## **PostgreSQL Schema Documentation for Food Delivery Service**

GROUP: David Xiao, Shadman Rakib, Raymond Li

-- Customer

CREATE TABLE Customer (

Customer\_ID SERIAL PRIMARY KEY,

First\_name VARCHAR(50) NOT NULL,

Last\_name VARCHAR(50) NOT NULL,

Phone\_number VARCHAR(15),

Email VARCHAR(100)

);

* CREATE TABLE Customer (...): Starts the definition of a new table named Customer.
* Customer\_ID SERIAL PRIMARY KEY: Creates a unique, auto-incrementing identifier. SERIAL is a PostgreSQL shorthand for creating an integer column with an associated sequence. Used here to uniquely identify each customer.
* First\_name VARCHAR(50) NOT NULL: Stores the customer's first name. VARCHAR(50) means it can store up to 50 characters. NOT NULL enforces that this field is required.
* Last\_name VARCHAR(50) NOT NULL: Same as above for the last name.
* Phone\_number VARCHAR(15): Optional. Accommodates various phone number formats.
* Email VARCHAR(100): Optional. Supports long email addresses.

### **Customer\_Addresses Table**

-- Customer Addresses

CREATE TABLE Customer\_Addresses (

Address\_ID SERIAL PRIMARY KEY,

Postal\_code VARCHAR(10) NOT NULL,

Street\_address VARCHAR(100) NOT NULL,

City VARCHAR(50) NOT NULL,

State VARCHAR(50) NOT NULL,

Customer\_ID INTEGER NOT NULL,

FOREIGN KEY (Customer\_ID) REFERENCES Customer(Customer\_ID) ON DELETE CASCADE

);

* Stores multiple addresses per customer (1:M relationship).
* Address\_ID SERIAL PRIMARY KEY: Unique, auto-incremented ID for each address.
* The address fields use reasonable VARCHAR lengths to balance validation and flexibility.
* Customer\_ID INTEGER NOT NULL: Links this address to a specific customer.
* FOREIGN KEY (...) ON DELETE CASCADE: If a customer is deleted, their addresses are too.

### **Restaurant Table**

-- Restaurant

CREATE TABLE Restaurant (

Restaurant\_ID SERIAL PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Postal\_code VARCHAR(10) NOT NULL,

Street\_address VARCHAR(100) NOT NULL,

City VARCHAR(50) NOT NULL,

State VARCHAR(50) NOT NULL,

Phone\_number VARCHAR(15),

Email VARCHAR(100)

);

* Represents restaurant locations.
* Fields mirror customer and address fields for consistency.
* Phone\_number and Email are optional to allow for incomplete records.

### **Menu\_Items Table**

-- Menu Items

CREATE TABLE Menu\_Items (

MenuItem\_ID SERIAL PRIMARY KEY,

Description TEXT,

Price NUMERIC(10, 2) NOT NULL,

Availability BOOLEAN NOT NULL DEFAULT TRUE

);

* MenuItem\_ID SERIAL PRIMARY KEY: Unique ID for each dish.
* Description TEXT: Allows detailed descriptions without character limits.
* Price NUMERIC(10, 2): Chosen over FLOAT for precision with currency.
* Availability BOOLEAN DEFAULT TRUE: Allows toggling if an item is available; DEFAULT TRUE assumes items are generally available.

### **Restaurant\_Menu\_Items Table**

-- Restaurant Menu Items (M:M)

CREATE TABLE Restaurant\_Menu\_Items (

Restaurant\_ID INTEGER,

MenuItem\_ID INTEGER,

PRIMARY KEY (Restaurant\_ID, MenuItem\_ID),

FOREIGN KEY (Restaurant\_ID) REFERENCES Restaurant(Restaurant\_ID) ON DELETE CASCADE,

FOREIGN KEY (MenuItem\_ID) REFERENCES Menu\_Items(MenuItem\_ID) ON DELETE CASCADE

);

* Many-to-many relationship between restaurants and menu items.
* Composite primary key ensures uniqueness of each menu item per restaurant.
* Cascade behavior keeps join table in sync with parent deletions.

### **Cuisine\_Types Table**

-- Cuisine Types

CREATE TABLE Cuisine\_Types (

Cuisine\_ID SERIAL PRIMARY KEY,

Cuisine\_name VARCHAR(50) NOT NULL

);

* Defines categories like "Korean", "Mediterranean".
* VARCHAR(50) chosen to support descriptive but not overly long cuisine names.

