

**SVKM's NMIMS**

**Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)**

**Computer Engineering Department (B Tech CSE/CSBS Sem IV/BTI Sem VIII/MBA.Tech-IV)**

**Database Management System**

**Project Report**

Program	B.Tech	
Semester	4th	
Name of the Project:	Zomato Data Analytics Using Sql and Power Bi	
Details of Project Members		
Batch	Roll No.	Name
B1	E034	Abizer Masavi
B1	E037	Subham Mohapatra
B1	E039	Arzaan Mulla
Date of Submission:		

**Contribution of each project Members:**

Roll No.	Name:	Contribution
E034	Abizer Masavi	Data analysis and preprocessing, Normalization, Query development, PowerBI
E037	Subham Mohapatra	Data analysis and preprocessing, Designing architecture, SQL implementation, PowerBI
E039	Arzaan Mulla	Report + Data cleaning, Data preprocessing, Normalization.

**Github link of your project:**

**Note:**

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004\_DBMSProject)
3. Submit all relevant files of your work ( Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

# **Project Report**

## **Zomato Data Analytics**

**by**

**Abizer Masavi, Roll number: E034**

**Subham Mohapatra, Roll number: E037**

**Arzaan Mulla, Roll number: E039**

**Course: DBMS**

**AY: 2024-25**

## Table of Contents

Sr no.	Topic	Page no.
1	Storyline	5
2	Components of Database Design	6
3	Entity Relationship Diagram	9
4	Relational Model	10
5	Normalization	13
6	SQL Queries	15
7	Project Demonstration	35
8	Self-learning beyond classroom	40
9	Learning from the project	42
10	Challenges faced	43
11	Conclusion	45

# I. Storyline

Picture a city that is vibrant where every meal ordered, every restaurant rated, and every new food craze contributes to the rich cultural fabric of the dining experience. Amidst all this busy life, the Zomato Data Analytics project is born—a journey to unravel the stories hidden in every data point, from the smoldering restaurant revenues to the unsaid ebbs and flows of the customer's hunger.

In essence, this project is about building a robust database that would be the center of Zomato activity. The system will record structured data from touchpoints: restaurant minute-to-minute performance logs, very fine-grained customer profiles, high-fidelity sales logs, and even minute-by-minute food trends feeds. With the inclusion of different sets of data, the database will not just contain information but ready it for heavy-duty analysis.

Operationally, the project is two-faceted. On the one hand, SQL will be used as data extract and transformation software. With well-crafted queries, the system will sift through layers of data—linking customer orders to demographic data, linking sales data to targeted time frames, and revealing concealed patterns in ordering habits. The rigorous data processing ensures accuracy and responsiveness in the outcomes.

Second, Power BI is the visualization hero. Restaurant owners simply have to sit down in front of an interactive dashboard where maps, graphs, and charts bring data to life. A restaurant owner can instantly be aware of what flies off the shelves when it is peak time, seasonal trends affect sales, and even be able to geographically map their regular customers. But analysts are able to go far deeper into complex sets of data in order to be able to predict trends, and management teams have at their fingertips strategic summaries placing before them opportunity for growth together with problems to be solved.

This is not a technical exercise in itself; it's empowering people. To the restaurant owner, it is replacing guesswork with data-driven decisions—changing menus, staffing, and offering targeted promotions through the utilization of real-time data. To the management, it offers a window into the pulse of the market, revealing shifts in consumer behavior and emerging food trends that can redefine Zomato's business plan.

Lastly, the Zomato Data Analytics project seeks to balance raw data and strategic decisions as harmoniously as possible. By very carefully building a database to monitor each detail of restaurant performance and customer action, the project makes each detail of data available with just the correct query. This is not just data in a bid to enhance operation efficiency but fuel innovation and growth in a rapidly moving food sector.

## **II. Components of Database Design**

### **Entities and Their Attributes-**

#### **Restaurant**

Attributes:

- restaurant\_id (Primary Key)
- name
- location
- rating
- cuisine type
- performance metrics

Overview:

The Restaurant entity contains all the necessary information about every restaurant. The restaurant\_id is the unique identifier.

#### **Users**

Attributes:

- user\_id (Primary Key)
- age
- gender
- occupation
- marital status

Overview:

This entity stores demographic data about the customers. Every user is identified uniquely by user\_id.

#### **Orders**

Attributes:

- order\_id (Primary Key)
- sales quantity
- sales amount
- order date
- r\_id (Foreign Key referencing Restaurant.restaurant\_id)
- user\_id (Foreign Key referencing Users.user\_id)

Overview:

The Orders table records information about every transaction such as what was sold, when it was sold, and associates the order with the restaurant and the customer.

## **Order Type**

Fields:

- order\_id (Primary Key and also a Foreign Key referencing Orders.order\_id)
- order type category (e.g., Dine-in, Delivery, or Takeaway)

Description:

This table categorizes orders into distinct types. As its primary key is the order\_id (which is also a foreign key), each order in the Orders table has one unique corresponding order type record.

## **Menu**

Fields:

- menu\_id (Primary Key)
- menu item information (e.g., item name, description)
- price
- r\_id (Foreign Key referencing Restaurant.restaurant\_id)
- f\_id (Foreign Key referencing Food.f\_id)

Description:

The Menu entity stores all the menu items of restaurants and their prices. A menu item is associated with a restaurant and is again categorized based on the food type.

## **Food**

Attributes:

- f\_id (Primary Key)
- food type indicator (indicating if an item is vegetarian or not)

Overview:

This entity specifies the food type, informing the system whether a specific item is vegetarian or non-vegetarian. The f\_id is referenced by the Menu table to specify the type of each menu item.

## **Relationships and Their Characteristics**

### **Restaurant and Orders**

Relationship:

- A restaurant may have several orders.
- One order is placed at one restaurant.

Cardinality:

- One-to-Many: One restaurant to several orders.

Participation:

- Orders: Mandatory (all orders should be linked to a restaurant).
- Restaurant: Optional in that a restaurant can have no orders at times (particularly if new or temporarily unused).

## **Users and Orders**

Relationship:

- There can be a lot of orders from a single user.
- One order corresponds to one user.

Cardinality:

- One-to-Many: One user to multiple orders.

Participation:

- Orders: Mandatory (an order has to belong to a registered user).
- Users: Optional (a user may not have ordered yet).

## **Orders and Order Type**

Relationship:

- There is one order type for each order.
- The Order Type class employs order\_id as both its primary key and its foreign key.

Cardinality:

- One-to-One: A single order is uniquely defined by one order type.

Participation:

- Orders: Each order is supposed to have a linked order type.
- Order Type: Required, since it relies on an order.

## **Restaurant and Menu**

Relationship:

- A restaurant has several menu items.
- A menu item is part of one restaurant.

Cardinality:

- One-to-Many: One restaurant to several menu items.

Participation:

- Menu: Optional (a menu may not be associated with a restaurant at all times, although normally it would be).
- Restaurant: Mandatory (each restaurant must be associated with a menu).

## **Food and Menu**

Relationship:

- One food type can be assigned to several menu items.
- Every menu item belongs to one food type.

Cardinality:

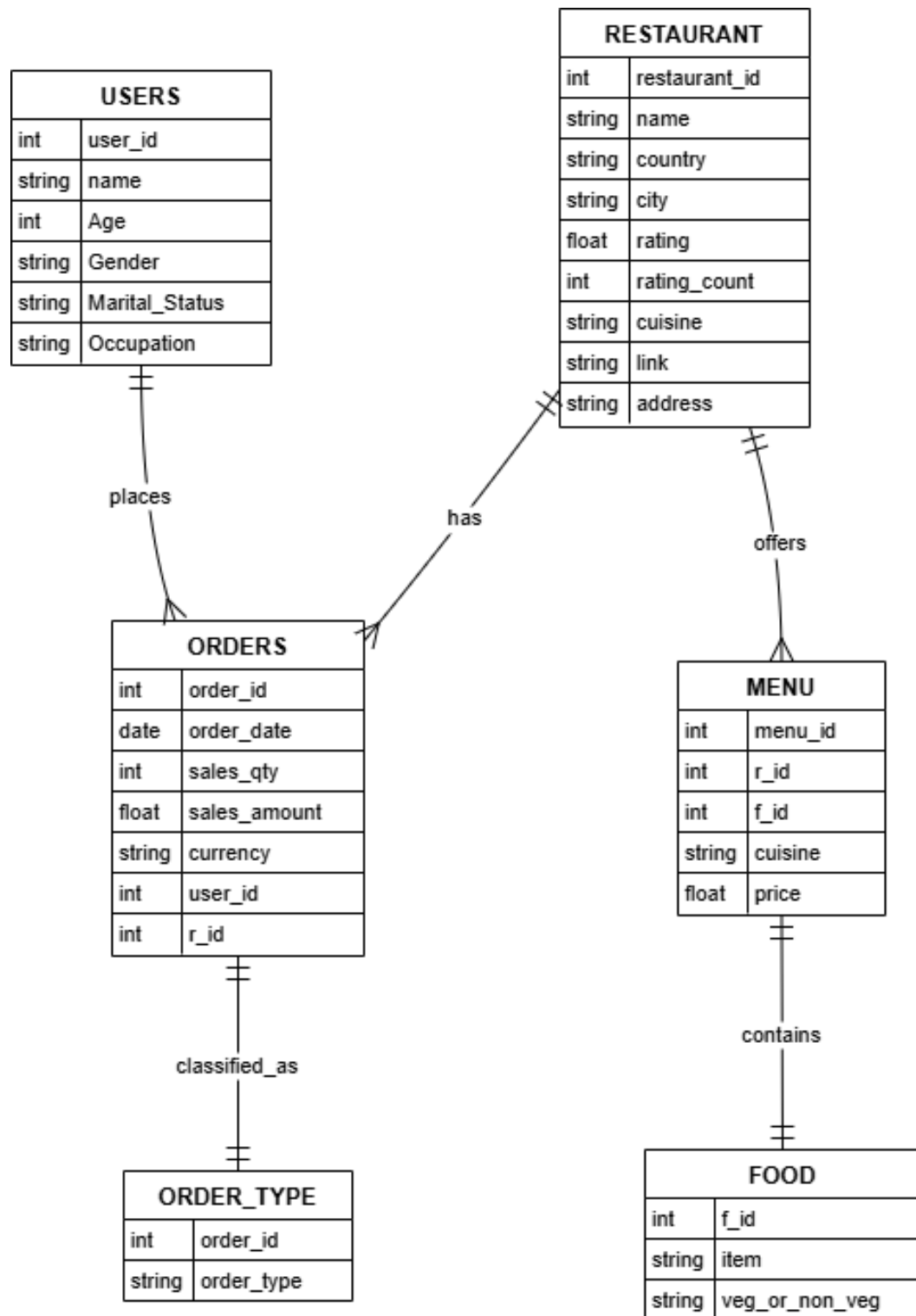
- One-to-Many: One food record to many menu items.

