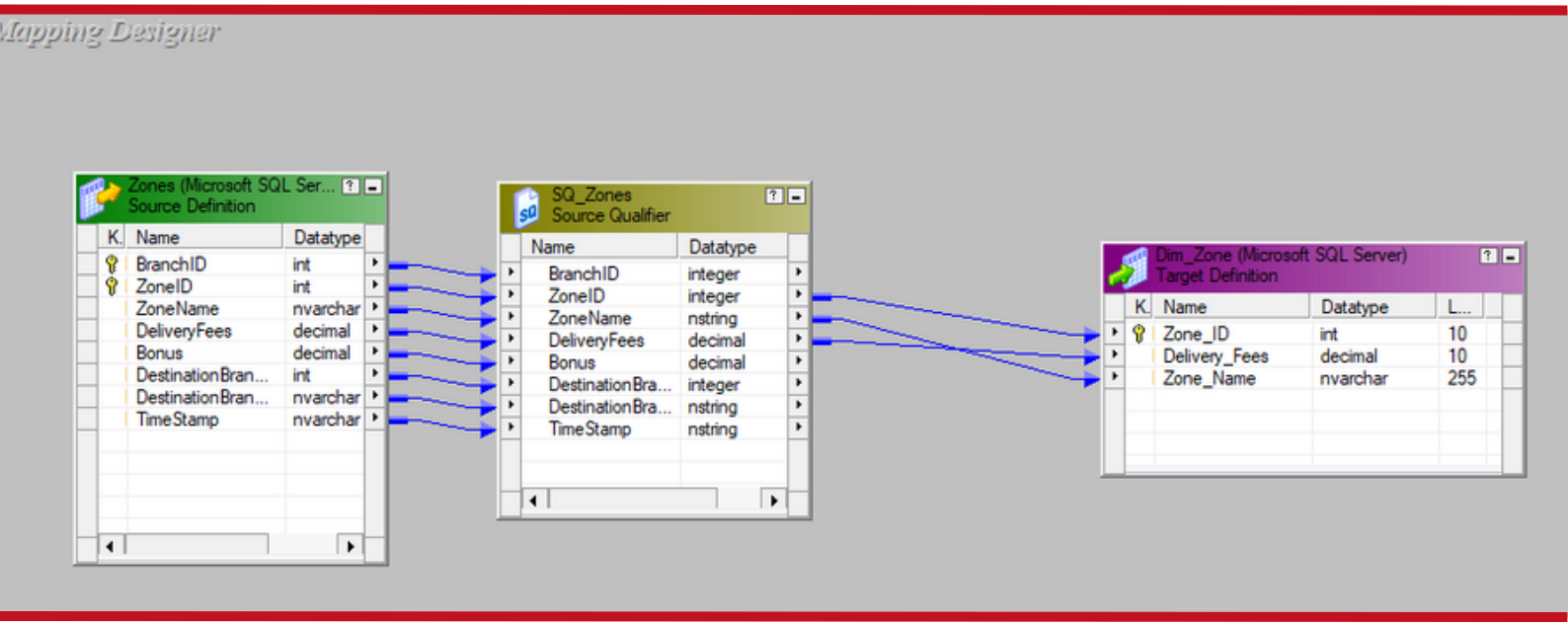


ETL INFORMATICA
DATA PIPELINE

DIM-DATE

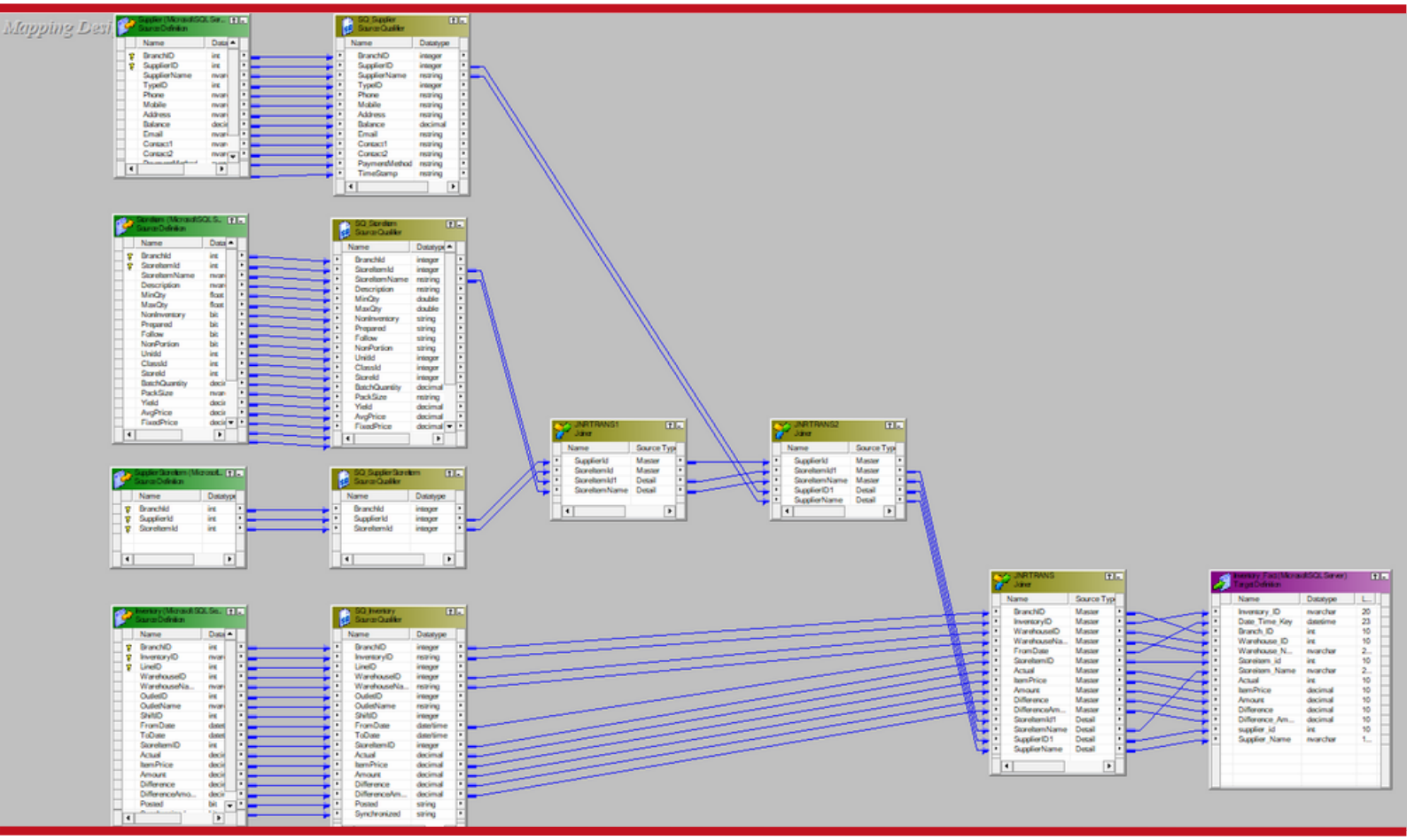


DIM-ZONES



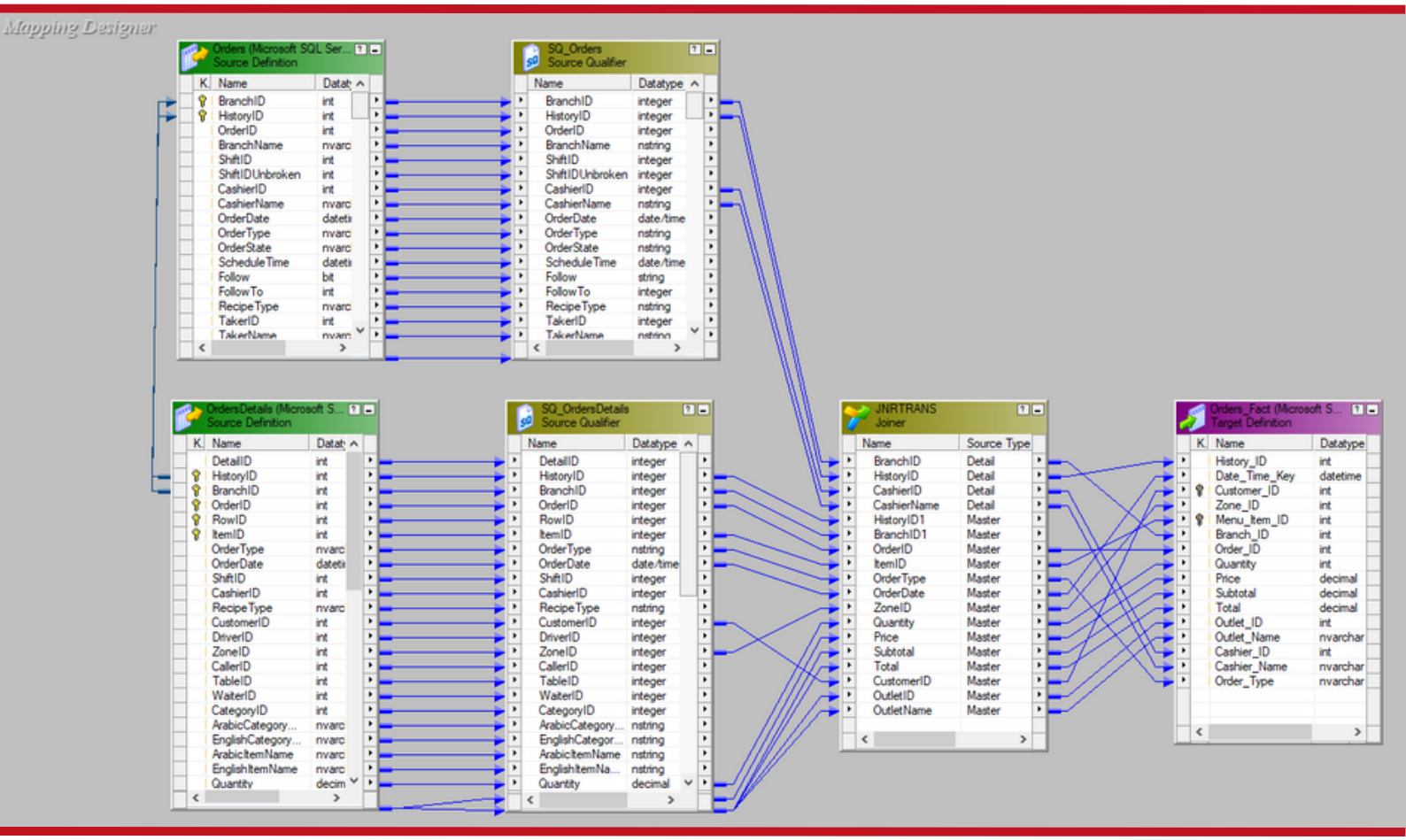
ETL INFORMATICA
DATA PIPELINE

FACT-INVENTORY



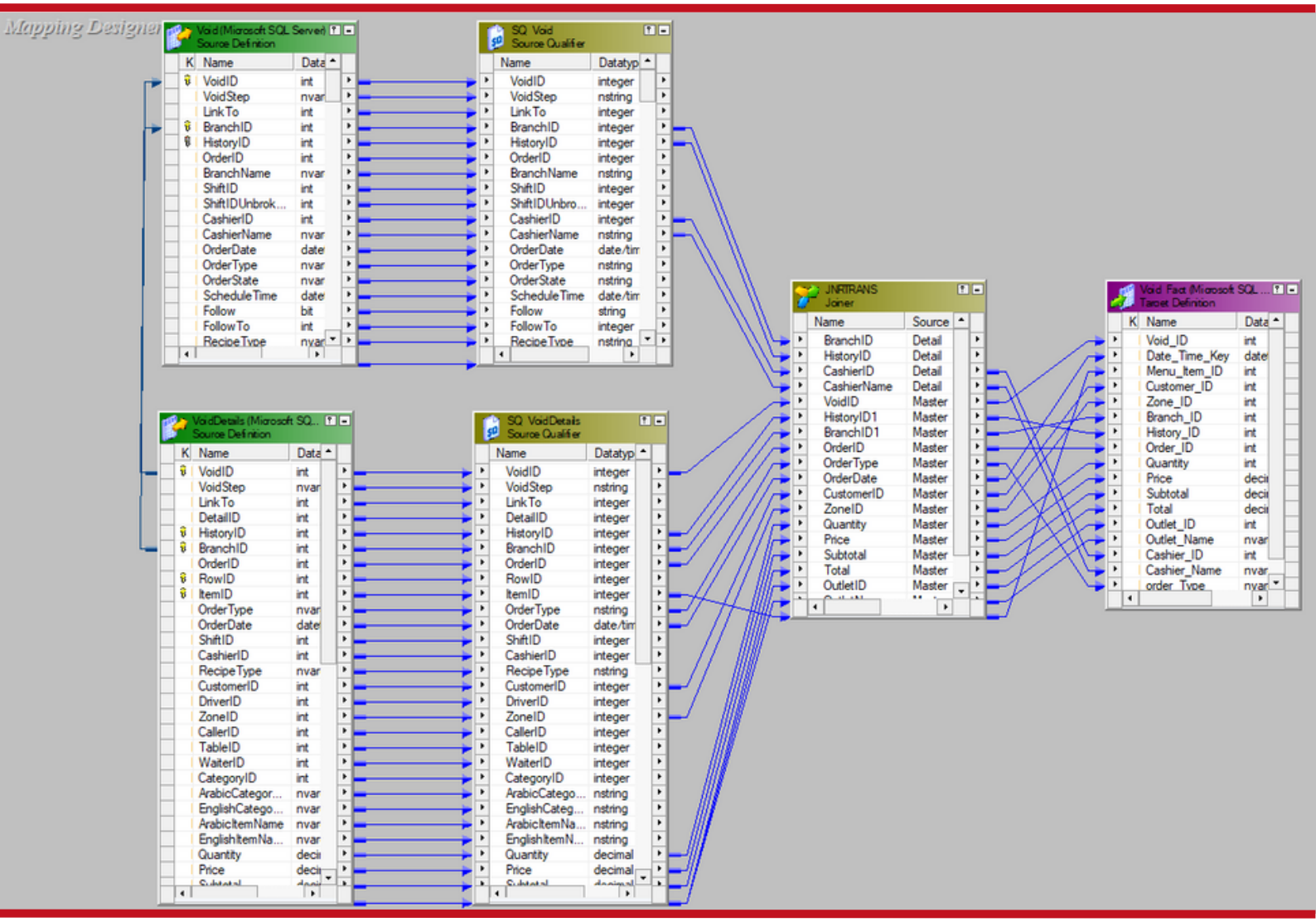
ETL INFORMATICA
DATA PIPELINE

FACT-ORDERS



ETL INFORMATICA
DATA PIPELINE

