Problem Statement: GlobalRides, a company similar to Uber and Uber Eats, operates a platform connecting riders, drivers, food delivery customers, and restaurants. The company seeks to design a relational database to effectively manage its multifaceted operations, including Riders, Drivers, Customers, Restaurants, Rides, Food Orders, Payments, and Reviews.

A user can serve multiple roles, such as being a Rider, a Customer, a Driver, and a Restaurant Owner. All users share common attributes, including User ID, Name (First, Middle, Last), Contact Details, Address, Gender, and Date of Birth. A user may provide multiple contact numbers.

For Rides, each booking has attributes like Ride ID, Pickup and Drop-off Locations, Pickup Time, Ride Fare, and Payment Status. Rides are associated with Drivers, who can be assigned multiple rides, while each ride can only have one driver. Drivers have specific attributes such as Driver ID, License Details, Vehicle Information, and Experience.

For Food Orders, customers can place orders with Restaurants listed on the platform. Each order records Order ID, Restaurant ID, Customer ID, Order Date, Delivery Status, Total Amount, and Payment Method. Orders consist of multiple items from the restaurant's menu. Each restaurant manages its menu, including attributes like Item ID, Name, Description, Price, and Food Category (e.g., Appetizers, Main Course, Desserts). Restaurants have attributes such as Restaurant ID, Name, Address, Cuisine, Operational Hours, and Ownership details. Restaurants can also run promotions tied to specific menu items. These promotions are unique within the restaurant and include Promotion ID, Description, and Validity Period.

Riders and Customers can leave reviews. Reviews have attributes like Review ID, Rating, Feedback Text, and Date, linked to the specific ride, food item, or restaurant being reviewed.

Employees of GlobalRides are integral to the company's operations and are categorized into several distinct roles: Platform Managers, Support Agents, and Delivery Coordinators. Each employee has an Employee ID, a unique identifier following a predefined format such as "E###," where "###" represents a sequence of digits. Employees must be at least 18 years old. Additional general attributes include their Start Date, which records when the employee joined the organization, and Department, signifying the specific area within the company they are associated with.

Delivery Coordinators are tasked with managing the logistics of delivery drivers, focusing on ensuring timely and efficient order deliveries. They are responsible for

assigning drivers to orders, optimizing routes, and addressing operational challenges such as delays or vehicle issues. Their role is crucial in maintaining the smooth flow of delivery operations and upholding service quality standards.

GlobalRides, a company similar to Uber and Uber Eats, operates a platform connecting riders, drivers, food delivery customers, and restaurants. The company seeks to design a relational database to effectively manage its multifaceted operations, including Riders, Drivers, Customers, Restaurants, Rides, Food Orders, Payments, and Reviews.

A user can serve multiple roles, such as being a Rider, a Customer, a Driver, and a Restaurant Owner. All users share common attributes, including User ID, Name (First, Middle, Last), Contact Details, Address, Gender, and Date of Birth. A user may provide multiple contact numbers.

For Rides, each booking has attributes like Ride ID, Pickup and Drop-off Locations, Pickup Time, Ride Fare, and Payment Status. Rides are associated with Drivers, who can be assigned multiple rides, while each ride can only have one driver. Drivers have specific attributes such as Driver ID, License Details, Vehicle Information, and Experience.

For Food Orders, customers can place orders with Restaurants listed on the platform. Each order records Order ID, Restaurant ID, Customer ID, Order Date, Delivery Status, Total Amount, and Payment Method. Orders consist of multiple items from the restaurant's menu. Each restaurant manages its menu, including attributes like Item ID, Name, Description, Price, and Food Category (e.g., Appetizers, Main Course, Desserts). Restaurants have attributes such as Restaurant ID, Name, Address, Cuisine, Operational Hours, and Ownership details. Restaurants can also run promotions tied to specific menu items. These promotions are unique within the restaurant and include Promotion ID, Description, and Validity Period.

Riders and Customers can leave reviews. Reviews have attributes like Review ID, Rating, Feedback Text, and Date, linked to the specific ride, food item, or restaurant being reviewed.

Employees of GlobalRides are integral to the company's operations and are categorized into several distinct roles: Platform Managers, Support Agents, and Delivery Coordinators. Each employee has an Employee ID, a unique identifier following a predefined format such as "E###," where "###" represents a sequence of digits. Employees must be at least 18 years old. Additional general attributes include their Start Date, which records when the employee joined the organization, and Department, signifying the specific area within the company they are associated with.

Delivery Coordinators are tasked with managing the logistics of delivery drivers, focusing on ensuring timely and efficient order deliveries. They are responsible for

assigning drivers to orders, optimizing routes, and addressing operational challenges such as delays or vehicle issues. Their role is crucial in maintaining the smooth flow of delivery operations and upholding service quality standards.