Participation:

- Menu: Mandatory (each menu item must be associated with a food type).



### III. Entity Relationship Diagram



# IV. Relational Model

## 1. USERS

**Table Name:** USERS

**Attributes:**

- **user\_id** (PK)
- name
- age
- gender
- marital\_status
- occupation

**Schema-**

```
USERS(  
  user_id    INT    PRIMARY KEY,  
  name       VARCHAR(100),  
  age        INT,  
  gender     VARCHAR(20),  
  marital_status VARCHAR(20),  
  occupation  VARCHAR(50)  
);
```

## 2. RESTAURANT

**Table Name:** RESTAURANT

**Attributes:**

- **restaurant\_id** (PK)
- name
- country
- city
- rating
- rating\_count
- cuisine
- link
- address

## Schema –

```
RESTAURANT(  
  restaurant_id INT      PRIMARY KEY,  
  name          VARCHAR(150),  
  country       VARCHAR(50),  
  city          VARCHAR(50),  
  rating        FLOAT,  
  rating_count  INT,  
  cuisine       VARCHAR(100),  
  link          VARCHAR(255),  
  address       VARCHAR(255)  
);
```

## 3. ORDERS

**Table Name:** ORDERS

**Attributes:**

- **order\_id** (PK)
- order\_date
- sales\_qty
- sales\_amount
- currency
- user\_id (FK → USERS.user\_id)
- restaurant\_id (FK → RESTAURANT.restaurant\_id)

## Schema –

```
ORDERS(  
  order_id  INT      PRIMARY KEY,  
  order_date DATE,  
  sales_qty  INT,  
  sales_amount DECIMAL(10, 2),  
  currency   VARCHAR(5),  
  user_id    INT      FOREIGN KEY REFERENCES USERS(user_id),  
  restaurant_id INT    FOREIGN KEY REFERENCES RESTAURANT(restaurant_id)  
);
```

#### 4. ORDER\_TYPE

**Table Name:** ORDER\_TYPE

**Attributes:**

- **order\_id** (PK & FK → ORDERS.order\_id)
- type (e.g., Dine-in, Delivery, Takeaway)

**Schema –**

```
ORDER_TYPE(  
  order_id INT      PRIMARY KEY REFERENCES ORDERS(order_id),  
  type   VARCHAR(20)  
);
```

#### 5. MENU

**Table Name:** MENU

**Attributes:**

- **menu\_id** (PK)
- r\_id (FK → RESTAURANT.restaurant\_id)
- item
- cuisine
- price
- f\_id (FK → FOOD.f\_id)

**Schema –**

```
MENU(  
  menu_id INT      PRIMARY KEY,  
  r_id   INT      FOREIGN KEY REFERENCES RESTAURANT(restaurant_id),  
  item   VARCHAR(100),  
  cuisine VARCHAR(50),  
  price  DECIMAL(10, 2),  
  f_id   INT      FOREIGN KEY REFERENCES FOOD(f_id)  
);
```

#### 6. FOOD

**Table Name:** FOOD

**Attributes:**

- **f\_id** (PK)
- item
- veg\_or\_non\_veg

**Schema-**

```
FOOD(  
  f_id   INT      PRIMARY KEY,  
  item   VARCHAR(100),  
  veg_or_non_veg VARCHAR(20)  
);
```

# V. Normalization

## 1. First Normal Form (1NF)

### Definition:

A table is in First Normal Form (1NF) if:

- Each column contains atomic values (no multiple values in a single column).
- Each record (row) is unique, and there are no duplicate rows.
- All entries in a column must be of the same type.

### Implementation in Our Database:

In our database, the menu and restaurants tables initially had a violation of 1NF, where the cuisine field stored multiple values (e.g., "Beverages, Pizzas") in a single cell. This violated the atomicity requirement of 1NF, as each column must contain a single, indivisible value.

### Resolution:

To resolve this, we introduced the menu\_cuisine table, which splits the multi-valued cuisine field into separate rows. This new structure eliminates the violation and ensures that each record now represents a single cuisine, maintaining 1NF compliance.

```
--Normalized menu
CREATE TABLE menu_cuisine (
    menu_id TEXT,
    cuisine TEXT
);
INSERT INTO menu_cuisine (menu_id, cuisine)
SELECT
    m.menu_id,
    TRIM(value) AS cuisine
FROM menu m,
     unnest(string_to_array(m.cuisine, ',')) AS value;

SELECT DISTINCT cuisine FROM menu_cuisine ORDER BY cuisine;
SELECT * FROM menu_cuisine
```

## 2. Second Normal Form (2NF)

### Definition:

A table is in Second Normal Form (2NF) if:

- It is in 1NF.
- There is no partial dependency, meaning that every non-key column is fully dependent on the entire primary key and not just a part of it.

### Implementation in Our Database:

In our menu table, we use a composite primary key consisting of menu\_id and f\_id, which together uniquely identify each record. The non-key attributes, such as price and cuisine, fully depend on both parts of the composite key, and there are no partial dependencies.

For example, if price depended solely on menu\_id or cuisine depended on f\_id independently, this would create a partial dependency and violate 2NF. However, both attributes depend on the full composite key, making the database compliant with 2NF.

### Conclusion for 2NF:

There are no partial dependencies in our database structure, and it is fully normalized to 2NF.

## 3. Third Normal Form (3NF)

### Definition:

A table is in Third Normal Form (3NF) if:

- It is in 2NF.

- There are no transitive dependencies, meaning that non-key attributes must not depend on other non-key attributes.

### **Implementation in Our Database:**

Upon analyzing the restaurant table, we found potential transitive dependencies:

1. rating\_count might depend on rating, and if rating\_count is derived from rating (i.e., the number of reviews corresponding to a rating), then this constitutes a transitive dependency.
2. Similarly, city might be functionally dependent on country if all restaurants within a specific city belong to the same country, creating another transitive dependency: city → country.

### **Resolution:**

To resolve these transitive dependencies and fully comply with 3NF, we propose the following changes:

1. Creating a separate table for ratings:  
We can store the rating and rating\_count in a separate ratings table, where each restaurant has a unique rating\_id that links to its rating and rating\_count. This separation will break the transitive dependency between rating\_count and rating, ensuring that rating\_count depends directly on rating\_id, not on rating.
2. Creating separate tables for cities and countries:  
We can create distinct tables for cities and countries to resolve the city → country transitive dependency. The restaurants table would then reference these new tables using foreign keys (city\_id, country\_id), eliminating redundancy and ensuring that city is no longer dependent on country.

# Preprocessing and inserting of data

## 1. Introduction to Data Preprocessing

Data preprocessing is a crucial step in preparing raw data for insertion into a database. The goal is to clean, format, and structure the data so that it is consistent and compatible with the destination database. This process involves several key steps, including handling missing or null values, escaping special characters, and ensuring that the data fits the schema of the target table.

In the case of this project, we are working with various data tables, such as the **restaurants table**, which require preprocessing before inserting into a relational database management system (RDBMS).

## 2. Preprocessing Data: General Workflow

The general workflow for preprocessing and inserting data follows these steps:

- **Step 1: Data Collection**

The first step involves collecting the raw data, which is typically stored in files such as Excel, CSV, or JSON. This data may come from various sources, such as web scraping, user inputs, or external databases.

- **Step 2: Data Cleaning**

The raw data may contain issues such as missing values (NaNs), inconsistent formats, or invalid entries (e.g., special characters). Data cleaning involves identifying and handling these issues to ensure the data is consistent and reliable.

- **Handling Missing Values:** Missing values are often represented as NULL in databases. These can be handled by replacing missing values with NULL or imputing them with default values or averages based on the dataset.
- **Escaping Special Characters:** Special characters like single quotes (') can break SQL queries. These characters are escaped by doubling them (e.g., turning ' into "), ensuring smooth insertion into the database.

- **Step 3: Data Transformation**

In this step, data might need to be transformed into a format that matches the target database schema. For instance:

- Converting text into proper case (e.g., capitalizing restaurant names).
- Formatting numerical values consistently (e.g., converting ratings to numeric types).
- Ensuring that the data fits the constraints of the database schema (e.g., ensuring IDs are unique).