FACT-VOIDS



APPENDIX
SQL QUERIES

QUERIES BUILDING DWH

```
create database DWH_Pizza_Pan;
use DWH_Pizza_Pan;
go

CREATE TABLE Dim_Customers (
    Customer_ID INT PRIMARY KEY,
    Branch_ID INT,
    Customer_Name NVARCHAR(255),
    Phone_Number NVARCHAR(50),
    Zone_Number NVARCHAR(50),
    Customer_Title NVARCHAR(50),
    Gender NVARCHAR(50)

);

CREATE TABLE Dim_Zone (
    Zone_ID INT PRIMARY KEY,
    Delivery_Fees DECIMAL(10, 2),
    Zone_Name NVARCHAR(255)

);

CREATE TABLE Dim_Items (
    Menu_Item_ID INT PRIMARY KEY,
    Branch_ID INT,
    Menu_Item_Name NVARCHAR(255),
    Price DECIMAL(10, 2),
    Category_ID INT,
    Category_Name NVARCHAR(255)

);
            CREATE TABLE Dim_Date (
                Date_Time_Key DATETIME PRIMARY KEY,
                Date DATE,
                Year INT,
                Quarter INT,
                Month_Number INT,
                Month_Name NVARCHAR(50),
                Day_of_Month INT,
                Day_of_Week INT,
                Day_Name NVARCHAR(50)

            );
```