Project Questions

1. Yes, a superclass/subclass relationship is very beneficial to the GlobalRides database because there are many entities that share common attributes. Specifically Users, and all their subtypes (Customers, Drivers, RestaurantOwners) as well as Employers and all of their subtypes. Its optimal for situations like that because it helps to reduce redundancy and enforce consistency

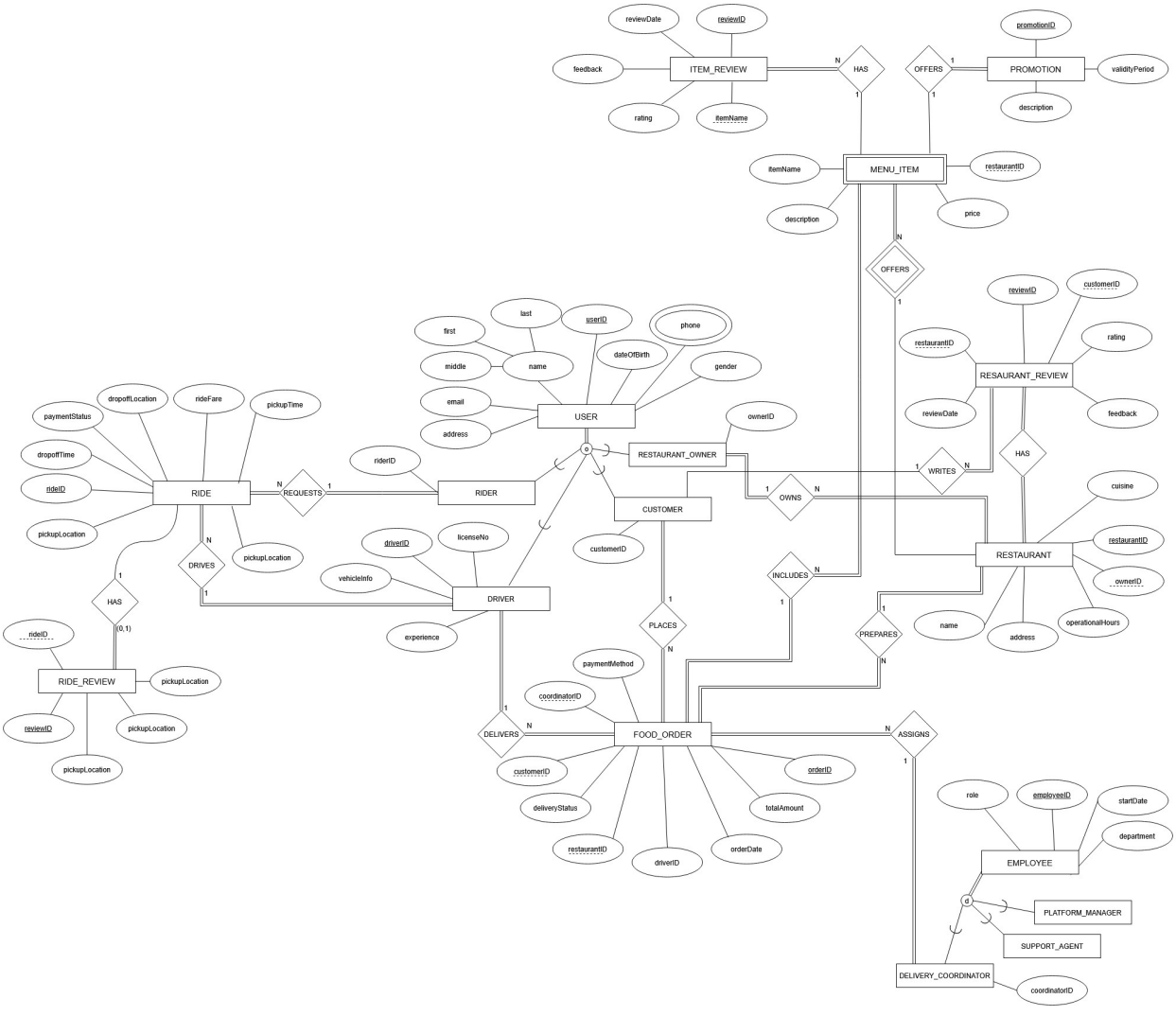
2. 5 possible additional rules

Rule	Description	Schema/Constraint Change
1. One active order per driver	A driver may not be assigned a second delivery until the first is marked “Delivered.”	Add a trigger on Food_Orders that prevents INSERT/UPDATE of driver_id when they have an existing order with delivery_status <> 'Delivered'.
2. Review only after completion	Customers can only leave a ride or food review once the ride/order is completed.	Enforce via a FOREIGN KEY to only allow insertion into Ride_Reviews/Restaurant_Reviews when the corresponding dropoff_time IS NOT NULL or delivery_status='Delivered'. Could also use a trigger for business-rule validation.
3. Promotion date overlap	A restaurant cannot run two promotions whose date ranges overlap.	On Promotions, add a UNIQUE constraint on (restaurant_id, valid_from, valid_to) via an exclusion constraint.
4. Driver license validation	Every driver must have a valid (non-expired) license on file.	Extend Drivers with license_number and license_expiry DATE NOT NULL and add a CHECK constraint license_expiry ≥ CURRENT_DATE.

