**RIVERS STATE UNIVERSITY**

**DEPARTMENT OF COMPUTER SCIENCE**

**ASSIGNMENT**

**ON**

**DATABASE DESIGN**

**PREPARED BY**

**ADEBAYO MICHAEL ADEWUMI**

**REG NO: PG.2023/02099**

### **1. Introduction**

#### **Overview of the Project**

Database design defines how data will be stored, organised, and managed in a database system. It involves determining the data's structure, relationships, and constraints to ensure efficient retrieval, manipulation, and maintenance. In this study I will be designing a database for a ride-hailing app.

The ride-hailing app, similar to Uber, is designed to connect riders and drivers in real-time. It provides a platform where users (riders) can request rides, and available drivers can accept these requests, facilitating seamless transportation from one location to another. The app involves the management of multiple entities, including riders, drivers, ride requests, routes, payments, and reviews.

#### **1.1 Purpose and Scope of the Database**

The purpose of the database is to manage the vast amount of data generated by the interactions between users (riders and drivers) and the app. This data includes user profiles, ride requests, ride history, vehicle information, payment processing, and location tracking. The database will ensure data integrity, optimise performance for real-time transactions, and maintain scalability as the user base grows.

**The scope of the database includes:**

* Storing user information (riders and drivers)
* Managing ride requests, ride assignments, and tracking rides
* Handling payment processing (fare calculation, transactions)
* Recording user feedback and ratings
* Supporting reporting and analytics for the business

#### **1.2 Problem Statement**

As the number of users and transactions grows, managing real-time ride requests and optimising driver allocation becomes a challenge without a well-designed, scalable database. The system needs to ensure fast matching of riders with drivers, accurate fare calculations, secure payment processing, and provide a user-friendly interface for both riders and drivers.

Challenges include:

* Handling real-time ride-matching
* Storing and retrieving large amounts of data efficiently
* Ensuring data security, particularly for payment and personal information
* Supporting rapid growth in user numbers and geographic expansion

#### **1.3 Objectives**

The objectives of the ride-hailing app database are:

* To design a scalable database that can handle real-time ride requests and efficient driver matching.
* To ensure data integrity and security, especially for user profiles and payment information.
* To optimise query performance for fast ride assignment, fare calculation, and reporting.
* To support future expansion in terms of new features (e.g., vehicle types) and geographic regions.
* To provide insights into user behaviour and performance analytics to improve operational efficiency.

**2.0 Requirements Analysis**

#### **2.1 Data Requirements**

The core entities in the ride-hailing app and their attributes are defined as follows:

* User (Riders, Drivers, and Admin):
  + UserID (Primary Key)
  + Name
  + Email
  + Phone Number
  + User Type (Rider or Driver, Admin)
  + Profile Picture (Optional)
  + Rating (Average rating for drivers/riders)
  + Registration Date
  + Current Location (for real-time location tracking)
* Driver:
  + DriverID (Primary Key, references UserID)
  + Vehicle ID (Foreign Key to Vehicle entity)
  + Driver Licence Number
  + Driving Experience (years)
  + Availability Status (Available/Unavailable)
  + Driver Rating (specific rating from riders)
* Vehicle:
  + VehicleID (Primary Key)
  + DriverID (Foreign Key, linking the vehicle to a driver)
  + Vehicle Type (Car, Bike, etc.)
  + Licence Plate Number
  + Year
  + Capacity (Number of passengers)
* Vehicle Brand:
  + Vehicle\_ Brand\_ ID (Primary Key)
  + VehicleID (Foreign Key, linking the vehicle brand to the vehicle)
  + Name
  + Is\_active (True/ False)
  + Created\_at
  + Updated\_at
* Vehicle Model:
  + Vehicle\_ Model\_ ID (Primary Key)
  + Vehicle\_Brand\_ID (Foreign Key, linking the vehicle model to the vehicle brand)
  + Name
  + Is\_active (True/ False)
  + Created\_at
  + Updated\_at
* Ride Request:
  + RequestID (Primary Key)
  + UserID (Foreign Key, linking the ride request to a rider)
  + Pickup Location (Coordinates or address)
  + Drop-off Location (Coordinates or address)
  + Request Time
  + Ride Status (Requested, Accepted, Completed, Cancelled)
* Ride:
  + RideID (Primary Key)
  + RequestID (Foreign Key, linking to the ride request)
  + DriverID (Foreign Key, linking to the assigned driver)
  + Pickup Time
  + Drop-off Time
  + Distance Covered (calculated in kilometres or miles)
  + Fare Amount
  + Payment Method (Card, Cash, Wallet)
* Payment:
  + PaymentID (Primary Key)
  + RideID (Foreign Key, linking to the completed ride)
  + Amount
  + Payment Method (Credit Card, Wallet, etc.)
  + Payment Status (Paid/Failed)
* Feedback:
  + FeedbackID (Primary Key)
  + RideID (Foreign Key)
  + UserID (Foreign Key, from either Rider or Driver)
  + Rating (1 to 5 stars)
  + Comments