## 3. Example: Restaurant Table

To provide a concrete example, let's consider the process of preprocessing and inserting data for the **restaurants table**.

- **Data Loading:**

The raw restaurant data is loaded into a Python environment using the **Pandas** library. We use the `read_excel()` function to read the data from an Excel file.

```
python
```

```
Copy
```

```
df = pd.read_excel("restaurant.xlsx")
```

- **Handling Missing Values:**

Missing values in the dataset are replaced with NULL for SQL compatibility. This step is important because missing values can cause SQL errors if not properly handled.

python

Copy

```
def escape_sql(value):  
    if pd.isna(value):  
        return "NULL"  
    value = str(value).replace("'", "'")  
    return f'{{ value }}'
```

- **Escaping Special Characters:**

In the restaurant data, names, addresses, and other text fields may contain single quotes ('), which need to be escaped to avoid breaking the SQL queries.

- **Generating SQL Queries:**

Once the data is cleaned, we generate SQL queries to create the table and insert the preprocessed data. For example:

- **Creating the Table:**

We define the SQL schema for the restaurants table with columns such as id, name, country, city, rating, rating\_count, cuisine, link, and address.

#### 4. Output: SQL File

The SQL queries generated (for both table creation and data insertion) are written to a .sql file. This file can then be executed in any RDBMS to create the table and insert the data. The file will contain:

The CREATE TABLE statement to define the schema.

Multiple INSERT INTO statements to insert the restaurant records.

For example, the generated SQL file might look like:

```
CREATE TABLE restaurants (  
    id INTEGER PRIMARY KEY,  
    name TEXT,  
    country TEXT,  
    city TEXT,  
    rating TEXT,  
    rating_count TEXT,  
    cuisine TEXT,  
    link TEXT,  
    address TEXT  
);
```

```
INSERT INTO restaurants VALUES (1, 'Restaurant A', 'India', 'New Delhi', '4.5', '150', 'Indian',  
'http://example.com', '123 Main Street');
```

```
INSERT INTO restaurants VALUES (2, 'Restaurant B', 'USA', 'New York', '4.7', '200', 'Italian',  
'http://example.com', '456 Broadway');
```



The preprocessing and insertion of data is an essential part of the data pipeline. For the **restaurants table**, the data was first cleaned by handling missing values and escaping special characters. Then, SQL queries were generated to create the database schema and insert the restaurant data. By automating this process using Python and SQL, data can be efficiently processed and inserted into a database for further analysis or application.

This process can be applied to other tables in the database, such as customer data, orders, or transactions, by adapting the preprocessing steps and SQL queries to fit the specific schema and data characteristics of each table.

## VI. SQL Queries

```
-- Query 1: Top 5 highest-rated restaurants in each city
-- Finds the top 5 restaurants per city based on ratings and rating counts.
WITH RankedRestaurants AS (
    SELECT city, r_id, name, rating,
           ROW_NUMBER() OVER (PARTITION BY city ORDER BY rating DESC, rating_count DESC)
    AS rank
    FROM restaurants WHERE rating IS NOT NULL
)
SELECT city, r_id, name, rating FROM RankedRestaurants WHERE rank <= 5;
```

Data Output Messages Notifications				
	city text	r_id [PK] integer	name text	rating text
1	Abids & Koti,Hyderabad	560855	Momo Street	5
2	Abids & Koti,Hyderabad	560860	Gobblers Rolls & Bowls	5
3	Abids & Koti,Hyderabad	394473	Swiss Castle	4.5
4	Abids & Koti,Hyderabad	22959	Hameedi Confectioners	4.5
5	Abids & Koti,Hyderabad	148766	Temptations	4.5
6	Abohar	530909	FOODY MOOD	4.7
7	Abohar	531342	Janta Sweet House	4.4
8	Abohar	156587	Bharawan Da Dhaba	4.4
9	Abohar	171675	Domino's Pizza	4.4
10	Abohar	420675	Royal Chicken	4.2

```
-- Query 2: Yearly sales per restaurant and identify highest sales year
-- Calculates yearly total sales per restaurant and identifies the highest sales year for each restaurant.
WITH SalesByYear AS (
    SELECT EXTRACT(YEAR FROM o.order_date) AS year, o.r_id, r.name AS restaurant_name,
    SUM(o.sales_amount) AS total_sales
```

```

FROM orders o JOIN restaurants r ON o.r_id = r.r_id
GROUP BY year, o.r_id, r.name
),
MaxSalesYear AS (
  SELECT r_id, MAX(total_sales) AS max_sales FROM SalesByYear GROUP BY r_id
)
SELECT s.year, s.restaurant_name, s.total_sales, ms.max_sales,
  CASE WHEN s.total_sales = ms.max_sales THEN 'Highest Sales' ELSE 'Not Highest' END AS
sales_comparison
FROM SalesByYear s JOIN MaxSalesYear ms ON s.r_id = ms.r_id
ORDER BY s.r_id, s.year;

```

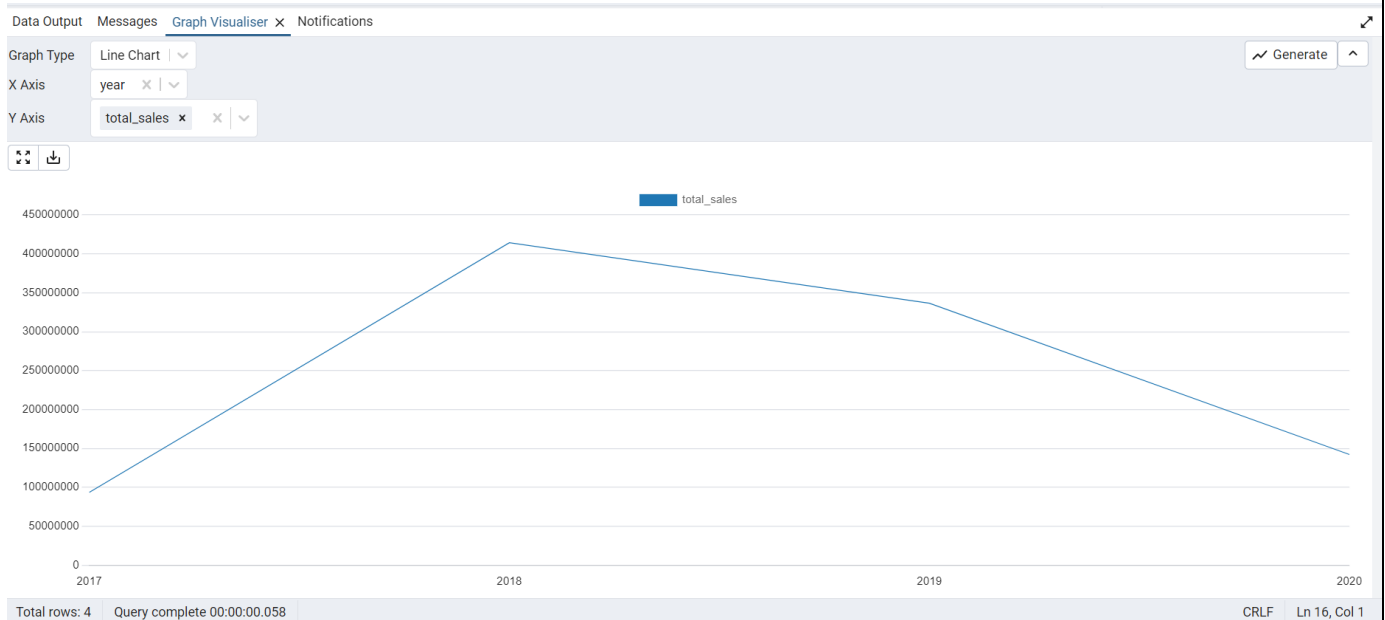
Data Output Messages Notifications					
	year numeric	restaurant_name text	total_sales bigint	max_sales bigint	sales_comparison text
1	2019	Tandoor Hut	3440	3440	Highest Sales
2	2019	Anand Sweets and Savouries	537	537	Highest Sales
3	2019	Anjappar	972	972	Highest Sales
4	2019	Tunday Kababi	3069	3069	Highest Sales
5	2019	Truffles	171	171	Highest Sales
6	2019	Delhi Food Point	34495	34495	Highest Sales
7	2019	Via Milano	764	764	Highest Sales
8	2019	Meghana Foods	588	588	Highest Sales
9	2019	The Hole in the Wall Cafe	222	222	Highest Sales
10	2019	Uday Sweets	44069	44069	Highest Sales

```

-- Total sales by year (Overall)
SELECT EXTRACT(YEAR FROM order_date) AS year, SUM(sales_amount) AS total_sales
FROM orders
GROUP BY year
ORDER BY year ASC;

```

Data Output Messages Notifications		
	year numeric	total_sales bigint
1	2017	93568402
2	2018	414308941
3	2019	336452114
4	2020	142235559



```
-- Query 3: Most ordered cuisines per country and city
-- Determines the popularity of cuisines by counting orders per country and city.
SELECT r.country, r.city, mc.cuisine, COUNT(*) AS total_orders
FROM orders o
JOIN restaurants r ON o.r_id = r.r_id
JOIN menu m ON o.r_id = m.r_id
JOIN menu_cuisine mc ON m.menu_id = mc.menu_id
GROUP BY r.country, r.city, mc.cuisine ORDER BY total_orders DESC;
```