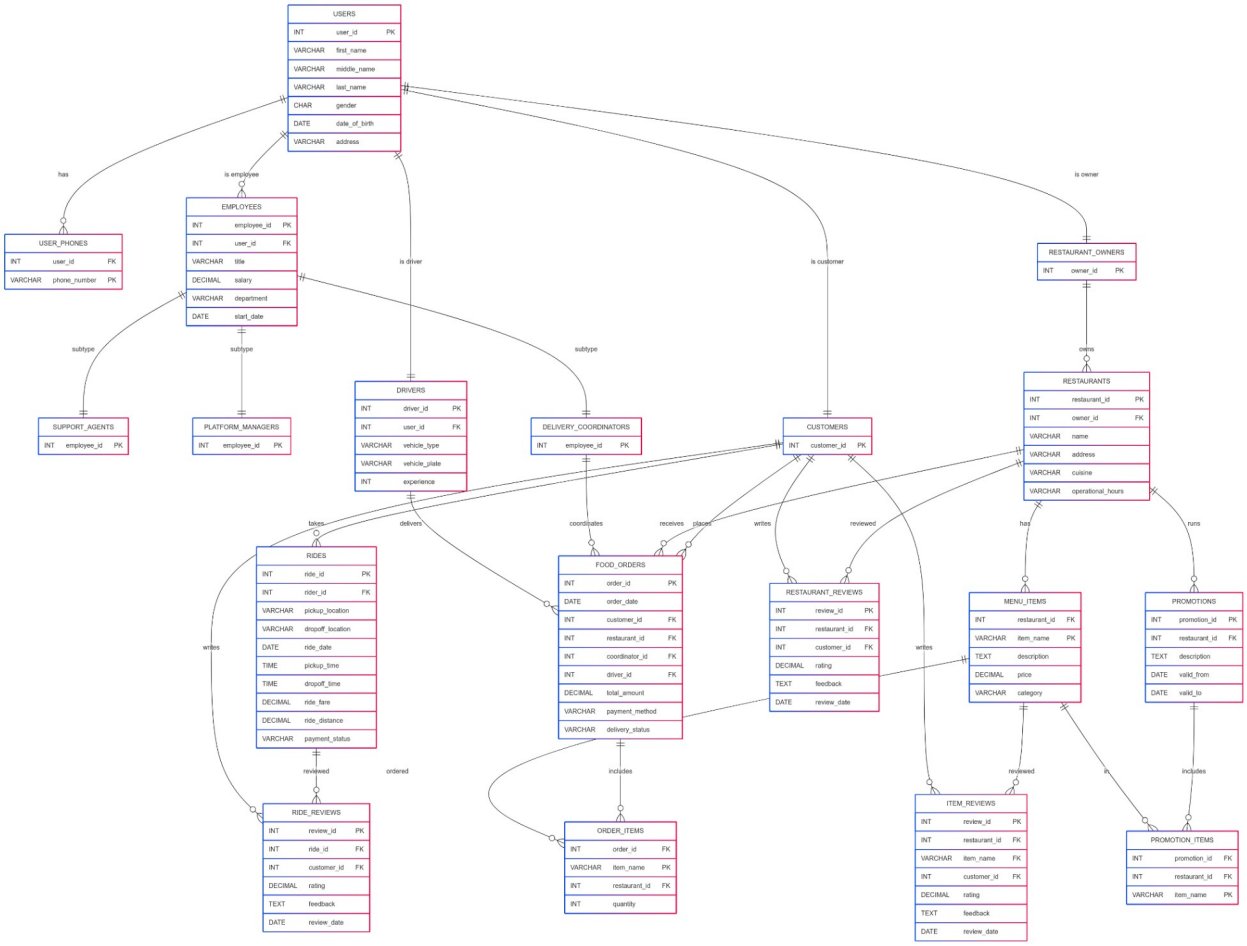
5. Minimum menu portfolio	Restaurants must offer at least 3 distinct menu items.	Use a trigger on DELETE from Menu_Items or CREATE of a restaurant, verifies that COUNT(*) ≥ 3.

3. A Relational DMBS is optimal in a situation like GlobalRides because it allows for complex queries to find out important information as the number of restaurants, orders, rides, users, and employees scales higher and higher. It also assists in minimizing redundant or incorrect data, and correctly handles the relationships between all possible entities.