### **3. Conceptual Design**

#### Entity-Relationship Diagram (ERD)

In the ERD, we will represent key entities, their attributes, and the relationships between them. Below is a textual representation of the entities and relationships in the ride-hailing app. An actual ERD diagram will be generated using diagramming Draw.io.

* Entities:
  + User (General for both Rider, Driver, and Admin)
  + Driver (Specialised User with additional attributes)
  + Vehicle (Linked to a Driver)
  + Ride Request (Rider’s request for a ride)
  + Ride (Once a ride is accepted and in progress/completed)
  + Payment (Associated with a Ride)
  + Feedback (Ratings and comments for a completed ride)

##### Relationships:

1. User "places" Ride Request:
   * A User (Rider) can place many Ride Requests.
   * Each Ride Request is associated with exactly one User (Rider).
   * Cardinality: 1 User can have 0 or more Ride Requests (1  
     ).
2. Driver "is assigned" to Ride:
   * A Driver (a specialised User) can accept many Rides.
   * Each Ride is assigned to exactly one Driver.
   * Cardinality: 1 Driver can have 0 or many Rides (1  
     ).
3. Driver "operates" Vehicle:
   * Each Driver operates one Vehicle.
   * Each Vehicle is assigned to exactly one Driver.
   * Cardinality: 1 Driver operates 1 Vehicle, 1 Vehicle is operated by 1 Driver (1:1).
4. Vehicle “belongs” to Vehicle Brand:
   * Each vehicle belongs to particular Vehicle Brand.
   * More than one vehicle can be associated with a particlar brand.
   * Cardinality: 1 Vehicle Brand can have 0 or many Vehicle.
5. Vehicle Brand “associated” to a model:
   * A particular Vehicle Brand can have more models.
   * Cardinality: 1 Vehicle Brand can own 0 or many Models.
6. Ride Request "generates" Ride:
   * A Ride Request may lead to one or more Rides (if the request is accepted).
   * A Ride is linked to exactly one Ride Request.
   * Cardinality: 1 Ride Request may result in 1 Ride, but 1 Ride must be linked to 1 Ride Request (1:1).
7. Ride "generates" Payment:
   * Each Ride must have one Payment after completion.
   * A user can have more than one payment.
   * Each Payment is associated with exactly one Ride.
   * Cardinality: 1 Ride results in 1 Payment (1:1).
8. User "gives" Feedback:
   * Both Riders and Drivers can give Feedback for a completed ride.
   * Each Feedback is linked to one User (either Rider or Driver) and one Ride.
   * Cardinality: 1 User can give many Feedback, and 1 Ride can receive many Feedback from different users (M  
     ).