Data Output Messages Notifications

	country text	city text	cuisine text	total_orders bigint
1	India	Adityapur	Chinese	35080
2	India	Adityapur	Indian	27587
3	India	Amritsar	North Indian	27464
4	India	Agra	North Indian	25477
5	India	Amritsar	Chinese	20411
6	India	HSR,Bangalore	North Indian	19028
7	India	Aurangabad	North Indian	18816
8	India	Agra	Chinese	18438
9	India	Allahabad	North Indian	18089
10	India	Koramangala,Bangalore	Chinese	17692

```
-- Query 4: Average order value per customer and identify high-value customers
-- Finds customers whose lifetime spending exceeds 5000 and calculates their average order value.
SELECT u.user_id, u.name, AVG(o.sales_amount) AS avg_order_value, SUM(o.sales_amount) AS
total_spent
FROM orders o JOIN users u ON o.user_id = u.user_id
GROUP BY u.user_id, u.name HAVING SUM(o.sales_amount) > 5000 ORDER BY total_spent DESC;
```

Data Output Messages Notifications				
<div> <div>☰</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>				
	user_id integer 🔒	name text 🔒	avg_order_value numeric 🔒	total_spent bigint 🔒
1	70591	Amanda Ballard	504878.000000000000	1514634
2	16975	Gina Carpenter	374156.250000000000	1496625
3	55915	Jonathan Vasquez	369746.750000000000	1478987
4	62645	Lisa Aguirre	369518.500000000000	1478074
5	94568	Richard Edwards	1338264.000000000000	1338264
6	1159	Elizabeth Martin	440217.666666666667	1320653
7	23667	Brian White	428137.333333333333	1284412
8	15402	Cassandra Benson	428031.000000000000	1284093
9	83766	Elizabeth Ryan	617720.000000000000	1235440
10	64054	Jeffrey Smith	1228148.000000000000	1228148

```
-- Query 5: Most frequently ordered menu items across restaurant categories
-- Lists menu items ranked by frequency of orders across all restaurant categories.
SELECT f.item, COUNT(*) AS times_ordered
FROM orders o JOIN menu m ON o.r_id = m.r_id JOIN food f ON m.f_id = f.f_id
GROUP BY f.item ORDER BY times_ordered DESC;
```

Data Output









Messages

Notifications

	<div>item</div> <div>text</div> <div></div>	<div>times_ordered</div> <div>bigint</div> <div></div>
1	Jeera Rice	6138
2	Veg Fried Rice	5896
3	Paneer Butter Masala	5886
4	French Fries	4206
5	Dal Fry	4128
6	Butter Naan	3942
7	Chicken Fried Rice	3698
8	Veg Biryani	3636
9	Cold Coffee	3526
10	Egg Fried Rice	2994

```
-- Query 6: Restaurants with low ratings but high sales
-- Identifies restaurants that have ratings below 3.0 but sales over 10,000 indicating potential quality issues.
SELECT r.r_id, r.name, r.rating, SUM(o.sales_amount) AS total_sales
FROM orders o JOIN restaurants r ON o.r_id = r.r_id WHERE r.rating ~ '^[0-9.]+'
```

GROUP BY r.r\_id, r.name, r.rating HAVING CAST(r.rating AS NUMERIC) < 3.0 AND  
SUM(o.sales\_amount) > 10000  
ORDER BY total\_sales DESC;

Data Output Messages Notifications				
				
			SQL	
	r_id [PK] integer	name text	rating text	total_sales bigint
1	516065	SRI LAKSHMI BAR AND RESTAURANT	2.8	1089685
2	416740	Burger It Up	2.9	671963
3	344732	Thakur Food Point & Restaurant & Tiffin Center	2.9	595556
4	432939	Mehfil Cafe/Food and Coffee	2.8	504037
5	356106	Burger Monster	2.1	392306
6	167569	Khurana Dhaba (Shaheed Bhagat Singh Cho...	2.8	357333
7	81319	Om Bikaner Sweets and Restaurant	2.2	350523
8	528832	DAKSH ROTI HUB	2.6	337819
9	307839	Simple Food	2.5	326940
10	346219	Pizza Hub	2.8	324569

-- Query 7: Sales trends by season, month, and week

-- Detects peak periods by aggregating total sales according to month, week, and season.

SELECT TO\_CHAR(order\_date, 'YYYY-MM') AS month, EXTRACT(WEEK FROM order\_date) AS  
week,  
CASE



```

WHEN EXTRACT(MONTH FROM order_date) IN (12,1,2) THEN 'Winter'
WHEN EXTRACT(MONTH FROM order_date) IN (3,4,5) THEN 'Spring'
WHEN EXTRACT(MONTH FROM order_date) IN (6,7,8) THEN 'Summer'
ELSE 'Autumn' END AS season,
SUM(sales_amount) AS total_sales FROM orders GROUP BY month, week, season ORDER BY month;

```

Data Output Messages Notifications				
<div> <div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div> </div>				
	month text 🔒	week numeric 🔒	season text 🔒	total_sales bigint 🔒
1	2017-10	43	Autumn	8670858
2	2017-10	44	Autumn	1720003
3	2017-10	41	Autumn	7701618
4	2017-10	40	Autumn	1215556
5	2017-10	42	Autumn	7041321
6	2017-11	45	Autumn	7410640
7	2017-11	48	Autumn	7399518
8	2017-11	47	Autumn	8640395
9	2017-11	44	Autumn	4066417
10	2017-11	46	Autumn	7868169

```

-- Query 8: Repeat vs. one-time customers
-- Classifies customers based on frequency of distinct order dates.
SELECT user_id, COUNT(DISTINCT order_date) AS order_count,

```

```

CASE WHEN COUNT(DISTINCT order_date) > 1 THEN 'Repeat' ELSE 'One-time' END AS
customer_type
FROM orders GROUP BY user_id;

```

Data Output Messages Notifications			
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>			
	user_id integer 🔒	order_count bigint 🔒	customer_type text 🔒
1	1	2	Repeat
2	2	3	Repeat
3	3	1	One-time
4	4	1	One-time
5	5	2	Repeat
6	6	1	One-time
7	8	1	One-time
8	9	3	Repeat
9	10	2	Repeat
10	13	3	Repeat

```

-- Query 9: Customers with highest lifetime spending
-- Finds the top 10 customers with the greatest total lifetime spending.
SELECT u.user_id, u.name, SUM(o.sales_amount) AS lifetime_spending
FROM orders o JOIN users u ON o.user_id = u.user_id GROUP BY u.user_id, u.name
ORDER BY lifetime_spending DESC LIMIT 10;

```

Data Output Messages Notifications			
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>			
	user_id integer 🔒	name text 🔒	lifetime_spending bigint 🔒
1	70591	Amanda Ballard	1514634
2	16975	Gina Carpenter	1496625
3	55915	Jonathan Vasquez	1478987
4	62645	Lisa Aguirre	1478074
5	94568	Richard Edwards	1338264
6	1159	Elizabeth Martin	1320653
7	23667	Brian White	1284412
8	15402	Cassandra Benson	1284093
9	83766	Elizabeth Ryan	1235440
10	64054	Jeffrey Smith	1228148

-- Query 10: Best-selling food items by occupation

-- Determines popular food items based on customers' occupations.

SELECT u.occupation, f.item, COUNT(\*) AS total\_orders

FROM orders o JOIN users u ON o.user\_id = u.user\_id JOIN menu m ON o.r\_id = m.r\_id JOIN food f ON m.f\_id = f.f\_id

GROUP BY u.occupation, f.item ORDER BY u.occupation, total\_orders DESC;

Data Output Messages Notifications			
<div> <div>≡+</div> <div></div> <div>▼</div> <div></div> <div>▼</div> <div></div> <div></div> <div></div> <div></div> <div>SQL</div> </div>			
	occupation text	item text	total_orders bigint
1	Employee	Jeera Rice	1890
2	Employee	Paneer Butter Masala	1828
3	Employee	Veg Fried Rice	1742
4	Employee	French Fries	1308
5	Employee	Dal Fry	1288
6	Employee	Butter Naan	1276
7	Employee	Cold Coffee	1142
8	Employee	Chicken Fried Rice	1130
9	Employee	Veg Biryani	1098
10	Employee	Dal Makhani	930

-- Query 11: Order quantity variation per restaurant over time

-- Measures correlation of sales quantity with time to understand quantity trends.

WITH qty\_trend AS (

SELECT o.r\_id, DATE\_TRUNC('month', o.order\_date) AS month, SUM(o.sales\_qty) AS total\_quantity  
FROM orders o WHERE o.r\_id IS NOT NULL GROUP BY o.r\_id, month

)

SELECT r.r\_id, r.name, CORR(EXTRACT(EPOCH FROM qt.month), qt.total\_quantity) AS  
qty\_trend\_score

FROM qty\_trend qt JOIN restaurants r ON qt.r\_id = r.r\_id

GROUP BY r.r\_id, r.name HAVING qty\_trend\_score IS NOT NULL;

Data Output Messages Notifications			
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>			
	r_id [PK] integer	name text	qty_trend_score double precision
1	494001	Kolkata Biryani - Jugsalai	1
2	477628	Grameen Kulfi	1
3	103649	Blooms	-1
4	293915	Subway	-1
5	557008	Kanya Restaurant	1
6	439417	DESI BIDESI	-1
7	523044	Olive Multicuisine Restaurant	-1
8	166613	Devi Family Restaurant	-1
9	97199	Domino's Pizza	1
10	304302	Cafe Coffee Day	-1

-- Query 12: Restaurants with increasing/decreasing sales trends

-- Analyzes correlation between monthly sales and time to identify trends.