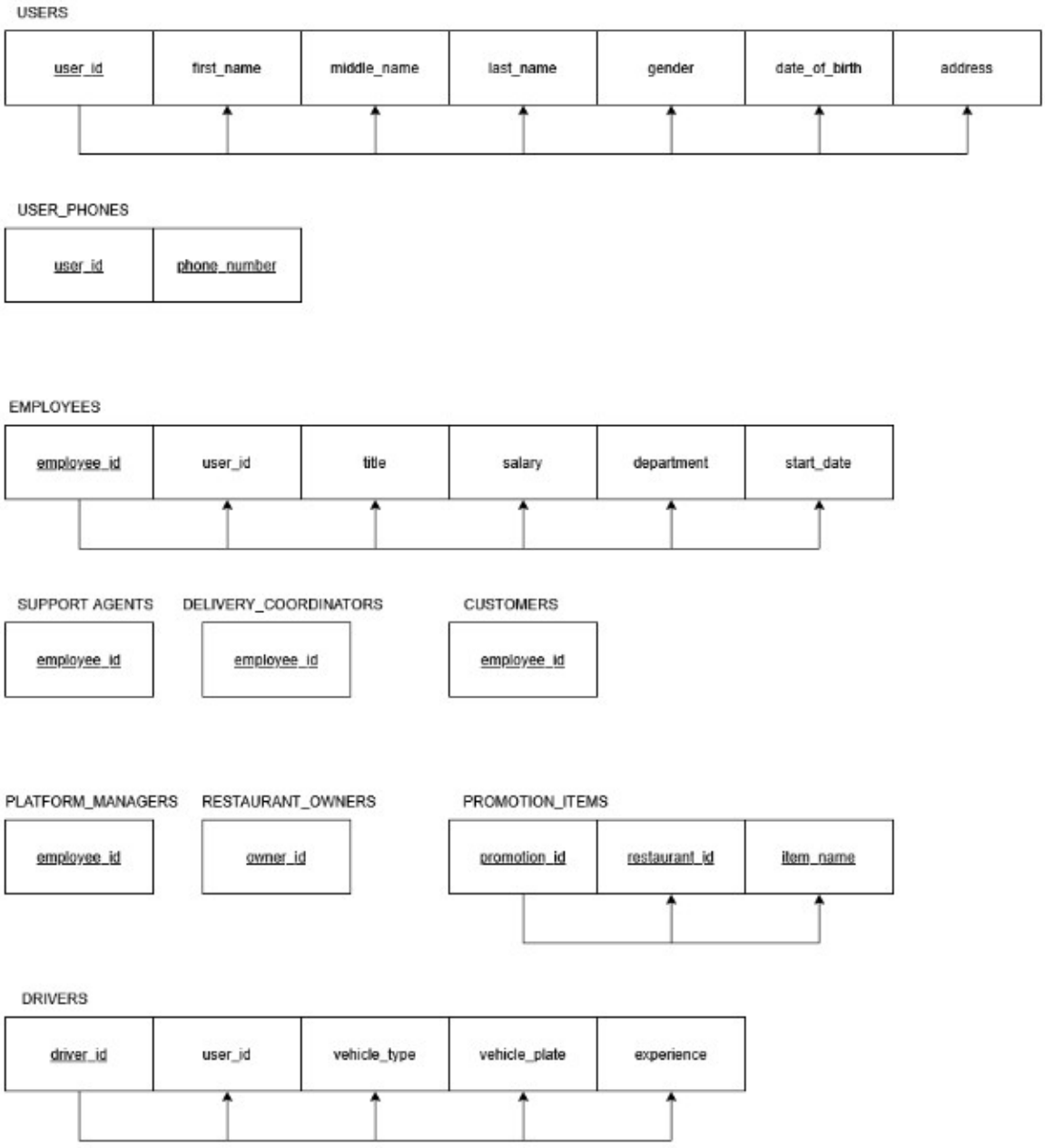
ERR Diagram

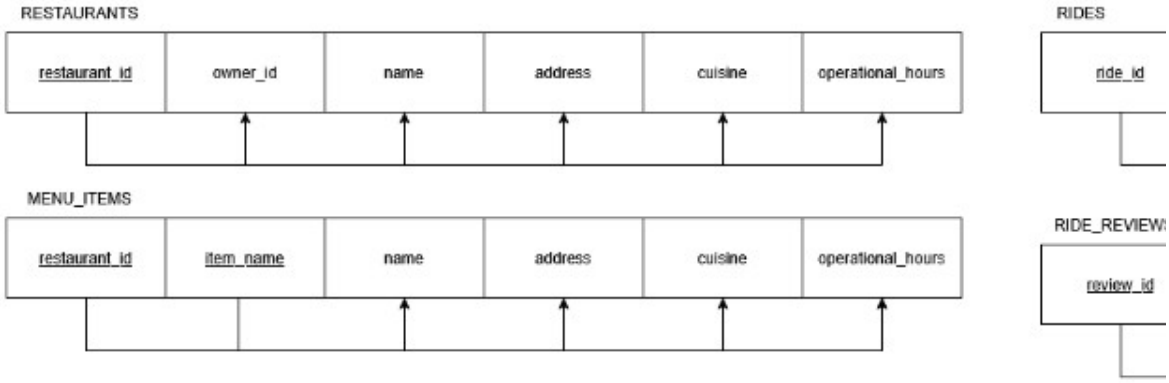


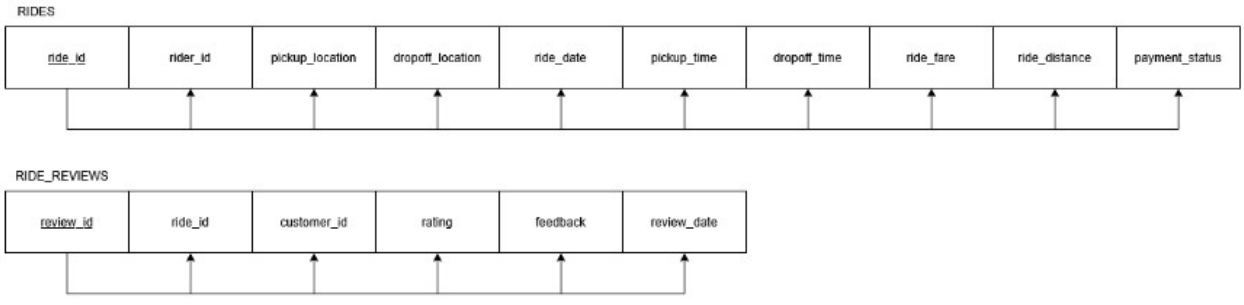
Relational Schema


USERS
INT user_id PK
VARCHAR first_name
VARCHAR middle_name
VARCHAR last_name
CHAR gender
DATE date_of_birth
VARCHAR address
has
is employee
is driver
is customer
is owner
USER_PHONES
INT user_id FK
VARCHAR phone_number PK
EMPLOYEES
INT employee_id PK
INT user_id FK
VARCHAR title
DECIMAL salary
VARCHAR department
DATE start_date
RESTAURANT_OWNERS
INT owner_id PK
subtype
subtype
subtype
owns
SUPPORT_AGENTS
INT employee_id PK
PLATFORM_MANAGERS
INT employee_id PK
DRIVERS
INT driver_id PK
INT user_id FK
VARCHAR vehicle_type
VARCHAR vehicle_plate
INT experience
DELIVERY_COORDINATORS
INT employee_id PK
CUSTOMERS
INT customer_id PK
RESTAURANTS
INT restaurant_id PK
INT owner_id FK
VARCHAR name
VARCHAR address
VARCHAR cuisine
VARCHAR operational_hours
takes
delivers
coordinates
receives
places
writes
reviewed
has
runs
RIDES
INT ride_id PK
INT rider_id FK
VARCHAR pickup_location
VARCHAR dropoff_location
DATE ride_date
TIME pickup_time
TIME dropoff_time
DECIMAL ride_fare
DECIMAL ride_distance
VARCHAR payment_status
FOOD_ORDERS
INT order_id PK
DATE order_date
INT customer_id FK
INT restaurant_id FK
INT coordinator_id FK
INT driver_id FK
DECIMAL total_amount
VARCHAR payment_method
VARCHAR delivery_status
RESTAURANT_REVIEWS
INT review_id PK
INT restaurant_id FK