#### Entity Descriptions and Attributes

* User:
  + UserID: Unique identifier for each user (Primary Key).
  + Name: Full name of the user.
  + Email: Email address.
  + Phone Number: User's phone number.
  + User Type: Indicates whether the user is a "Rider" or a "Driver."
  + Rating: Average rating received by the user.
  + Profile Picture: Optional, picture of the user.
  + Registration Date: Date the user registered on the platform.
* Driver (a specialised User):
  + DriverID: Unique identifier for the driver (Primary Key, Foreign Key from User).
  + VehicleID: Links to the driver’s vehicle.
  + Driver Licence Number: Licence number of the driver.
  + Driving Experience: Number of years of driving experience.
  + Availability Status: Availability for accepting rides (Available/Unavailable).
* Vehicle:
  + VehicleID: Unique identifier for the vehicle (Primary Key).
  + DriverID: Links the vehicle to a specific driver (Foreign Key).
  + Vehicle Type: Type of the vehicle (Car, Bike, etc.).
  + Licence Plate Number: Registration number of the vehicle.
  + Year: The year the vehicle was manufactured.
  + Capacity: Number of passengers the vehicle can accommodate.
* Vehicle Brand:
  + Vehicle\_Brand\_ID: Uniquely identifies the vehicle brand (Primary Key).
  + VehicleID: Links the brand to a specific vehicle (Foreign key).
  + Name: The name of the vehicle brand.
  + Is\_active: This take a boolean value (true/false) showing if the brand is currently in use.
  + Created\_at: Timestamp showing when this vehicle brand was created.
  + Updated\_at: Date and time of last update.
* Vehicle Model:
  + Vehicle\_Model\_ID: Uniquely indentifies the vehicle model ( Primary Key).
  + Vehicle\_Brand\_ID: Links the model to a specific brand (Foreign Key).
  + Name: Name of the vehicle model.
  + Is\_active: This take a boolean value (true/false) showing if the model is currently in use.
  + Created\_at: Timestamp showing when this vehicle model was created.
  + Updated\_at: Date and time of last update.
* Ride Request:
  + RequestID: Unique identifier for each ride request (Primary Key).
  + UserID: Rider’s ID (Foreign Key, from User entity).
  + Pickup Location: The location from which the rider wants to be picked up.
  + Drop-off Location: The destination where the rider wants to go.
  + Request Time: Timestamp when the ride was requested.
  + Ride Status: Current status of the ride (Requested, Accepted, Completed, Cancelled).
* Ride:
  + RideID: Unique identifier for each ride (Primary Key).
  + RequestID: Links to the original ride request (Foreign Key).
  + DriverID: The driver assigned to the ride (Foreign Key).
  + Pickup Time: The time the driver picked up the rider.
  + Drop-off Time: The time the rider was dropped off.
  + Distance Covered: The distance travelled during the ride.
  + Fare Amount: The calculated fare for the ride.
  + Payment Method: Indicates the method used for payment (Card, Cash, etc.).
* Payment:
  + PaymentID: Unique identifier for each payment (Primary Key).
  + RideID: Links to the ride that generated the payment (Foreign Key).
  + Amount: The total amount paid.
  + Payment Method: The method of payment (Card, Wallet, etc.).
  + Payment Status: Indicates whether the payment was successful (Paid/Failed).
* Feedback:
  + FeedbackID: Unique identifier for feedback (Primary Key).
  + RideID: Links the feedback to a specific ride (Foreign Key).
  + UserID: Links the feedback to the user who provided it (Foreign Key).
  + Rating: Rating given (1-5 stars).
  + Comments: Optional comments by the user regarding the ride experience.

#### Relationships and Cardinality

1. User to Ride Request:
   * One User can place many Ride Requests (1).
2. Driver to Ride:
   * One Driver can handle many Rides, but each Ride has only one Driver (1  
     ).
3. Driver to Vehicle:
   * One Driver operates one Vehicle, and one Vehicle is assigned to one Driver (1:1).
4. Ride Request to Ride:
   * One Ride Request generates one Ride, and each Ride is associated with one Ride Request (1:1).
5. Vehicle to
6. Ride to Payment:
   * Each Ride generates one Payment, and each Payment is tied to one Ride (1:1).
7. User to Feedback:
   * One User can give many Feedback, and each Ride can receive multiple Feedback from different users (M).

### **4. Logical Design**

The logical design phase involves converting the conceptual design into a detailed relational schema. This schema defines the tables, their fields, data types, and relationships between them. We’ll also normalise the tables to ensure the database is efficient, avoids redundancy, and maintains data integrity.