WITH sales\_monthly AS (

SELECT o.r\_id, DATE\_TRUNC('month', o.order\_date) AS month, SUM(o.sales\_amount) AS  
monthly\_sales

FROM orders o WHERE o.r\_id IS NOT NULL GROUP BY o.r\_id, month

)

SELECT r.r\_id, r.name, CORR(EXTRACT(EPOCH FROM sm.month), sm.monthly\_sales) AS  
sales\_trend

FROM sales\_monthly sm JOIN restaurants r ON sm.r\_id = r.r\_id GROUP BY r.r\_id, r.name HAVING  
sales\_trend IS NOT NULL;

Data Output Messages Notifications

	r_id [PK] integer	name text	sales_trend double precision
1	165533	IFC Restaurant	-1
2	533186	Aditya Inn Restaurant	-1
3	483552	AUROUS RESTRO LOUNGE	-1
4	109839	Manohar Maharaj & Sons	1
5	487422	Foodie Baba	-1
6	494001	Kolkata Biryani - Jugsalai	1
7	578207	SAMBHU DOSA	-1
8	109258	What's up Food	-1
9	477628	Grameen Kulfi	1
10	387674	Biggies Burger	-1

-- Query 13: Restaurants with stable order quantity

-- Finds restaurants with low variation (standard deviation) in monthly order quantity.

WITH monthly\_qty AS (

SELECT o.r\_id, DATE\_TRUNC('month', o.order\_date) AS month, SUM(o.sales\_qty) AS total\_quantity  
FROM orders o GROUP BY o.r\_id, month

), stddevs AS (

SELECT r\_id, STDDEV\_POP(total\_quantity) AS qty\_stddev FROM monthly\_qty GROUP BY r\_id  
)

SELECT r.r\_id, r.name, s.qty\_stddev FROM stddevs s JOIN restaurants r ON s.r\_id = r.r\_id  
WHERE s.qty\_stddev IS NOT NULL ORDER BY s.qty\_stddev ASC;

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑

🗄

⬇

📈

SQL

	r_id [PK] integer	name text	qty_stddev numeric
1	468609	Xero degrees	0
2	10896	Taco Bell	0
3	556211	9 Tea ml Coffee Bar	0
4	427159	THE BETEL LEAF CO - Originally from Bangalore	0
5	44127	Foods 49	0
6	368584	Momogato	0
7	94220	Kaveri Sweets	0
8	511713	The Pizza Diet	0
9	513030	Fruitfull	0
10	449783	MINGS DYNASTY	0

-- Query 14: High ratings but low sales restaurants (potentially under-marketed)

-- Spots high-rated restaurants with disproportionately low sales.

SELECT r.r\_id, r.name, r.rating, SUM(o.sales\_amount) AS total\_sales

FROM orders o JOIN restaurants r ON o.r\_id = r.r\_id WHERE CAST(r.rating AS NUMERIC) >= 4.5  
GROUP BY r.r\_id, r.name, r.rating HAVING SUM(o.sales\_amount) < 5000;

Data Output

Messages

Notifications

SQL

	r_id [PK] integer	name text	rating text	total_sales bigint
1	427159	THE BETEL LEAF CO - Originally from Bangalore	4.9	3306
2	42756	Baskin Robbins	4.6	2750
3	513719	Biryani Hazir Ho	4.9	444
4	300427	TPP - The Pizza Project	4.6	310
5	433252	OM SAI FRANKIES	4.6	199
6	79446	Pavanputra Ice Cream	4.5	4491
7	533145	hot and fresh canadian pizza	4.9	3852
8	159377	Lutyens kitchen	4.5	88
9	512831	Biryani Hazir Ho	4.5	162
10	8057	Golden phoenix	4.5	2356

-- Query 15: Most popular cuisines across restaurant rating tiers  
 -- Identifies the most ordered cuisines categorized by restaurant rating (High, Medium, Low, Unrated).

```
SELECT
CASE
  WHEN CAST(r.rating AS NUMERIC) >= 4.5 THEN 'High Rated'
  WHEN CAST(r.rating AS NUMERIC) >= 3.5 THEN 'Medium Rated'
  WHEN CAST(r.rating AS NUMERIC) >= 0 THEN 'Low Rated'
  ELSE 'Unrated'
END AS rating_tier,
mc.cuisine,
COUNT(*) AS total_orders
FROM orders o
JOIN menu m ON o.r_id = m.r_id
JOIN menu_cuisine mc ON m.menu_id = mc.menu_id
JOIN restaurants r ON o.r_id = r.r_id
WHERE r.rating ~ '[0-9.]+'
GROUP BY rating_tier, mc.cuisine
ORDER BY rating_tier, total_orders DESC;
```








Data Output Messages Notifications

	rating_tier text	cuisine text	total_orders bigint
1	High Rated	Desserts	6959
2	High Rated	Ice Cream	5307
3	High Rated	Chinese	4703
4	High Rated	Beverages	4230
5	High Rated	North Indian	3724
6	High Rated	Indian	3555
7	High Rated	Snacks	2319
8	High Rated	Pizzas	2223
9	High Rated	Bakery	1655
10	High Rated	Sweets	1534



-- Query 16: Food preference (veg vs. non-veg) by gender  
-- Analyzes how food preference (vegetarian vs non-vegetarian) varies across different genders.

```
SELECT
  u.gender,
  f.veg_or_non_veg,
  COUNT(*) AS total_orders
FROM orders o
JOIN users u ON o.user_id = u.user_id
JOIN menu m ON o.r_id = m.r_id
JOIN food f ON m.f_id = f.f_id
GROUP BY u.gender, f.veg_or_non_veg;
```

Data Output Messages Notifications			
			
			SQL
	gender text	veg_or_non_veg text	total_orders bigint
1	Female	Non-veg	137181
2	Female	Veg	312573
3	Male	Non-veg	184715
4	Male	Veg	428294

-- Query 17: Customers who place consistently low-spend orders  
-- Identifies customers with the lowest average order value consistently.

```
SELECT
  u.user_id,
  u.name,
  AVG(o.sales_amount) AS avg_order_value
FROM users u
JOIN orders o ON u.user_id = o.user_id
GROUP BY u.user_id, u.name
ORDER BY avg_order_value ASC;
```

Data Output Messages Notifications			
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>			
	user_id integer 🔒	name text 🔒	avg_order_value numeric 🔒
1	97913	Michael Wang	0.00000000000000000000
2	92860	Maurice Hansen	0.00000000000000000000
3	11728	Nicole Hill	0.00000000000000000000
4	33667	Kelly Davis	0.00000000000000000000
5	36764	Ashley Hawkins	0.00000000000000000000
6	86313	Paul Thompson	0.00000000000000000000
7	47100	Matthew Park	0.00000000000000000000
8	59082	William Jones	0.00000000000000000000
9	74412	Kelly Young	0.00000000000000000000
10	69917	Christina Martinez	0.00000000000000000000

-- Query 18: Average sales and order count by restaurant rating tier  
 -- Calculates total orders and revenue, grouped by restaurant rating tiers.

```

SELECT
CASE
  WHEN CAST(r.rating AS NUMERIC) >= 4.5 THEN 'High Rated'
  WHEN CAST(r.rating AS NUMERIC) >= 3.5 THEN 'Medium Rated'
  WHEN CAST(r.rating AS NUMERIC) >= 0 THEN 'Low Rated'
  ELSE 'Unrated'
END AS rating_tier,
COUNT(*) AS total_orders,
SUM(o.sales_amount) AS total_revenue
FROM orders o
JOIN restaurants r ON o.r_id = r.r_id
WHERE r.rating ~ '^[0-9.]+$'
GROUP BY rating_tier
ORDER BY total_revenue DESC;


```

Data Output Messages Notifications			
<div> <div>≡+</div> <div>📄</div> <div>▼</div> <div>📋</div> <div>▼</div> <div>🗑️</div> <div>🗄️</div> <div>⬇️</div> <div>📈</div> <div>SQL</div> </div>			
	rating_tier text	total_orders bigint	total_revenue bigint
1	Medium Rated	48187	331822609
2	Low Rated	8856	67085352
3	High Rated	4483	26951049

```
-- Query 19: Restaurants with highest sales variation by day of the week
-- Finds restaurants that have large fluctuations in sales throughout the week.
WITH daily_sales AS (
  SELECT
    r.r_id,
    EXTRACT(DOW FROM o.order_date) AS day_of_week,
    SUM(o.sales_amount) AS daily_sales
  FROM orders o
  JOIN restaurants r ON o.r_id = r.r_id
  GROUP BY r.r_id, day_of_week
)
SELECT
  r.r_id,
  r.name,
  MAX(daily_sales) - MIN(daily_sales) AS sales_variation
FROM daily_sales d
JOIN restaurants r ON d.r_id = r.r_id
GROUP BY r.r_id, r.name
ORDER BY sales_variation DESC;
```







SQL

	r_id [PK] integer 	name text 	sales_variation bigint 
1	397980	Chaw Box	65296
2	183009	Baker Street	42107
3	510109	The Bawarchi	40180
4	500628	Instaa Fast Food Center	35681
5	101351	Pizza Hut	26075
6	516171	FRUITASTY	19616
7	129550	Asian Inn	17546
8	413451	Chulha Chauka	14338
9	263185	The Chocolate Room	13004
10	549242	RAMA JUICE AND SHAKES PARLOUR (LASSI N SHA...	12579