INT customer_id FK
DECIMAL rating
TEXT feedback
DATE review_date
MENU_ITEMS
INT restaurant_id FK
VARCHAR item_name PK
TEXT description
DECIMAL price
VARCHAR category
PROMOTIONS
INT promotion_id PK
INT restaurant_id FK
TEXT description
DATE valid_from
DATE valid_to
writes
writes
reviewed
ordered
includes
reviewed
in
includes
RIDE_REVIEWS
INT review_id PK
INT ride_id FK
INT customer_id FK
DECIMAL rating
TEXT feedback
DATE review_date
ORDER_ITEMS
INT order_id FK
VARCHAR item_name PK
INT restaurant_id FK
INT quantity
ITEM_REVIEWS
INT review_id PK
INT restaurant_id FK
VARCHAR item_name FK
INT customer_id FK
DECIMAL rating
TEXT feedback
DATE review_date
PROMOTION_ITEMS
INT promotion_id FK
INT restaurant_id FK
VARCHAR item_name PK

Dependency Diagrams




RESTAURANTS
restaurant_id
owner_id
name
address
cuisine
operational_hours
RIDES
ride_id
MENU_ITEMS
restaurant_id
item_name
name
address
cuisine
operational_hours
RIDE_REVIEW
review_id
PROMOTIONS
promotion_id
restaurant_id
description
valid_from
valid_to
ORDER_ITEMS
order_id
item_name
restaurant_id
quantity
RESTAURANT_REVIEWS
review_id
restaurant_id
customer_id
rating
feedback
review_date
ITEM_REVIEWS
review_id
restaurant_id
item_name
customer_id
rating
feedback
review_date


RIDES
ride_id
rider_id
pickup_location
dropoff_location
ride_date
pickup_time
dropoff_time
ride_fare
ride_distance
payment_status
RIDE_REVIEWS
review_id
ride_id
customer_id
rating
feedback
review_date