APPENDIX
SQL QUERIES

QUERIES BUILDING DWH

```
CREATE TABLE Orders_Fact (
    History_ID INT,
    Date_Time_Key DATETIME,
    Customer_ID INT,
    Zone_ID INT,
    Menu_Item_ID INT,
    Branch_ID INT,
    Order_ID INT,
    Quantity INT,
    Price DECIMAL(10, 2),
    Subtotal DECIMAL(10, 2),
    Total DECIMAL(10, 2),
    Outlet_ID INT,
    Outlet_Name NVARCHAR(255),
    Cashier_ID INT,
    Cashier_Name NVARCHAR(255),
    FOREIGN KEY (Date_Time_Key) REFERENCES Dim_Date(Date_Time_Key),
    FOREIGN KEY (Customer_ID) REFERENCES Dim_Customers(Customer_ID),
    FOREIGN KEY (Zone_ID) REFERENCES Dim_Zone(Zone_ID),
    FOREIGN KEY (Menu_Item_ID) REFERENCES Dim_Items(Menu_Item_ID),

);
```

```
CREATE TABLE Inventory_Fact (
    Inventory_ID INT,
    Date_Time_Key DATETIME,
    Branch_ID INT,
    Warehouse_ID INT,
    Warehouse_Name NVARCHAR(255),
    Storeitem_id INT,
    Storeitem_Name NVARCHAR(255),
    Actual INT,
    ItemPrice DECIMAL(10, 2),
    Amount DECIMAL(10, 2),
    Difference DECIMAL(10, 2),
    Difference_Amount DECIMAL(10, 2),
    FOREIGN KEY (Date_Time_Key) REFERENCES Dim_Date(Date_Time_Key),

);
```

APPENDIX
SQL QUERIES

QUERIES BUILDING DWH

```
CREATE TABLE Void_Fact (
    Void_ID INT,
    Date_Time_Key DATETIME,
    Menu_Item_ID INT,
    Customer_ID INT,
    Zone_ID INT,
    Branch_ID INT,
    History_ID INT,
    Order_ID INT,
    Quantity INT,
    Price DECIMAL(10, 2),
    Subtotal DECIMAL(10, 2),
    Total DECIMAL(10, 2),
    Outlet_ID INT,
    Outlet_Name NVARCHAR(255),
    Cashier_ID INT,
    Cashier_Name NVARCHAR(255),
    FOREIGN KEY (Date_Time_Key) REFERENCES Dim_Date(Date_Time_Key),
    FOREIGN KEY (Menu_Item_ID) REFERENCES Dim_Items(Menu_Item_ID),
    FOREIGN KEY (Customer_ID) REFERENCES Dim_Customers(Customer_ID),
    FOREIGN KEY (Zone_ID) REFERENCES Dim_Zone(Zone_ID),

);
```

This SQL query retrieves the primary key for each table in the database

```
SELECT t.name AS TableName, c.name AS PrimaryKeyColumn
FROM sys.tables t INNER JOIN  sys.indexes i
ON t.object_id = i.object_id
INNER JOIN sys.index_columns ic
ON i.object_id = ic.object_id AND i.index_id = ic.index_id
INNER JOIN sys.columns c
ON ic.object_id = c.object_id AND ic.column_id = c.column_id
WHERE i.is_primary_key = 1
ORDER BY t.name, ic.index_column_id;
```