#### Relational Schema

The relational schema is an outline of how data will be stored in tables. Below are the main tables derived from the entities in the conceptual design:

* User Table
  + UserID: INT (Primary Key)
  + Name: VARCHAR(100)
  + Email: VARCHAR(100)
  + Phone Number: VARCHAR(15)
  + User Type: ENUM('Rider', 'Driver')
  + Rating: DECIMAL(3, 2)
  + Profile Picture: VARCHAR(255) (Optional)
  + Registration Date: DATETIME
* Driver Table
  + DriverID: INT (Primary Key, Foreign Key from UserID)
  + VehicleID: INT (Foreign Key)
  + Driver Licence Number: VARCHAR(50)
  + Driving Experience: INT (Number of years)
  + Availability Status: ENUM('Available', 'Unavailable')
* Vehicle Table
  + VehicleID: INT (Primary Key)
  + DriverID: INT (Foreign Key)
  + Vehicle Type: ENUM('Car', 'Bike', etc.)
  + Licence Plate Number: VARCHAR(20)
  + Year: INT
  + Capacity: INT (Number of passengers)
* Ride Request Table
  + RequestID: INT (Primary Key)
  + UserID: INT (Foreign Key, from User)
  + Pickup Location: VARCHAR(255)
  + Drop-off Location: VARCHAR(255)
  + Request Time: DATETIME
  + Ride Status: ENUM('Requested', 'Accepted', 'Completed', 'Canceled')
* Ride Table
  + RideID: INT (Primary Key)
  + RequestID: INT (Foreign Key, from Ride Request)
  + DriverID: INT (Foreign Key, from Driver)
  + Pickup Time: DATETIME
  + Drop-off Time: DATETIME
  + Distance Covered: DECIMAL(5, 2) (Distance in km or miles)
  + Fare Amount: DECIMAL(8, 2)
  + Payment Method: ENUM('Card', 'Cash', 'Wallet')
* Payment Table
  + PaymentID: INT (Primary Key)
  + RideID: INT (Foreign Key)
  + Amount: DECIMAL(8, 2)
  + Payment Method: ENUM('Credit Card', 'Wallet', 'Cash')
  + Payment Status: ENUM('Paid', 'Failed')
* Feedback Table
  + FeedbackID: INT (Primary Key)
  + RideID: INT (Foreign Key)
  + UserID: INT (Foreign Key, from User)
  + Rating: INT (1 to 5 stars)
  + Comments: VARCHAR(500) (Optional)

#### Tables, Fields, and Data Types

Let's break down the tables further, including their fields and data types.

1. User Table:
   * Fields: UserID, Name, Email, Phone Number, User Type, Rating, Profile Picture, Registration Date
   * Data Types:
     + UserID: INT (Primary Key, Auto Increment)
     + Name: VARCHAR(100)
     + Email: VARCHAR(100)
     + Phone Number: VARCHAR(15)
     + User Type: ENUM('Rider', 'Driver')
     + Rating: DECIMAL(3, 2)
     + Profile Picture: VARCHAR(255) (Nullable)
     + Registration Date: DATETIME

User Table

| Users | Data Types |
| --- | --- |
| userId(Primary Key) | SERIAL or INT |
| Name | VARCHAR(100) |
| Email | VARCHAR(100) |
| Phone Number | VARCHAR(15) |
| User Type | ENUM(‘Rider’, ‘Driver’) |
| Rating | DECIMAL(3, 2) |
| Profile Picture | VARCHAR(255) (Nullable) |
| Registration Date | DATETIME |

Sample Output:

| **UserID** | **Name** | **Email** | **Phone**  **Number** | **User Type** | **Rating** | **Profile Picture** | **Registration Date** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Lara Jones | lara@gmail.com | 07088945623 | Driver | 4.20 | Null | 2020-11-12  09:57:25 |

1. Driver Table:
   * Fields: DriverID, VehicleID, Driver Licence Number, Driving Experience, Availability Status
   * Data Types:
     + DriverID: INT (Primary Key, Foreign Key from UserID)
     + VehicleID: INT (Foreign Key)
     + Driver Licence Number: VARCHAR(50)
     + Driving Experience: INT
     + Availability Status: ENUM('Available', 'Unavailable')