SQL

Schema

```
-- USERS
CREATE TABLE Users (
  user_id       INT             PRIMARY KEY AUTO_INCREMENT,
  first_name            VARCHAR(50)       NOT NULL,
  middle_name       VARCHAR(50),
  last_name             VARCHAR(50)       NOT NULL,
  gender        CHAR(1)       CHECK (gender IN ('M','F','O')),
  date_of_birth         DATE            NOT NULL,
  address               VARCHAR(200)
);

-- USER_PHONES
CREATE TABLE User_Phones (
  user_id       INT             NOT NULL,
  phone_number      VARCHAR(20)       NOT NULL,
  PRIMARY KEY(user_id, phone_number),
  FOREIGN KEY(user_id) REFERENCES Users(user_id) ON DELETE CASCADE
);

-- EMPLOYEES & subtypes
CREATE TABLE Employees (
  employee_id         INT             PRIMARY KEY AUTO_INCREMENT,
  user_id       INT             NOT NULL UNIQUE,
  title             VARCHAR(50),
  salary        DECIMAL(10,2),
  department            VARCHAR(50),
  start_date            DATE            NOT NULL,
  FOREIGN KEY(user_id) REFERENCES Users(user_id) ON DELETE RESTRICT
);
CREATE TABLE Support_Agents (
  employee_id         INT PRIMARY KEY,
  FOREIGN KEY(employee_id) REFERENCES Employees(employee_id)
);
CREATE TABLE Platform_Managers (
  employee_id         INT PRIMARY KEY,
```

```
  FOREIGN KEY(employee_id) REFERENCES Employees(employee_id)
);
CREATE TABLE Delivery_Coordinators (
  employee_id         INT PRIMARY KEY,
  FOREIGN KEY(employee_id) REFERENCES Employees(employee_id)
);

-- CUSTOMERS, DRIVERS, OWNERS
CREATE TABLE Customers (
  customer_id         INT PRIMARY KEY,
  FOREIGN KEY(customer_id) REFERENCES Users(user_id)
);
CREATE TABLE Drivers (
  driver_id     INT PRIMARY KEY,
  user_id       INT     NOT NULL UNIQUE,
  vehicle_type          VARCHAR(30),
  vehicle_plate         VARCHAR(20),
  experience            INT,
  FOREIGN KEY(driver_id) REFERENCES Users(user_id)
);
CREATE TABLE Restaurant_Owners (
  owner_id              INT PRIMARY KEY,
  FOREIGN KEY(owner_id) REFERENCES Users(user_id)
);

-- RESTAURANTS
CREATE TABLE Restaurants (
  restaurant_id         INT             PRIMARY KEY AUTO_INCREMENT,
  owner_id              INT             NOT NULL,
  name          VARCHAR(100)   NOT NULL,
  address               VARCHAR(200),
  cuisine       VARCHAR(50),
  operational_hours VARCHAR(100),
  FOREIGN KEY(owner_id) REFERENCES Restaurant_Owners(owner_id)
);

-- MENU items
CREATE TABLE Menu_Items (
  restaurant_id         INT             NOT NULL,
  item_name             VARCHAR(100) NOT NULL,
```

```
  description           TEXT,
  price         DECIMAL(8,2),
  category              VARCHAR(50),
  PRIMARY KEY(restaurant_id, item_name),
  FOREIGN KEY(restaurant_id) REFERENCES Restaurants(restaurant_id)
);

-- PROMOTIONS & link to items
CREATE TABLE Promotions (
  promotion_id      INT           PRIMARY KEY AUTO_INCREMENT,
  restaurant_id     INT           NOT NULL,
  description           TEXT,
  valid_from            DATE,
  valid_to      DATE,
  FOREIGN KEY(restaurant_id) REFERENCES Restaurants(restaurant_id)
);
CREATE TABLE Promotion_Items (
  promotion_id      INT           NOT NULL,
  restaurant_id     INT           NOT NULL,
  item_name             VARCHAR(100)   NOT NULL,
  PRIMARY KEY(promotion_id, restaurant_id, item_name),
  FOREIGN KEY(promotion_id) REFERENCES Promotions(promotion_id),
  FOREIGN KEY(restaurant_id, item_name)
        REFERENCES Menu_Items(restaurant_id, item_name)
);

-- FOOD ORDERS & their items
CREATE TABLE Food_Orders (
  order_id      INT             PRIMARY KEY AUTO_INCREMENT,
  order_date            DATE          NOT NULL,
  customer_id           INT           NOT NULL,
  restaurant_id         INT           NOT NULL,
  coordinator_id        INT,
  driver_id     INT,
  total_amount          DECIMAL(10,2),
  payment_method    VARCHAR(20),
  delivery_status   VARCHAR(20),
  FOREIGN KEY(customer_id) REFERENCES Customers(customer_id),
  FOREIGN KEY(restaurant_id) REFERENCES Restaurants(restaurant_id),
  FOREIGN KEY(coordinator_id) REFERENCES Delivery_Coordinators(employee_id),
```

```
  FOREIGN KEY(driver_id) REFERENCES Drivers(driver_id)
);
CREATE TABLE Order_Items (
  order_id      INT             NOT NULL,
  item_name           VARCHAR(100)   NOT NULL,
  restaurant_id       INT             NOT NULL,
  quantity      INT             NOT NULL CHECK (quantity>0),
  PRIMARY KEY(order_id, item_name),
  FOREIGN KEY(order_id) REFERENCES Food_Orders(order_id) ON DELETE
CASCADE,
  FOREIGN KEY(restaurant_id, item_name)
        REFERENCES Menu_Items(restaurant_id, item_name)
);

-- REVIEWS
CREATE TABLE Restaurant_Reviews (
  review_id           INT             PRIMARY KEY AUTO_INCREMENT,
  restaurant_id       INT             NOT NULL,
  customer_id         INT             NOT NULL,
  rating        DECIMAL(2,1)   NOT NULL CHECK (rating BETWEEN 0 AND 5),
  feedback            TEXT,
  review_date         DATE            NOT NULL,
  FOREIGN KEY(restaurant_id) REFERENCES Restaurants(restaurant_id),
  FOREIGN KEY(customer_id) REFERENCES Customers(customer_id)
);
CREATE TABLE Item_Reviews (
  review_id           INT             PRIMARY KEY AUTO_INCREMENT,
  restaurant_id       INT             NOT NULL,
  item_name           VARCHAR(100)   NOT NULL,
  customer_id         INT             NOT NULL,
  rating        DECIMAL(2,1)   NOT NULL CHECK (rating BETWEEN 0 AND 5),
  feedback            TEXT,
  review_date         DATE            NOT NULL,
  FOREIGN KEY(restaurant_id, item_name)
        REFERENCES Menu_Items(restaurant_id, item_name),
  FOREIGN KEY(customer_id) REFERENCES Customers(customer_id)
);

-- RIDES & their reviews
CREATE TABLE Rides (
```

```
  ride_id        INT             PRIMARY KEY AUTO_INCREMENT,
  rider_id       INT             NOT NULL,
  pickup_location   VARCHAR(200),
  dropoff_location  VARCHAR(200),
  ride_date             DATE          NOT NULL,
  pickup_time           TIME,
  dropoff_time          TIME,
  ride_fare     DECIMAL(8,2),
  ride_distance         DECIMAL(6,2),
  payment_status     VARCHAR(20),
  FOREIGN KEY(rider_id) REFERENCES Customers(customer_id)
);
CREATE TABLE Ride_Reviews (
  review_id             INT             PRIMARY KEY AUTO_INCREMENT,
  ride_id        INT             NOT NULL,
  customer_id           INT             NOT NULL,
  rating          DECIMAL(2,1)   NOT NULL CHECK (rating BETWEEN 0 AND 5),
  feedback              TEXT,
  review_date           DATE          NOT NULL,
  FOREIGN KEY(ride_id) REFERENCES Rides(ride_id),
  FOREIGN KEY(customer_id) REFERENCES Customers(customer_id)
);
```