APPENDIX
SQL QUERIES

BI QUERIES

CALCULATE THE MONTHLY DIFFERENCE OF PROFIT

```
with sub_query as
(
    select CONVERT(DATE, CONCAT(YEAR(orderdate), '-',
    MONTH(orderdate) , '-01') , 23) as month_year , grandtotal
    from Orders
    -- This sub query to avoid us
) ,

extract_date as
(    select month_year as month_year, sum(grandtotal)
        over(partition by month_year order by month_year) as monthly_sales
            from sub_query
) ,

distinct_query as
(
   /*This subquery to get the distinct values
   of month year to avoid having duplicate rows in the result*/
    select distinct month_year , monthly_sales
    from extract_date
)

select month_year , monthly_sales , lag(monthly_sales , 1)
    over (order by month_year) as previous_month_sales,
    case
        when lag(monthly_sales, 1)
        over (order by month_year) is null then 0 -- handle the first month
        else round(((monthly_sales - lag(monthly_sales, 1)
        over (order by month_year)) / lag(monthly_sales, 1)
        over (order by month_year)) * 100, 2)
    end as "sales_percentage_difference%"
from distinct_query;
```

APPENDIX
SQL QUERIES

BI QUERIES

MONETARY MODEL FOR CUSTOMER SEGMENTATION

```
with sub_query as
(
    select distinct CustomerID ,
    CustomerName ,
    LastOrderDate as last_purchase ,
    LastOrderDate ,
    NumberOfOrder as Frequency ,
    TotalOfAmount as Monetary
    from Customers
) ,
rfm_values as
(
    -- Sub query to calculate the Recency, Frequency and  Monetary
    SELECT
    DISTINCT CustomerID,
    CustomerName,
    datediff(day, last_purchase ,  max(LastOrderDate) OVER ()) AS Recency,
    Frequency,
    Monetary,
    (Frequency + Monetary) / 2 AS fm_average
FROM
    sub_query
) ,
rfm_scores as
```

```sql
(
    -- Sub query to calculate the Recency_score , Frequency_score , Monetary_score
    select distinct CustomerID ,  CustomerName , Recency , Frequency ,  Monetary ,
    ntile(5) over(order by Recency desc) as Recency_score , ntile(5) over(order by fm_average) as avg_fm_score
    from rfm_values
)
select distinct CustomerID ,  CustomerName , Recency , Frequency ,  Monetary , Recency_score , avg_fm_score ,
    case
        when Recency is null and AVG_FM_Score in (0 , 1 , 2) then 'Lost'
        when Recency_score = 5 and AVG_FM_Score in (5 , 4) then 'Champions'
        when Recency_score  = 4 and AVG_FM_Score = 5 then 'Champions'
        when Recency_score in (5,4) and AVG_FM_Score = 2 then 'Potential Loyalists'
        when Recency_score in (3,4) and AVG_FM_Score = 3 then 'Potential Loyalists'
        when Recency_score = 5 and AVG_FM_Score = 3 then 'Loyal Customers'
        when Recency_score = 4 and AVG_FM_Score = 4 then 'Loyal Customers'
        when Recency_score = 3 and AVG_FM_Score in (4 , 5) then 'Loyal Customers'
        when Recency_score = 5 and AVG_FM_Score = 1 then 'Recent Customers'
        when Recency_score in (4 , 3) and AVG_FM_Score = 1 then 'Promising'
        when Recency_score = 3 and AVG_FM_Score = 2 then 'Customers Needing Attention'
        when Recency_score = 2 and AVG_FM_Score in (3 , 2) then 'Customers Needing Attention'
        when Recency_score = 2 and AVG_FM_Score in (4 , 5) then 'At Risk'
        when Recency_score = 1 and AVG_FM_Score = 3 then 'At Risk'
        when Recency_score = 1 and AVG_FM_Score IN (5 , 4) then 'Cant Lose Them'
        when Recency_score = 1 and AVG_FM_Score = 2 then 'Hibernating'
        /*This value was not provided in the given table but was set
        in the logic, as not adding it will allow some rows to have empty values.*/
        when Recency_score = 2 and AVG_FM_Score  = 1 then 'Lost'
        when Recency_score = 1 and AVG_FM_Score  = 1 then 'Lost'
    end as cust_segment
from rfm_scores;
```

APPENDIX
SQL QUERIES

BI QUERIES

PERCENTAGE OF ORDERS MODIFIED

```
with void_cnt as
(
    select count (distinct historyid) as void_count
    from Void
), orders_cnt as
(
    select count(historyid) as orders_count
    from Orders
)
select round (CAST (void_count AS FLOAT) / orders_count , 4)
* 100 as 'percentage_of_orders_modified%'
from void_cnt , orders_cnt;
```