-- Query 20: Menu items with increasing sales momentum  
 -- Identifies food items with sales growth comparing the recent 90-day period to previous periods.

```
WITH recent_sales AS (
  SELECT
    m.f_id,
    SUM(o.sales_qty) AS qty_last_60
  FROM orders o
  JOIN menu m ON o.r_id = m.r_id
  WHERE o.order_date >= CURRENT_DATE - INTERVAL '90 days'
  GROUP BY m.f_id
),
previous_sales AS (
  SELECT
    m.f_id,
    SUM(o.sales_qty) AS qty_prev
  FROM orders o
  JOIN menu m ON o.r_id = m.r_id
  WHERE o.order_date < CURRENT_DATE - INTERVAL '90 days'
  GROUP BY m.f_id
)
SELECT
  f.item,
  COALESCE(r.qty_last_60, 0) AS qty_last_60,
  COALESCE(p.qty_prev, 0) AS qty_prev
FROM food f
LEFT JOIN recent_sales r ON f.f_id = r.f_id
LEFT JOIN previous_sales p ON f.f_id = p.f_id
ORDER BY qty_last_60 DESC;
```

Data Output Messages Notifications			
       SQL			
	item text	qty_last_60 bigint	qty_prev bigint
1	Veg Creamy Burger	0	291
2	White Sauce	0	296
3	Pink Sauce	0	178
4	Chicken Pasta	0	502
5	Noodles	0	764
6	Singapuri Noodles	0	2271
7	Dry Manchurian	0	1630
8	Gravy Manchurian	0	1279
9	Egg Sandwich	0	2321
10	Red Chilli Papper	0	100

## **VII. Project demonstration**

### **Tools/Software/Libraries Used:**

For the development and demonstration of this project, the following tools, software, and libraries were utilized:

#### **1. Database Management System (DBMS):**

- **PostgreSQL (PgAdmin4):** This was used for managing the relational database. PostgreSQL helped in creating the tables, running complex SQL queries, and normalizing the data which performed better than any other software.

#### **2. Programming Languages:**

- **SQL:** The primary language used for querying and manipulating data in the PostgreSQL database, including executing complex queries for analysis.
- **Python:** Python was used for data processing and cleaning to import onto PostgreServer.

#### **3. Database Design and ERD Tool:**

- **Draw.io:** This tool was used for designing and visualizing the database schema (Entity-Relationship Diagram). It helped in the creation and documentation of the relational database structure.

#### **4. GUI :**

- **Power BI:** Used as the GUI for the project. Power BI allows users to interact with the database, run queries, and visualize the results dynamically through interactive dashboards and reports. It enables data visualization and analysis without requiring complex technical knowledge, offering a user-friendly interface for displaying trends, sales data, and other metrics.

#### **5. Database Management Tools:**

- **pgAdmin:** A web-based administration tool for managing PostgreSQL databases. It provides a user-friendly interface to query, manage, and visualize the data in the database.

#### **6. Libraries for Data Analysis and Visualization (If applicable):**

- **Pandas:** A Python library used for data manipulation and analysis. It was used in cases where complex data processing was required, especially for transforming raw data into a structured format.

### **Screenshot and Description of the Demonstration of Project:**

#### **1. Dashboard Overview:**

The first page of the Zomato dashboard presents an engaging and user-friendly interface, with a dynamic and vibrant theme focused on the food and restaurant industry. The page showcases a visually appealing background featuring a bowl of noodles and various fresh ingredients, reinforcing the platform's culinary nature. The "Zomato" logo is prominently displayed, immediately highlighting the purpose of the platform. A clear "Dashboard" button invites users to access detailed analytics and insights, making it easy to navigate and interact with the data. The design effectively blends aesthetic appeal with functionality,

offering users a seamless and enjoyable experience as they explore key metrics and restaurant performance data.



The second page provides a comprehensive and dynamic view of the company's overall performance, focusing on key metrics such as the total sales amount, user performance, and sales trends across various cities.

**1. General Overview:**

- The total sales amount for Zomato is ₹987M, with a total of 2M orders processed, contributing to 148K ratings.
- The dashboard presents the sales in various categories, including Non-Veg, Veg, and Other cuisines, along with the respective sales amounts and ratings for each category.
- The Top 10 Cities by sales are prominently displayed, providing a breakdown of the highest-performing locations. Notable cities like Tirupati, Electronic City, and Baner rank at the top, reflecting the diversity of Zomato's reach.

**2. User Performance:**

- The platform provides insights into gain and lost users over a specified period, highlighting 12K new users and 33K lost users, with a breakdown by gender. These figures are essential for understanding Zomato's user retention and acquisition strategies.
- The users by age graph gives a demographic breakdown, helping to identify the most active user segments.

**3. Yearly Sales Trend:**

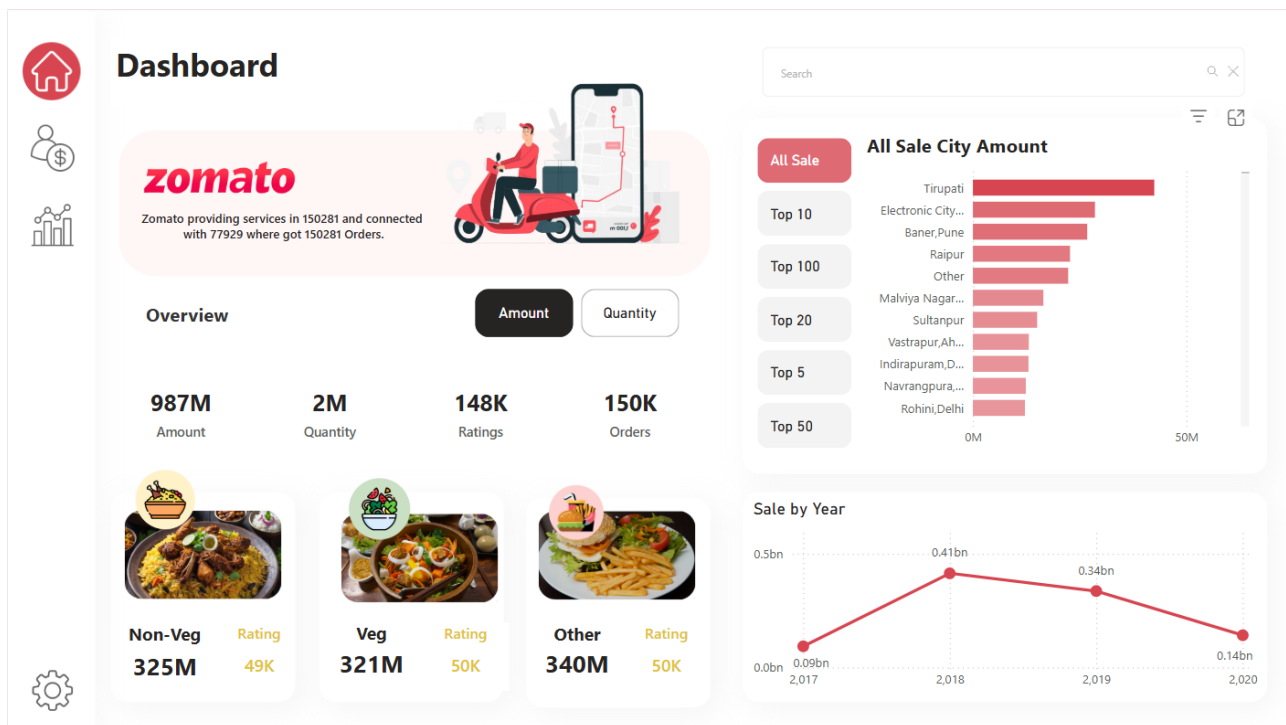
- A graphical representation of sales by year shows the growth trajectory, with a noticeable peak in 2018 and a decline in 2020. This visualization helps assess the impact of various factors on sales performance.

#### 4. All Sale City Amount:

- The dashboard displays a bar chart showing sales amounts across multiple cities, with Tirupati leading by a significant margin. Other cities such as Electronic City, Baner, and Raipur are also among the highest-performing regions, illustrating regional variations in sales patterns.

#### 5. Toggle Option Between Amount and Quantity:

- The dashboard includes a convenient toggle option that allows users to switch between viewing Sales Amount and Quantity of Orders. This feature provides users with a flexible view of the data, allowing them to compare both the financial value and volume of sales across different regions and categories.



## 2. Here are the key sections in the dashboard:

### 1. Zomato Overview:

- Active Users: The number of users who are actively using the platform.
- User Count: The total count of users on the platform.
- Ratings: Total number of ratings provided by users.
- Orders: The total number of orders made by users.