Create_view

```
-- 1. LoyalCustomers: orders every month in the past year
CREATE VIEW LoyalCustomers AS
SELECT customer_id
FROM Food_Orders
WHERE order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 1 YEAR)
GROUP BY customer_id
HAVING COUNT(DISTINCT MONTH(order_date)) = 12;

-- 2. TopRatedRestaurants: avg rating ≥4.5 in past 6 months
CREATE VIEW TopRatedRestaurants AS
SELECT restaurant_id
FROM Restaurant_Reviews
WHERE review_date >= DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH)
GROUP BY restaurant_id
HAVING AVG(rating) >= 4.5;

-- 3. ActiveDrivers: ≥20 deliveries in last 2 weeks
```

```
CREATE VIEW ActiveDrivers AS
SELECT driver_id
FROM Food_Orders
WHERE driver_id IS NOT NULL
  AND order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 14 DAY)
GROUP BY driver_id
HAVING COUNT(*) >= 20;

-- 4. PopularMenuItems: top 10 by qty in past 3 months
CREATE VIEW PopularMenuItems AS
SELECT oi.restaurant_id, oi.item_name,
          SUM(oi.quantity) AS total_ordered
FROM Order_Items oi
JOIN Food_Orders fo ON fo.order_id = oi.order_id
WHERE fo.order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 3 MONTH)
GROUP BY oi.restaurant_id, oi.item_name
ORDER BY total_ordered DESC
LIMIT 10;

-- 5. ProminentOwners: owners w/ ≥50 combined orders last month
CREATE VIEW ProminentOwners AS
SELECT ro.owner_id
FROM Restaurants r
JOIN Food_Orders fo ON fo.restaurant_id = r.restaurant_id
JOIN Restaurant_Owners ro ON ro.owner_id = r.owner_id
WHERE fo.order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
GROUP BY ro.owner_id
HAVING COUNT(*) >= 50;
```