SALES GENERATED PER CASHIER

```
select CashierID , CashierName , sum(GrandTotal) as total
from Orders
group by CashierID , CashierName
order by total desc;
```

APPENDIX
SQL QUERIES

BI QUERIES

CUSTOMER REPEAT RATE

```sql
select round(cast(count(*) AS FLOAT) / (select count(*) from Customers) , 4)
* 100 as Customer_repeat_rate
from Customers
where FirstOrderDate != LastOrderDate;
```

CUSTOMER RETENTION RATE

```sql
WITH CustomerRetentionData_2017 AS (
    SELECT COUNT(CustomerID) AS TotalCustomers_2017,
        SUM(CASE WHEN YEAR(FirstOrderDate) = 2017 THEN 1 ELSE 0 END) AS NewCustomers_2017,
        SUM(CASE WHEN YEAR(LastOrderDate) > 2017 THEN 1 ELSE 0 END) AS ReturningCustomers_2017
    FROM Customers
    WHERE YEAR(FirstOrderDate) <= 2017
) , CustomerRetentionData_2018 as
(
    SELECT COUNT(CustomerID) AS TotalCustomers_2018,
        SUM(CASE WHEN YEAR(FirstOrderDate) = 2018 THEN 1 ELSE 0 END) AS NewCustomers_2018,
        SUM(CASE WHEN YEAR(LastOrderDate) > 2018 THEN 1 ELSE 0 END) AS ReturningCustomers_2018
    FROM Customers
    WHERE YEAR(FirstOrderDate) <= 2018
) , CustomerRetentionData_2019 as
(
    SELECT COUNT(CustomerID) AS TotalCustomers_2019 ,
        SUM(CASE WHEN YEAR(FirstOrderDate) = 2019 THEN 1 ELSE 0 END) AS NewCustomers_2019 ,
        SUM(CASE WHEN YEAR(LastOrderDate) > 2019 THEN 1 ELSE 0 END) AS ReturningCustomers_2019
    FROM Customers
    WHERE YEAR(FirstOrderDate) <= 2019
) , CustomerRetentionData_2020 as
(
    SELECT COUNT(CustomerID) AS TotalCustomers_2020,
        SUM(CASE WHEN YEAR(FirstOrderDate) = 2020 THEN 1 ELSE 0 END) AS NewCustomers_2020,
        SUM(CASE WHEN YEAR(LastOrderDate) > 2020 THEN 1 ELSE 0 END) AS ReturningCustomers_2020
    FROM Customers
    WHERE YEAR(FirstOrderDate) <= 2020
)
```

```sql
SELECT '2017' as Year , TotalCustomers_2017 as TotalCustomers ,
NewCustomers_2017 as NewCustomers,
    ReturningCustomers_2017 as ReturningCustomers ,
    round(((ReturningCustomers_2017 * 1.0) / NewCustomers_2017) * 100 , 2)
    AS CustomerRetentionRate
    from CustomerRetentionData_2017
    union
select '2018' as Year , TotalCustomers_2018,
NewCustomers_2018, ReturningCustomers_2018,
    round(((ReturningCustomers_2018 * 1.0) / NewCustomers_2018) * 100 , 2)
    AS CustomerRetentionRate_2018
    from CustomerRetentionData_2018
    union
select '2019' as Year , TotalCustomers_2019, NewCustomers_2019,
ReturningCustomers_2019,
    round(((ReturningCustomers_2019 * 1.0) / NewCustomers_2019) * 100 , 2)
    AS CustomerRetentionRate_2019
    from CustomerRetentionData_2019
    union
select '2020' as Year , TotalCustomers_2020, NewCustomers_2020,
ReturningCustomers_2020,
    round(((ReturningCustomers_2020 * 1.0) / NewCustomers_2020) * 100 , 2)
    AS CustomerRetentionRate_2020
    from CustomerRetentionData_2020;
```

