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

GROUP: David Xiao, Shadman Rakib, Raymond Li

## **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.

-- 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,

Name VARCHAR(100) NOT NULL,

Description TEXT,

Price NUMERIC(10, 2) NOT NULL,

Availability BOOLEAN NOT NULL DEFAULT TRUE

);

* MenuItem\_ID SERIAL PRIMARY KEY: Unique ID for each dish.
* Name VARCHAR(100): Used 100 as max length, most reasonable length for menu item name.
* 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**

-- Secure Payment Method (Reusable)

CREATE TABLE Payment\_Method (

PaymentMethod\_ID SERIAL PRIMARY KEY,

Customer\_ID INTEGER NOT NULL,

Method\_Type VARCHAR(50) NOT NULL,

Masked\_Details VARCHAR(100),

Provider\_Method\_ID VARCHAR(100),

Last\_Used TIMESTAMP,

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

);

* Allows storing multiple payment methods per customer.
* PaymentMethod\_ID allows linking
* Customer\_ID is the customer the payment method is attached to
* Masked\_Details, Payment method information is only kept in a masked state \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*4325 ex. This is to prevent holding actual payment method data for security reasons.
* Provider\_Method\_ID: Rather, Payment information is held with the payment provider, using the token provided for the method. This avoids holding the actual payment method information while still allowing payment processing.
* Last\_Used: Keeps track of when this was last used for tracking reasons.

### **Orders Table**

-- Orders

CREATE TABLE Orders (

Order\_ID SERIAL PRIMARY KEY,

Created\_At TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

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,

Delivery\_Address\_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),

FOREIGN KEY (Delivery\_Address\_ID) REFERENCES Customer\_Addresses(Address\_ID)

);

* Represents the core transactional unit.
* Created\_At: helps to keep track of when the order was made. Uses TIMESTAMP, and defaults to the CURRENT\_TIMESTAMP
* 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,

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.
* MenuItem\_ID: Reference to original menu item.
* Order\_ID: Establishes the relationship to a specific order.

## **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: Retrieve all orders by a customer**

-- Query 1: All orders by customer ID 1

SELECT

o.Order\_ID,

o.Created\_At,

o.Status,

o.OrderItem\_Price,

o.Taxes,

o.Delivery\_Fee,

(o.OrderItem\_Price + o.Taxes + o.Delivery\_Fee) AS Total\_Amount,

r.Name AS Restaurant\_Name

FROM Orders o

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

WHERE o.Customer\_ID = 1

ORDER BY o.Created\_At DESC;

**Purpose: View historical orders placed by a given customer.**

**Explanation:**

* **Orders o: Retrieves the main order data.**
* **JOIN Restaurant: Fetches restaurant name associated with the order.**
* **WHERE clause: Filters for a specific customer.**
* **ORDER BY: Orders results from newest to oldest.**

**Use Case: Useful in customer profiles or order history pages.**

### **-- Query: Show all menu items for a restaurant**

-- Query 2: Menu items for restaurant ID 1

SELECT

mi.MenuItem\_ID,

mi.Name AS Item\_Name,

mi.Description,

mi.Price,

mi.Availability

FROM Menu\_Items mi

JOIN Restaurant\_Menu\_Items rmi ON mi.MenuItem\_ID = rmi.MenuItem\_ID

WHERE rmi.Restaurant\_ID = 1

ORDER BY mi.Name;

**Purpose: Display the current menu of a restaurant.**

**Explanation:**

* **Menu\_Items mi: Base menu items.**
* **JOIN with Restaurant\_Menu\_Items: Establish which items are served at which restaurants.**
* **Filters by specific restaurant.**

**Use Case: Menu rendering on restaurant detail views.**

### **-- Query: Show items within a specific order**

-- Query 3: Items in order ID 1

SELECT

oi.OrderItem\_ID,

mi.Name AS Item\_Name,

mi.Description,

mi.Price AS Unit\_Price,

oi.Quantity,

(mi.Price \* oi.Quantity) AS Line\_Total

FROM Order\_Item oi

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

WHERE oi.Order\_ID = 1

ORDER BY oi.OrderItem\_ID;

**Purpose: Show what items were included in a particular order.**

**Explanation:**

* **Order\_Item oi: Fetches each line item in the order.**
* **JOIN with Menu\_Items: Retrieves names, descriptions, and prices.**
* **Uses multiplication to calculate line totals.**

**Use Case: Order summaries and receipts.**

### **-- Query: Total and status of a specific order**

-- Query 4: Total amount for order ID 1

SELECT

o.Order\_ID,

o.OrderItem\_Price,

o.Taxes,

o.Delivery\_Fee,

(o.OrderItem\_Price + o.Taxes + o.Delivery\_Fee) AS Total\_Amount,

p.Payment\_Status

FROM Orders o

LEFT JOIN Payments p ON o.Order\_ID = p.Order\_ID

WHERE o.Order\_ID = 1;

**Purpose: Retrieve full charge breakdown and payment status.**

**Explanation:**

* **Orders o: Gets pricing info.**
* **LEFT JOIN with Payments: Gets payment status (if available).**
* **SUM fields into total amount.**