### **Restaurant\_Cuisines Table**

CREATE TABLE Restaurant\_Cuisines (

Restaurant\_ID INTEGER,

Cuisine\_ID INTEGER,

PRIMARY KEY (Restaurant\_ID, Cuisine\_ID),

FOREIGN KEY (Restaurant\_ID) REFERENCES Restaurant(Restaurant\_ID) ON DELETE CASCADE,

FOREIGN KEY (Cuisine\_ID) REFERENCES Cuisine\_Types(Cuisine\_ID) ON DELETE CASCADE

);

* Another M:M join table.
* Composite PK ensures a cuisine can’t be assigned to the same restaurant more than once.

### **Delivery\_Personnel Table**

CREATE TABLE Delivery\_Personnel (

Personnel\_ID SERIAL PRIMARY KEY,

First\_name VARCHAR(50) NOT NULL,

Last\_name VARCHAR(50) NOT NULL,

Phone\_number VARCHAR(15),

Email VARCHAR(100),

Availability BOOLEAN NOT NULL DEFAULT TRUE

);

* Availability BOOLEAN NOT NULL DEFAULT TRUE: Ensures a new delivery person is assumed to be available unless specified otherwise.
* Other fields are consistent with those used for customers.

### **Payment\_Method Table**

CREATE TABLE Payment\_Method (

Payment\_ID SERIAL PRIMARY KEY,

Transaction\_id VARCHAR(100),

Payment\_type VARCHAR(50) NOT NULL,

Payment\_status VARCHAR(20) NOT NULL DEFAULT 'Pending',

Customer\_ID INTEGER NOT NULL,

FOREIGN KEY (Customer\_ID) REFERENCES Customer(Customer\_ID) ON DELETE CASCADE

);

* Allows storing multiple payment methods per customer.
* Transaction\_id: Useful if integrating external systems.
* Payment\_status DEFAULT 'Pending': Provides a default state before confirmation.

### **Orders Table**

CREATE TABLE Orders (

Order\_ID SERIAL PRIMARY KEY,

Delivery\_time TIMESTAMP,

Real\_time\_location VARCHAR(200),

Special\_Instructions TEXT,

Status VARCHAR(50) NOT NULL DEFAULT 'Processing',

OrderItem\_Price NUMERIC(10, 2) NOT NULL,

Taxes NUMERIC(10, 2) NOT NULL,

Delivery\_Fee NUMERIC(10, 2) NOT NULL,

Customer\_ID INTEGER NOT NULL,

Restaurant\_ID INTEGER NOT NULL,

Personnel\_ID INTEGER NOT NULL,

FOREIGN KEY (Customer\_ID) REFERENCES Customer(Customer\_ID) ON DELETE CASCADE,

FOREIGN KEY (Restaurant\_ID) REFERENCES Restaurant(Restaurant\_ID),

FOREIGN KEY (Personnel\_ID) REFERENCES Delivery\_Personnel(Personnel\_ID)

);

* Represents the core transactional unit.
* Delivery\_time uses PostgreSQL's TIMESTAMP type for flexible date/time storage.
* Real\_time\_location: Could integrate with a tracking system.
* OrderItem\_Price, Taxes, and Delivery\_Fee use precise numeric types to avoid rounding errors.
* All foreign keys link to their respective entities.

### **Order\_Item Table**

-- Order Item

CREATE TABLE Order\_Item (

OrderItem\_ID SERIAL PRIMARY KEY,

Quantity INTEGER NOT NULL,

Name VARCHAR(100) NOT NULL,

MenuItem\_ID INTEGER NOT NULL,

Order\_ID INTEGER NOT NULL,

FOREIGN KEY (MenuItem\_ID) REFERENCES Menu\_Items(MenuItem\_ID),

FOREIGN KEY (Order\_ID) REFERENCES Orders(Order\_ID) ON DELETE CASCADE

);

* OrderItem\_ID: Separate ID for flexibility (can be referenced independently if needed).
* Quantity: How many of the item were ordered.
* Name: Snapshot of item name at time of order (in case name changes later).
* MenuItem\_ID: Reference to original menu item.
* Order\_ID: Establishes the relationship to a specific order.

## **Design Decisions**

* **SERIAL vs. UUID**: SERIAL was chosen for simplicity and readability. It's auto-incrementing and works well for small to medium-scale systems.
* **VARCHAR(n) Sizes**: Chosen based on estimated max field length while preventing unnecessary space allocation.
* **BOOLEAN DEFAULT values**: For fields like Availability, this helps reduce developer burden by assuming typical defaults.
* **ON DELETE CASCADE**: Used wherever dependent data should be automatically removed for referential integrity.

## **Test Query Documentation for Food Delivery PostgreSQL Schema**

-- DROP TABLES IF THEY EXIST (for re-runs)

DROP TABLE IF EXISTS

Order\_Item, Orders, Payment\_Method, Delivery\_Personnel,

Restaurant\_Cuisines, Cuisine\_Types, Restaurant\_Menu\_Items, Menu\_Items, Restaurant,

Customer\_Addresses, Customer CASCADE;

Above code included for rerunning the test schema.