APPENDIX
SQL QUERIES

BI QUERIES

NUMBER OF CUSTOMERS VISITS AND NUMBER OF ORDERS MADE PER MONTH

```
with join_query as
(
    select orderID , convert(date , concat(year(orderdate) , '-'
    ,month(orderdate) , '-01') , 23)  as OrderDate ,
        o.grandtotal , c.CustomerID , c.CustomerName
    from Orders o left outer join Customers c
    on o.customerid = c.CustomerID
) , num_query as
(
    select OrderDate , count(OrderDate) as NumberOfOrders ,
    count(distinct(customerID)) as NumberOfCustomers
    from join_query
    group by OrderDate
) , prev_query as
(
    select OrderDate , NumberOfOrders , lag(NumberOfOrders , 1)
    over(order by OrderDate) as PrevOrd , NumberOfCustomers ,
        lag(NumberOfCustomers , 1)
        over(order by OrderDate) as PrevCust
    from num_query
)
select orderdate , numberoforders , prevord ,
    case
        when prevord is null then 0  -- handle the first month
        else round(((numberoforders * 1.00 - prevord * 1.00)
        / prevord * 1.00) , 2) * 100
    end as 'PercentageOrdersDiff%',
    numberofcustomers ,
    prevcust ,
        case
        when prevcust is null then 0  -- handle the first month
        else round(((numberofcustomers * 1.00 - prevcust * 1.00)
        / prevcust * 1.00) , 2) * 100
    end as 'PercentageCustomersDiff%'
from prev_query;
```

APPENDIX
SQL QUERIES

BI QUERIES

CUSTOMER CHURN RATE

```
with join_query as
(
    -- To join the orders with the customers tables

    select orderID , CONVERT(date , concat(year(orderdate) , '-'
    , month(orderdate) , '-01')) as OrderFirstDay ,
        o.grandtotal , c.customerID , c.CustomerName
        , FirstOrderDate , LastOrderDate
    from Orders o left outer join Customers c
    on o.customerid = c.CustomerID
) , status_query as
(
    -- To know if the customer is dropped or retained

    select orderID , OrderFirstDay , EOMONTH(OrderFirstDay)
    as OrderLastDay , grandtotal , customerID , CustomerName ,
        FirstOrderDate , LastOrderDate ,
        (case
        when (FirstOrderDate is null) or
        (LastOrderDate is null) then 'NAN'
        when (FirstOrderDate >= OrderFirstDay) and
        (LastOrderDate <= EOMONTH(OrderFirstDay)) then 'Dropped'
        else 'Retained'
        end) as Status
    from join_query
) , churn_query as
(
    select concat(year(OrderFirstDay) , '-'
    , month(OrderFirstDay)) as Month_Year,
        sum(case when status = 'Dropped'
        then 1 else 0 end) as count_of_dropped ,
        sum(case when status = 'Retained'
        then 1 else 0 end) as count_of_retained
    from status_query
    group by OrderFirstDay
)

  -- To calculate the churn rate
  select Month_Year , count_of_dropped , count_of_retained ,
      CONVERT(decimal(10 , 3), ROUND((count_of_dropped * 1.00)
      / (count_of_retained * 1.00) , 4) * 100) as Churn_Rate
  from churn_query
  order by convert(date , concat(Month_Year , '-01'));
```

APPENDIX
SQL QUERIES

BI QUERIES

AVERAGE COVERS (NUMBER OF ORDERS / NUMBER OF WORKING DAYS)

```
with sub_query as
(
    select OrderID , convert(date , concat(year(orderdate) ,
    '-' , month(orderdate) , '-' , day(orderdate)))
    as  OrderDate , grandtotal
    from Orders
), sub_query1 as
(
    select OrderDate , count(OrderID) as MealsServed
    , count(*)
    over(partition by concat(year(orderdate) , '-' , month(orderdate))) as NumOfDays
    from sub_query
    group by OrderDate
)
select format(OrderDate, 'yyyy-MM') AS MonthYear,
sum(MealsServed) / max(NumOfDays) AS AvgMealsPerDay
from sub_query1
group by format(OrderDate, 'yyyy-MM')
order by format(OrderDate, 'yyyy-MM');
```

THANK YOU FOR YOUR APRECIATE