Driver Table:

| Driver | Data Type |
| --- | --- |
| DriverID | INT(Primary Key, Foreign Key from UserID |
| VehicleID | INT(Foreign Key) |
| Driver Licence Number | VARCHAR(50) |
| Driving Experience | INT |
| Availability Status | ENUM(‘Available’, ‘Unavailability’) |

Sample Output

| **DriverID** | **VehicleID** | **Driver\_License\_Number** | **Driving Experience** | **Availability Status** |
| --- | --- | --- | --- | --- |
| 1 | 1 | ABC100BC124 | 8 | Available |

1. Vehicle Table:
   * Fields: VehicleID, DriverID, Vehicle Type, License Plate Number, Year, Capacity
   * Data Types:
     + VehicleID: INT (Primary Key, Auto Increment)
     + DriverID: INT (Foreign Key from Driver)
     + Vehicle Type: ENUM('Car', 'Bike', etc.)
     + License Plate Number: VARCHAR(20)
     + Year: INT
     + Capacity: INT

Vehicle Table:

| Vehicle | Data Type |
| --- | --- |
| VehicleID | INT (Primary Key, Auto Increment) |
| DriverID | INT (Foreign Key from Driver) |
| Vehicle Type | ENUM(‘Car’, ‘Bike’, etc.) |
| Licence Plate Number | VARCHAR(20) |
| Year | INT |
| Capacity | INT |

Sample Table:

| **VehicleID** | **DriverID** | **Vehicle Type** | **Licence Plate Number** | **Year** | **Capacity** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | Car | APP-406CV | 2015 | 5 |

1. Vehicle Brand:
   * Fields:Vehicle\_Brand\_ID, VehicleID, Name, is\_active, Created\_at, Updated\_at Data Types:
     + Vehicle\_Brand\_ID: INT (Primary Key, Auto Increment)
     + VehicleID: INT (Foreign Key from Vehicle)
     + Name: VARCHAR(25)
     + is\_active:BOOLEAN
     + Created\_at: DATETIME
     + Updated\_at: DATETIME

| **Vehicle Brand** | **Data Type** |
| --- | --- |
| Vehicle\_ Brand\_ID | INT (Primary key, Auto Increment) |
| VehicleID | INT (Foreign Key) |
| Name | VARCHAR(25) |
| is\_active | Boolean |
| Created\_at | DATETIME |
| Updated\_at | DATETIME |

1. Vehicle Model:
   * Fields:Vehicle\_Model\_ID, Vehicle\_Brand\_ID, Name, is\_active, Created\_at, Update\_at Data Types:
     + Vehicle\_Model\_ID: INT (Primary Key, Auto Increment)
     + Vehicle\_Brand\_ID: INT (Foreign Key from Vehicle Brand)
     + Name: VARCHAR(25)
     + is\_active:BOOLEAN
     + Created\_at: DATETIME
     + Updated\_at:DATETIME

| **Vehicle Model** | **Data Type** |
| --- | --- |
| Vehicle\_ Model\_ID | INT (Primary key, Auto Increment) |
| Vehicle\_Brand\_ID | INT (Foreign Key) |
| Name | VARCHAR(25) |
| is\_active | Boolean |
| Created\_at | DATETIME |
| Updated\_at | DATETIME |

1. Ride Request Table:
   * Fields: RequestID, UserID, Pickup Location, Drop-off Location, Request Time, Ride Status
   * Data Types:
     + RequestID: INT (Primary Key, Auto Increment)
     + UserID: INT (Foreign Key from User)
     + Pickup Location: VARCHAR(255)
     + Drop-off Location: VARCHAR(255)
     + Request Time: DATETIME
     + Ride Status: ENUM('Requested', 'Accepted', 'Completed', 'Cancelled')
2. Ride Request Table:

| Ride Request | Data Type |
| --- | --- |
| RequestID | INT (Primary Key, Auto Increment) |
| UserID | INT (Foreign Key from User) |
| Pickup Location | VARCHAR(255) |
| Drop-off Location | DATETIME |
| Ride Status | ENUM(‘Requested’, ‘Accepted’, ‘Completed’, ‘Cancelled’) |

