DBMS ASSIGNMENT

Nikki Gautam

Question 2:

Part C:

We expect a write up of at-least ten sentences explaining your rationale behind the process. Why was a particular table normalized? What level of normalization did you use? If something is left denormalized, why was it left that way? Any query that could benefit from it?
If you are storing the phone number as a string instead of a number, what are the benefits and the drawbacks, and which one did you use?

Normalization Choice:

- The database tables were normalized up to **3NF** to ensure data consistency, eliminate redundancy, and simplify updates and deletions
- Each table is designed to store data relevant to a single entity or relationship, minimizing duplicate data storage
- For example, the "**Route**" table was normalized by separating the route details, ensuring that routes are stored independently of trips, making it reusable for multiple buses and trips

Denormalized Tables:

- Certain fields, like "**amenities**" in the "**Bus**" table, were intentionally left denormalized as a single string
- This was done to simplify queries for frequent operations like fetching all amenities for a given bus
- Storing amenities in a normalized format (e.g., a separate "**BusAmenities**" table) would increase join operations for common queries and degrade performance in a real-time scenario

Ticket Table Design:

- The "**Ticket**" table includes the "**fare**" and "**status**" fields to avoid repetitive joins with a separate fare table or booking status table
- Since ticket information is frequently queried (e.g., for booking details, cancellations, or fare breakdowns), keeping it denormalized allows faster access

Phone Number as a String:

Phone numbers were stored as strings (VARCHAR) rather than integers. This decision was made to accommodate:

- **International Numbers:** Including leading zeros and country codes (e.g., "+91")
- **Standardized Format:** Allows storing formatted strings (e.g., "(123) 456-7890") for readability
- **Operations:** Phone numbers are identifiers rather than values for arithmetic operations

However, storing them as strings increases storage size slightly

Check-in Table Normalization:

- The "**CheckIn**" table was normalized to ensure that each check-in entry corresponds to a valid ticket and maintains a one-to-one relationship
- This avoids duplicate check-ins for the same ticket, preserving data integrity

Bus Tracking Table:

- The "**BusTracking**" table is denormalized to store the latest location directly
- This is essential for real-time tracking queries, as normalizing location history into another table would require frequent joins and degrade performance during location lookups

Normalization of Feedback:

- The "**Feedback**" table was normalized to link "**user_id**" and "**trip_id**" directly, ensuring that feedback is tied to a specific user and trip
- This prevents redundancy and enables efficient aggregation of ratings for trips or users

Levels of Normalization:

- All tables except "**Bus**" (amenities) and "**BusTracking**" (current location) are normalized to **3NF**, eliminating partial and transitive dependencies
- This ensures that updates and deletions occur without anomalies

Query Benefits of Denormalization:

Denormalized fields help in implifying commonly used queries

For example:

```
SELECT bus_id, amenities FROM Bus WHERE type = 'AC';
```

This avoids additional joins and accelerates retrieval of buses with specific features