Queries

```
-- TopEarningDrivers: top 5 drivers by sum(ride_fare + order delivery fees)
SELECT d.driver_id, u.first_name, u.last_name,
          SUM(COALESCE(fo.total_amount,0)) AS total_earnings
FROM Drivers d
JOIN Users u             ON u.user_id = d.driver_id
LEFT JOIN Food_Orders fo
  ON fo.driver_id = d.driver_id
GROUP BY d.driver_id, u.first_name, u.last_name
ORDER BY total_earnings DESC
LIMIT 5;

-- HighSpendingCustomers: spent >1000
SELECT c.customer_id, u.first_name, u.last_name,
```

```
        SUM(fo.total_amount) AS total_spent
FROM Customers c
JOIN Users u             ON u.user_id = c.customer_id
JOIN Food_Orders fo ON fo.customer_id = c.customer_id
GROUP BY c.customer_id, u.first_name, u.last_name
HAVING SUM(fo.total_amount) > 1000;

-- FrequentReviewers: ≥10 reviews + avg rating
SELECT rv.customer_id, u.first_name, u.last_name,
        COUNT(*)      AS num_reviews,
        AVG(rv.rating) AS avg_rating
FROM (
  SELECT customer_id, rating FROM Restaurant_Reviews
  UNION ALL
  SELECT customer_id, rating FROM Item_Reviews
) rv
JOIN Users u ON u.user_id = rv.customer_id
GROUP BY rv.customer_id, u.first_name, u.last_name
HAVING COUNT(*) >= 2;

-- InactiveRestaurants: no orders in past month
SELECT r.restaurant_id, r.name
FROM Restaurants r
LEFT JOIN Food_Orders fo
  ON fo.restaurant_id = r.restaurant_id
        AND fo.order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
        AND fo.order_date <= CURDATE()
WHERE fo.order_id IS NULL;

-- PeakOrderDay: weekday w/ most orders last month
SELECT DAYNAME(order_date) AS weekday,
        COUNT(*)      AS orders_count
FROM Food_Orders
WHERE order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
AND order_date <= CURDATE()
GROUP BY DAYNAME(order_date)
ORDER BY orders_count DESC
LIMIT 1;

-- HighEarningRestaurants: top 3 by revenue past year
SELECT r.restaurant_id, r.name,
        SUM(fo.total_amount) AS revenue
FROM Restaurants r
JOIN Food_Orders fo
```

```
  ON fo.restaurant_id = r.restaurant_id
WHERE fo.order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 1 YEAR)
AND fo.order_date <= CURDATE()
GROUP BY r.restaurant_id, r.name
ORDER BY revenue DESC
LIMIT 3;

-- PopularCuisineType: most ordered cuisine past 6 months
SELECT r.cuisine,
        SUM(oi.quantity) AS total_orders
FROM Restaurants r
JOIN Order_Items oi
  ON oi.restaurant_id = r.restaurant_id
JOIN Food_Orders fo
  ON fo.order_id = oi.order_id
WHERE fo.order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH)
AND fo.order_date <= CURDATE()
GROUP BY r.cuisine
ORDER BY total_orders DESC
LIMIT 1;

-- LongestRideRoutes: top 5 by ride_distance
SELECT ride_id, rider_id, ride_distance
FROM Rides
ORDER BY ride_distance DESC
LIMIT 5;

-- DriverRideCounts: # rides per driver past 3 months
SELECT fo.driver_id, u.first_name, u.last_name,
        COUNT(*) AS ride_count
FROM Food_Orders fo
JOIN Drivers d ON d.driver_id = fo.driver_id
JOIN Users u   ON u.user_id   = d.driver_id
WHERE fo.order_date >= DATE_SUB(CURRENT_DATE, INTERVAL 3 MONTH)
AND fo.order_date <= CURDATE()
GROUP BY fo.driver_id, u.first_name, u.last_name;

-- UndeliveredOrders:
SELECT
        order_id,
        delivery_status
FROM Food_Orders AS fo
WHERE fo.delivery_status <> 'delivered';
```

```sql
-- MostCommonPaymentMethods: across rides & food
SELECT payment_method, COUNT(*) AS usage_count
FROM (
  SELECT payment_method FROM Food_Orders
  UNION ALL
  SELECT payment_status AS payment_method FROM Rides
) AS pm
GROUP BY payment_method
ORDER BY usage_count DESC
LIMIT 1;

-- MultiRoleUsers: users who are both drivers & owners
SELECT u.user_id, u.first_name, u.last_name
FROM Users u
WHERE u.user_id IN (SELECT user_id FROM Drivers)
  AND u.user_id IN (SELECT user_id  FROM Restaurant_Owners);

-- DriverVehicleTypes: distribution by type
SELECT vehicle_type, COUNT(*) AS num_drivers
FROM Drivers
GROUP BY vehicle_type;
```