### **Query 1: Retrieve Order Details**

SELECT o.Order\_ID, c.First\_name, r.Name AS Restaurant, oi.Name AS Item, oi.Quantity

FROM Orders o

JOIN Customer c ON o.Customer\_ID = c.Customer\_ID

JOIN Order\_Item oi ON o.Order\_ID = oi.Order\_ID

JOIN Restaurant r ON o.Restaurant\_ID = r.Restaurant\_ID;

**Purpose**:

* Retrieve basic details about customer orders.
* Join multiple tables to show which customer ordered what item, from which restaurant.

**Explanation**:

* Orders o: base table representing each order.
* JOIN Customer c: gets the customer name.
* JOIN Order\_Item oi: retrieves what was ordered and how much.
* JOIN Restaurant r: links the restaurant name.

**Use Case**:

* Useful for order history dashboards or packing labels.

### **Query 2: Aggregate Revenue Per Restaurant**

SELECT

r.Name AS Restaurant,

COUNT(o.Order\_ID) AS Total\_Orders,

SUM(o.OrderItem\_Price + o.Taxes + o.Delivery\_Fee) AS Total\_Revenue

FROM Orders o

JOIN Restaurant r ON o.Restaurant\_ID = r.Restaurant\_ID

GROUP BY r.Name

ORDER BY Total\_Revenue DESC;

**Purpose**:

* Calculate the total number of orders and total revenue per restaurant.

**Explanation**:

* SUM(...): adds up each order's total amount including taxes and delivery.
* GROUP BY r.Name: groups results by restaurant.
* ORDER BY Total\_Revenue DESC: shows the top-earning restaurants first.

**Use Case**:

* Restaurant performance reports or business intelligence dashboards.

### **Query 3: Daily/Monthly Revenue**

SELECT

DATE\_TRUNC('day', o.Delivery\_time) AS Revenue\_Day,

SUM(o.OrderItem\_Price + o.Taxes + o.Delivery\_Fee) AS Total\_Revenue

FROM Orders o

GROUP BY Revenue\_Day

ORDER BY Revenue\_Day;

**Purpose**:

* Analyze revenue on a per-day or per-month basis.

**Explanation**:

* DATE\_TRUNC('day', ...): rounds timestamps to daily granularity.
* You can replace 'day' with 'month' for monthly reporting.

**Use Case**:

* Time-series financial tracking, trend reporting.

### **Query 4: Menu Item Popularity**

SELECT

mi.Description AS Menu\_Item,

COUNT(oi.OrderItem\_ID) AS Times\_Ordered

FROM Order\_Item oi

JOIN Menu\_Items mi ON oi.MenuItem\_ID = mi.MenuItem\_ID

GROUP BY mi.Description

ORDER BY Times\_Ordered DESC;

**Purpose**:

* Identify which menu items are most frequently ordered.

**Explanation**:

* Counts number of times each menu item appears in order records.

**Use Case**:

* Menu optimization, highlighting popular dishes.

### **Query 5: Delivery Personnel Workload**

SELECT

dp.First\_name || ' ' || dp.Last\_name AS Deliverer,

COUNT(o.Order\_ID) AS Orders\_Assigned

FROM Orders o

JOIN Delivery\_Personnel dp ON o.Personnel\_ID = dp.Personnel\_ID

GROUP BY dp.Personnel\_ID, dp.First\_name, dp.Last\_name

ORDER BY Orders\_Assigned DESC;

**Purpose**:

* Measure how many orders each delivery person handled.

**Explanation**:

* Joins orders to delivery personnel and groups by individual.

**Use Case**:

* Workload balancing, staffing analysis.

### **Query 6: Daily Revenue Per Restaurant**

SELECT

r.Name AS Restaurant,

DATE\_TRUNC('day', o.Delivery\_time) AS Revenue\_Day,

SUM(o.OrderItem\_Price + o.Taxes + o.Delivery\_Fee) AS Daily\_Revenue

FROM Orders o

JOIN Restaurant r ON o.Restaurant\_ID = r.Restaurant\_ID

GROUP BY r.Name, Revenue\_Day

ORDER BY r.Name, Revenue\_Day;

**Purpose**:

* Analyze daily revenue broken down by each restaurant.

**Explanation**:

* Joins orders to restaurants, truncates timestamps to the day, and groups by both restaurant and date.
* Revenue is summed per restaurant per day.

**Use Case**:

* Operational reporting for restaurant managers to see performance over time.