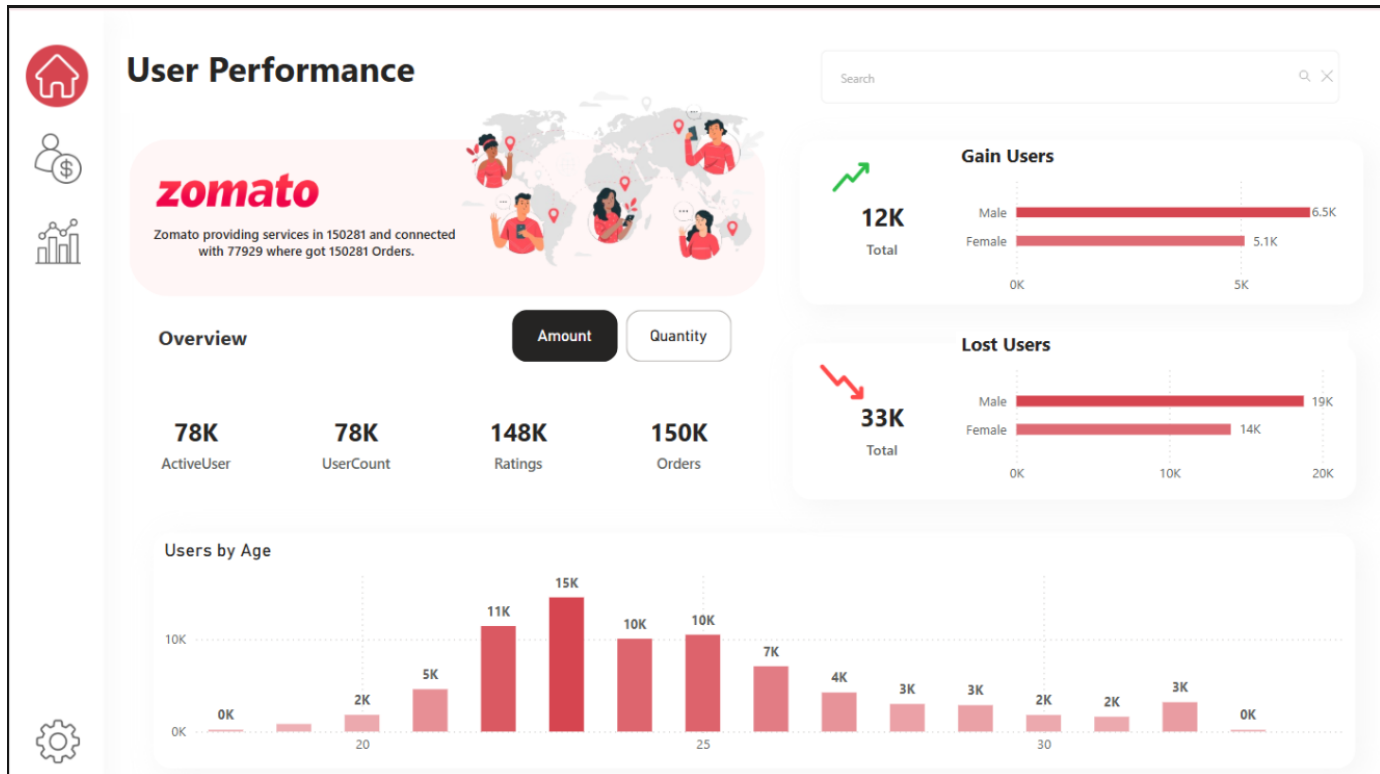
### 2. User Gain and Loss:

- Gain Users: The number of new users who joined in the recent period, separated by gender.
- Lost Users: The number of users who left the platform, again separated by gender.

### 3. User Age Distribution:

- The graph shows the breakdown of users based on their age group, highlighting how many users belong to each age group.





### 3. City Performance Overview

The City Performance section of the Zomato Dashboard provides a detailed breakdown of the company's performance across various cities. The key metrics highlighted in this section include total sales amount, order quantity, ratings, and user activity for each city, offering a comprehensive view of Zomato's market reach and success in different regions.

#### 1. General Overview:

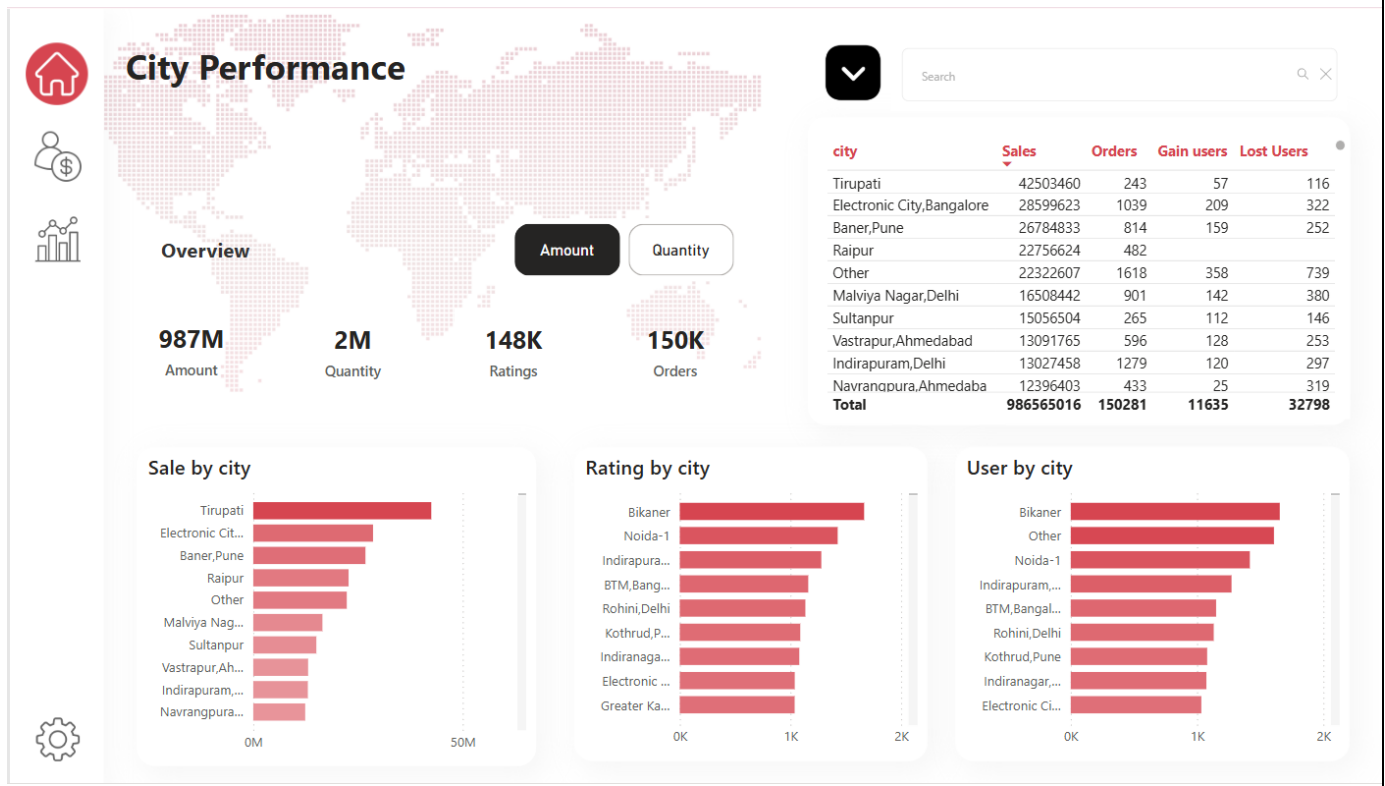
- Total Sales Amount: ₹987M
- Total Orders: 2M
- Total Ratings: 148K
- The dashboard allows switching between viewing sales data by Amount and Quantity, offering a flexible way to analyze the performance of cities based on financial or volume metrics.

#### 2. Sales by City:

- The Top Performing Cities in terms of sales are displayed with Tirupati at the top, followed by Electronic City, Baner, and Raipur. These cities show the highest total sales, which reflects the demand and consumer engagement in these regions.
- The chart helps in identifying cities with the highest and lowest sales performance, facilitating strategic decisions such as targeted marketing or regional expansion.

#### 3. Rating by City:

- A separate Rating by City chart displays cities with the highest ratings, with Bikaner leading the pack, followed by Noida-1, Indirapuram, and others. High ratings are often indicative of customer satisfaction and service quality in these regions.
  - This metric helps in evaluating not just sales performance, but also customer sentiment and feedback across different locations.
4. User Activity by City:
- The User by City chart showcases the distribution of users across cities, with Bikaner, Other, and Noida-1 having the largest user bases.
  - This section provides insights into user growth, engagement, and retention across cities, which is crucial for understanding market penetration and identifying regions with growth potential.
5. Additional Insights:
- The data also includes insights into Gain Users and Lost Users, helping track user retention and acquisition trends in different cities. For instance, Tirupati shows significant gain users of 57, while Sultanpur has lost users of 146. These insights are vital for analyzing the effectiveness of retention strategies and the need for targeted interventions.
6. City Data and User Behavior:
- The dashboard also tracks user behavior metrics, providing an overview of user engagement and activity levels across cities. This data assists in understanding which regions are driving the most activity and where additional focus is required to boost user engagement.



## **VIII. Self -Learning beyond classroom**

Through this project there was a lot of self-learning that we did that was well beyond what was taught in class. Performing our job, we had the chance to go through a wide set of tools, concepts, and technologies mostly used in the data analytics industry, improving dramatically our problem-solving and technical implementation skills.

### **1. PostgreSQL: Learning and Working**

PostgreSQL was the database system we chose for this project mainly because it's open-source and has good SQL support. Since we hadn't worked with it before, we had to start from scratch—setting it up on both Windows and Linux, creating databases and tables, and learning how to give roles and permissions. We used pgAdmin for managing things visually, and also got comfortable running queries through the terminal. Along the way, we explored some of PostgreSQL's useful features like `EXTRACT()` and `TO_CHAR()` for handling dates, and even used `CORR()` for basic correlation analysis. Learning how to use PostgreSQL properly helped us build our system efficiently and gave us hands-on experience we hadn't had in class.

