FOOD DELIVERY APP (LIKE SWIGGY/ZOMATO)

Part A: NORMALIZATION PROCESS

➔ **Assessing First Normal Form (1NF):**

My tables are already in 1NF, as all of them have atomic attributes (e.g., phone_number, address, etc.) and no repeating groups.

➔ **Ensuring Second Normal Form (2NF):**

My tables don't have composite primary keys, so 2NF is satisfied, but we need to ensure no partial dependencies in any case:

- **Order_Item Table**: The order_item_id is the primary key, and the columns order_id, menu_item_id, quantity, price, customizations, and special_instructions all depend on the full primary key.
    - If price was a static value that only depends on menu_item_id, we would need to review whether this is redundant. Since price is specific to an order, it is ok for it to be stored here.

➔ **Ensuring Third Normal Form (3NF)**

Some possible transitive dependencies in my tables:

- **Orders Table**: The payment_method attribute is included in the Orders table. This could create a transitive dependency on Payments, where payment_method might be duplicated. We could remove payment_method from Orders and keep it only in the Payments table, ensuring that payment-related data is maintained in one place.
- **Menu Table**: The price and description fields are ok here because they depend on menu_id. There are no transitive dependencies in this table.
- **Delivery_Partners Table**: No issues with transitive dependencies here either, because all attributes depend on the delivery_partner_id.
- **Discounts Table**: This clearly has no transitive dependencies, so it is in 3NF.

➔ **Denormalization (if necessary)**

Denormalization can be used in specific cases to improve performance, especially in heavy applications. We could apply it in the following:

Price in Order_Item: Storing the price in the Order_Item table makes queries faster, but it can cause problems if prices change because we have to update multiple places.

Orders Table: Adding payment_method and status directly to the Orders table makes it quicker to get order details, but it can lead to repeated data that needs to stay updated.

Part C: RATIONALE BEHIND NORMALIZATION DECISIONS

I normalized the tables up to 3NF to reduce data duplication, keep data accurate, and make the database simpler. This helps in ensuring that each record is unique and there are no unnecessary dependencies. For example, the Orders table only links to other tables using foreign keys, and order-related details like total price and delivery address are included. By moving payment details to a separate Payments table, we avoid extra dependencies.

In the Order_Item table, I stored the price of the menu item when the order is placed. This might seem like denormalization, but it helps improve read performance and avoids complex joins when fetching order details, as prices don't usually change after the order.

I decided to store phone numbers as strings (VARCHAR) instead of integers because phone numbers can have special symbols like "+" for international numbers and leading zeros. Using a string makes sure these numbers are stored correctly, even though it takes up a little more space than integers.

I normalized the database up to 3NF, as there were no multi-value dependencies, and going beyond that wouldn't improve the app's performance. To speed up searches, I plan to create indexes on frequently searched columns like user_id, restaurant_id, order_id, and menu_id.

BUS TRACKING APP (LIKE REDBUS)

Part A: Normalization Process

1. **1st Normal Form (1NF):** All the tables in the given schema adhere to 1NF as each table has a primary key, and no repeating groups or arrays are present.

2. **2nd Normal Form (2NF):** To satisfy 2NF, all non-key attributes must depend entirely on the primary key.
   Cases where partial dependencies are eliminated:
   - In the Ticket table, UserID and TripID together uniquely identify a ticket. All other fields in the table, such as SeatNumber, BookingDate, and Fare, are fully dependent on this combination of keys, removing partial dependency.
   - In the TripStop table, TripID and StopOrder together uniquely identify a stop in a trip, ensuring that all attributes depend on this key combination.
3. **3rd Normal Form (3NF):** For 3NF some key adjustments:
   - The Driver table has attributes like Name, PhoneNumber, and LicenseNumber that are directly related to the driver and not to the bus. Hence, this table is normalized by ensuring no transitive dependencies exist.
   - The Trip table contains StartTime, EndTime, and SeatMap. All of these are directly dependent on the TripID and not on each other.
4. **Denormalization:** The maintenance table require partial denormalization to improve query performance.
   - The Maintenance table contains MaintenanceCost which is a derived value. This could be used as part of a query, but it is stored for convenience and to avoid recalculating it on every query.

Part C: RATIONALE BEHIND NORMALIZATION:

In my food delivery app schema, I normalized the tables up to 3NF to reduce redundancy, improve data integrity, and make the database easier to manage. For example, in the Ticket table, I made sure that all attributes like SeatNumber, BookingDate, and Fare depend on the combined primary key (UserID + TripID), ensuring no partial dependencies. Similarly, in the TripStop table, TripID and StopOrder uniquely identify the trip stop, so all fields are dependent on this combination.

In the Driver table, I made sure that attributes like Name and PhoneNumber are only related to the driver and not the bus, eliminating any transitive dependencies. For the Trip table, I kept fields like StartTime, EndTime, and SeatMap linked to the TripID to avoid any unnecessary dependencies.

I used partial denormalization in the Maintenance table by storing the MaintenanceCost, as it's derived from other values. This helps improve query performance, so we don't have to calculate the cost every time.

Regarding phone numbers, I stored them as strings (VARCHAR) instead of integers because phone numbers may contain special characters like "+" and leading zeros, and using strings ensures proper storage. While strings take up more space, it’s worth it to store numbers correctly.