**Use Case: Order tracking and billing confirmation.**

### **-- Query: Restaurants serving a cuisine**

-- Query 5: Restaurants offering "Mexican"

SELECT

r.Restaurant\_ID,

r.Name AS Restaurant\_Name,

r.Street\_address,

r.City,

r.State,

r.Phone\_number,

ct.Cuisine\_name

FROM Restaurant r

JOIN Restaurant\_Cuisines rc ON r.Restaurant\_ID = rc.Restaurant\_ID

JOIN Cuisine\_Types ct ON rc.Cuisine\_ID = ct.Cuisine\_ID

WHERE ct.Cuisine\_name = 'Mexican'

ORDER BY r.Name;

**Purpose: Filter restaurants by cuisine type.**

**Explanation:**

* **JOINs traverse the many-to-many relationship.**
* **Filters for a specific cuisine name.**

**Use Case: Search filters and discovery pages.**

### **-- Query: Show active (in-progress) orders**

-- Query 6: Orders not yet delivered

SELECT

o.Order\_ID,

o.Created\_At,

o.Status,

c.First\_name || ' ' || c.Last\_name AS Customer\_Name,

r.Name AS Restaurant\_Name,

dp.First\_name || ' ' || dp.Last\_name AS Delivery\_Person,

ca.Street\_address || ', ' || ca.City AS Delivery\_Address

FROM Orders o

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

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

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

JOIN Customer\_Addresses ca ON o.Delivery\_Address\_ID = ca.Address\_ID

WHERE o.Status IN ('Processing', 'Preparing', 'Out for Delivery')

ORDER BY o.Created\_At;

**Purpose: Monitor all orders that are still underway.**

**Explanation:**

* **Uses WHERE ... IN (...) to check multiple status types.**
* **Combines customer and courier names.**

**Use Case: Logistics dashboards and live tracking.**

**-- Query: Customer's most recent order SELECT \* FROM Orders WHERE Customer\_ID = 1 ORDER BY Created\_At DESC LIMIT 1;**

### **-- Query: Show most recent order by Customer**

-- Query 7: Most recent order by customer 1

SELECT

o.Order\_ID,

o.Created\_At,

o.Status,

r.Name AS Restaurant\_Name,

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

FROM Orders o

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

WHERE o.Customer\_ID = 1

ORDER BY o.Created\_At DESC

LIMIT 1;

**Purpose: Retrieve the last order a customer placed.**

**Explanation:**

* **ORDER BY and LIMIT ensure only the latest is shown.**

**Use Case: Reorder suggestions, default address prefilling.**

### **-- Query: Completed orders within a date range**

-- Query 8: Completed orders from '2025-05-01' to '2025-05-21'

SELECT

o.Order\_ID,

o.Created\_At,

c.First\_name || ' ' || c.Last\_name AS Customer\_Name,

r.Name AS Restaurant\_Name,

(o.OrderItem\_Price + o.Taxes + o.Delivery\_Fee) AS Total\_Amount,

dp.First\_name || ' ' || dp.Last\_name AS Delivery\_Person

FROM Orders o

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

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

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

WHERE o.Status = 'Delivered'

AND o.Created\_At >= '2025-05-01'

AND o.Created\_At < '2025-05-21'::DATE + INTERVAL '1 day'

ORDER BY o.Created\_At DESC;

**Purpose: Report on deliveries made within a given time window.**

**Explanation:**

* **Filters by order status and creation date.**
* **Adds pricing info and restaurant name.**

**Use Case: Financial reporting, performance analysis.**

**-- Query: All currently available delivery personnel SELECT \* FROM Delivery\_Personnel WHERE Availability = TRUE;**

### **-- Query: All currently available delivery personnel**

-- Query 9: Available delivery personnel

SELECT

Personnel\_ID,

First\_name || ' ' || Last\_name AS Full\_Name,

Phone\_number,

Email,

Availability

FROM Delivery\_Personnel

WHERE Availability = TRUE

ORDER BY Last\_name, First\_name;

**Purpose: Find all delivery staff who are marked as available.**

**Explanation:**

* **Simple WHERE clause checks availability flag.**

**Use Case: Dispatching, shift planning.**

### **-- Query: Most frequently ordered items**

-- Query 10: Most popular menu items

SELECT

mi.MenuItem\_ID,

mi.Name AS Item\_Name,

mi.Description,

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

SUM(oi.Quantity) AS Total\_Quantity\_Ordered,

AVG(mi.Price) AS Average\_Price,

STRING\_AGG(DISTINCT r.Name, ', ') AS Available\_At\_Restaurants

FROM Menu\_Items mi

JOIN Order\_Item oi ON mi.MenuItem\_ID = oi.MenuItem\_ID

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

JOIN Restaurant\_Menu\_Items rmi ON mi.MenuItem\_ID = rmi.MenuItem\_ID

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

GROUP BY mi.MenuItem\_ID, mi.Name, mi.Description

ORDER BY Total\_Quantity\_Ordered DESC, Times\_Ordered DESC

LIMIT 20;

**Purpose: Discover top-selling food items.**

**Explanation:**

* **Groups and counts how many times each item was ordered.**
* **Sorted descending to highlight best sellers.**

**Use Case: Menu optimization, popularity badges.**