### **2. Advanced SQL Concepts**

During the project, we had to go beyond simple `SELECT` statements and learned to write more complex queries. We used Common Table Expressions (CTEs) and window functions like `ROW_NUMBER()` and `RANK()` to get more detailed insights from the data. These came in handy when we had to rank restaurants or analyze trends. We also used functions like `CASE WHEN`, `DATE_TRUNC()`, and `GROUP BY` to filter and organize our results better. These concepts weren't fully covered in class, so figuring them out while working on the project really helped us improve our SQL skills in a more practical way.

### **3. Power BI Data Modeling and Visualization**

In fact, visualization tools were not even on the syllabus but we learned Power BI on our own to come up with easy-to-understand dashboards/reports. PostgreSQL Connecting to Power BI and ETL using Power Query For measuring KPI we wrote custom DAX formula and started using slicers, charts and map visualizations. This enabled us to deliver real-time visuals reflecting customer insights, regional cuisines, revenue streams, and more. Power BI — Learning Power BI provided us with a good sense of what it takes to convert data to visual artifacts that help with decision-making in businesses.

### **4. Normalising Data and Optimising Schema**

To apply normalization to a real-world schema like Zomato's, we had to go back and strengthen our understanding of theoretical concepts like functional dependencies, partial dependencies, and transitive dependencies. We focused on taking our schema step-by-step through First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF). This process helped us break down complex relationships into clean, well-structured tables while making sure that important data wasn't lost or duplicated. At the same time, we were able to maintain consistency and improve scalability, which is essential when dealing with large datasets like those in food delivery platforms. Going through normalization in a practical context really helped us develop a methodical approach to designing efficient

and reliable databases—something that’s key for anyone looking to work seriously in backend or data architecture.

### **5. Time Series Analysis and Season Variations**

We learned how to analyze the temporal nature of sales, leveraging SQL functions to extract the year, month, week, and even seasons from order dates. To better understand cycles of business and their impact on sales, we categorized data into quarters, mapping performance to seasons: Winter, Spring, Summer and Autumn. We also looked at how correlations over time could indicate whether a restaurant was improving, declining or stable. Such insights are critical for industries like food delivery, where customer demand fluctuates greatly with time.

### **6. Dealing with Problems with Real Data**

Real datasets would how to face real concerns such as missing values, empty entries, different formats, and damaged data. We learned how to clean and filter such data using SQL—applying such methods as regular expressions (~ operator), IS NOT NULL checks, and type casting. These problems made us realize the importance of preparing data prior to any analysis and helped us get a glimpse of the real-world scenarios of data engineers.

### **7. Domain Knowledge in Food Tech**

The food delivery and restaurant industry is larger than we imagined, so we researched initial steps to build meaningful insights. We examined how customers use food platforms, how culinary preferences vary across demographics, and how location affects restaurant sales. This domain specific learning helped us build a more relevant database schema and interpret analytics from a business perspective. We also explored how Zomato uses Data Science for marketing, doing personalization, revenue optimization.

### **8. Using Python for Data Handling and Generating Insert Queries**

One key aspect of the project was automating the process of inserting data into PostgreSQL. To handle large datasets efficiently, we used **Python** with **Pandas** to read data from Excel files and generate SQL insert queries. This involved using Python to load the data, clean it, and prepare it for insertion into the database. For example, we wrote a script that would load the data from an Excel file using `pandas.read_excel()`. Handle special characters like single quotes that could break SQL queries, ensuring data was properly escaped. Dynamically generate INSERT statements for each row in the dataset.

:

## **IX. Learning from the Project**

This project gave us a real-world understanding of how database systems work, beyond what we learned in class. Working on an application like Zomato analytics allowed us to see how a DBMS functions in practice. We didn't just learn what a database is, but also why it is so important for modern applications.

One of the biggest takeaways was our hands-on experience with PostgreSQL. We had previously learned SQL in a basic way, but this project allowed us to explore its full potential. We learned how to write complex queries, use advanced features like CTEs, window functions, and date functions, and even run correlation analysis. It was surprising to see how SQL can do much more than just fetch data – it can reveal hidden trends and business insights that would have been missed otherwise.

Another important part of the project was learning Power BI for data visualization. We had no previous experience with data visualization software, but Power BI introduced us to a whole new area. We learned how to clean and organize data, then create interactive dashboards that are both professional and insightful. This gave us a better understanding of how decision-makers use dashboards to quickly assess what's happening and identify key issues.

Overall, this project made us realize how crucial databases are in today's world. We saw how platforms like Zomato rely on databases to store large amounts of data about customers, restaurants, and orders in an organized way, allowing for quick and meaningful analysis. We understood how companies use databases to make data-driven decisions, offer personalized experiences, and stay competitive in the industry.

This project also helped us see how SQL, DBMS, and visualization tools work together to build smart systems. Most importantly, it showed us that we can use these tools to solve real-world problems and create something valuable.

## **X. Challenges Faced**

Throughout the development of this project, several challenges were encountered that required careful handling and strategic decisions to overcome. These challenges spanned across database management, data cleaning, query execution, and creating an interactive GUI for analysis. Below are some of the key obstacles faced during the project:

1. Issues with MySQL Workbench for Large Datasets:
  - Initially, MySQL Workbench was used to import and manage the data. However, as the dataset grew in size, the tool began to exhibit performance issues, particularly when handling large datasets. MySQL Workbench was unable to effectively import and process the data, causing delays and disruptions in the project's progress.
  - To resolve this issue, pgAdmin4, a more robust and scalable PostgreSQL management tool, was adopted. PostgreSQL was better suited for handling large volumes of data efficiently, making it a more reliable solution for managing our database.
2. Data Import and Dataset Quality Issues:
  - During the data import process, it became apparent that the dataset contained numerous errors and inconsistencies. One of the significant issues encountered was the presence of multivalues in fields that were supposed to be atomic, such as the cuisine column in the restaurant data. The cuisine column contained multiple values (e.g., "Beverages, Pizzas") in a single cell, violating the First Normal Form (1NF).
  - To tackle this, Python was employed for data cleaning. Libraries such as Pandas were used to preprocess and split the multivalued entries into separate rows, ensuring the data complied with normalization standards. This step also involved handling null values and standardizing data formats, which helped improve data consistency.
3. Errors Encountered While Running Queries:
  - While writing and executing SQL queries, multiple errors were faced. Some queries failed due to incorrect joins, data type mismatches, or the absence of required indexes, leading to slower query performance. Optimization of queries was crucial to improve execution times and ensure that the queries worked as expected.
  - Troubleshooting and debugging queries often involved revisiting the database schema and identifying areas where relationships or data types were not properly defined. Adjustments were made to ensure that the queries provided accurate results, especially when aggregating large amounts of data.
4. Challenges with Loading Large Datasets into Power BI:
  - A significant challenge arose when trying to load the huge datasets into Power BI for visualization. The size of the data exceeded the recommended limits for Power BI's standard import capabilities, resulting in performance issues, slow loading times, and occasionally system crashes.
  - To address this, strategies such as data aggregation and pre-filtering were employed to reduce the dataset size before importing it into Power BI. Additionally, direct query mode was used in some cases, where Power BI could query the database directly rather than importing large datasets into memory.
5. Challenges in Building the GUI on Power BI:

- Developing a graphical user interface (GUI) on Power BI proved to be a challenging task due to the complexity of the data and the need for an intuitive and interactive layout. The interactivity required in the GUI, such as filtering, dynamic charts, and drill-down options, required careful design and configuration.
  - The challenge was in ensuring that the visualizations were responsive and user-friendly while providing actionable insights. A significant amount of time was spent on designing the dashboard layout, ensuring that all the key metrics, such as total sales, user activity, and city performance, were displayed clearly. Additionally, implementing the toggle option between viewing the data by amount or quantity further complicated the design.
6. Additional Challenges:
- Throughout the project, data inconsistencies were encountered that affected the overall quality of the analysis. There were mismatched values across different tables, missing records, and inconsistencies in naming conventions.
  - Ensuring data integrity and quality required continuous monitoring and manual intervention. These issues were resolved by writing custom scripts to clean the data, ensuring it adhered to the necessary standards before proceeding with the analysis.

# **XI. Conclusion**

This project on **Zomato Data Analytics** has provided us with a practical understanding of how database systems are designed and applied in real-world scenarios. It allowed us to implement concepts from database theory into a functional application involving data modelling, query design, and visualization.

We began by designing a relational database that accurately reflects the structure and operations of a food delivery platform. Through normalization, we ensured the schema remained logically sound, minimized redundancy, and supported efficient data handling. This foundational work helped us write and execute queries that delivered precise and scalable results.

Using **PostgreSQL**, we moved beyond simple queries and explored advanced features such as Common Table Expressions (CTEs), window functions, and correlation analysis. These tools allowed us to extract insights related to customer behaviour, sales trends, seasonal variation, and restaurant performance.

Working with these techniques helped improve our understanding of how databases can be used not just for storage, but also for business analysis.

The use of **Power BI** enabled us to present the results of our analysis in a clear and accessible format. We created visualizations that summarized key metrics, enabling better interpretation of trends and operational outcomes. This component of the project demonstrated how data visualization complements backend data processing by making the insights more practical and actionable.

We also encountered and resolved several data quality issues, including missing or inconsistent entries. Addressing these challenges gave us experience in preparing data for analysis—an essential step in any data-driven project. Additionally, we developed a better understanding of how domain knowledge impacts data interpretation, especially in an industry like food delivery where user preferences and regional factors play a key role.

Overall, the project has strengthened our understanding of database systems and their integration with analytics tools. It reinforced the importance of clean design, accurate querying, and effective data presentation, all of which are essential for building reliable data solutions in real-world applications.