Sample Output:

| RequestID | UserID | Pickup Location | Drop-Off Location | Request Time | Ride Status |
| --- | --- | --- | --- | --- | --- |
| 100 | 1 | Atilary flyover | Mile 1 market | 2024-10-09  12:34:22 | Accepted |

1. Ride Table:
   * Fields: RideID, RequestID, DriverID, Pickup Time, Drop-off Time, Distance Covered, Fare Amount, Payment Method
   * Data Types:
     + RideID: INT (Primary Key, Auto Increment)
     + RequestID: INT (Foreign Key from Ride Request)
     + DriverID: INT (Foreign Key from Driver)
     + Pickup Time: DATETIME
     + Drop-off Time: DATETIME
     + Distance Covered: DECIMAL(5, 2)
     + Fare Amount: DECIMAL(8, 2)
     + Payment Method: ENUM('Card', 'Cash', 'Wallet')

Ride Table:

| Ride | Data Type |
| --- | --- |
| RideID | INT (Primary Key, Auto Increment) |
| RequestedID | INT (Foreign Key from Ride Request) |
| DriverID | INT (Foreign Key from Driver) |
| Pickup Time | DATETIME |
| Drop-off Time | DATETIME |
| Distance Covered | DECIMAL(5, 2) |
| Fare Amount | DECIMAL(8, 2) |
| Payment Method | ENUM(‘Card’, ‘Cash’, ‘Wallet’) |

Sample Output:

| RideID | RequestID | DriverID | Pickup Time | Drop-off Time | Distance Covered | Fare Amount | Payment method |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 100 | 1 | 2024-10-09  12:39:01 | 2024-10-09  12:55:12 | 3 | 5000.00 | Cash |

1. Payment Table:
   * Fields: PaymentID, RideID, Amount, Payment Method, Payment Status
   * Data Types:
     + PaymentID: INT (Primary Key, Auto Increment)
     + RideID: INT (Foreign Key from Ride)
     + Amount: DECIMAL(8, 2)
     + Payment Method: ENUM('Credit Card', 'Wallet', 'Cash')
     + Payment Status: ENUM('Paid', 'Failed')

Payment Table:

| Payment | Data Type |
| --- | --- |
| PaymentID | INT (Primary Key, Auto Increment) |
| RideID | INT (Foreign Key from Ride) |
| Amount | DECIMAL(8, 2) |
| Payment Method | ENUM(‘Credit Card’, ‘Wallet’, ‘Cash’) |
| Payment Status | ENUM(‘Paid’, ‘failed’) |

Sample Output:

| PaymentID | RideID | Amount | Payment Method | Payment Status |
| --- | --- | --- | --- | --- |
| 20 | 5 | 5000.00 | Cash | Paid |

1. Feedback Table:
   * Fields: FeedbackID, RideID, UserID, Rating, Comments
   * Data Types:
     + FeedbackID: INT (Primary Key, Auto Increment)
     + RideID: INT (Foreign Key from Ride)
     + UserID: INT (Foreign Key from User)
     + Rating: INT (1 to 5)
     + Comments: VARCHAR(500) (Nullable)

Feedback table

| Feedback | Data Type |
| --- | --- |
| FeedbackID | INT (Primary key, Auto Increment) |
| RideID | INT (Foreign key from Ride) |
| UserID | INT (Foreign Key from User) |
| Rating | INT(1 to 5) |
| Comments | VARCHAR(500) (Nullable) |

Sample Output:

| FeedbackID | RideID | UserID | Rating | Comments |
| --- | --- | --- | --- | --- |
| 225 | 5 | 1 | 4 | The driver is such a nice lady. |

#### **Normalisation**

##### First Normal Form (1NF):

In 1NF, each table should:

* Contain atomic values (no repeating groups or arrays).
* Each field should have a unique value for each row.

Example: In the User table, each user’s email, name, and phone number are unique, and all attributes are atomic (e.g., only one value per cell). We don’t store multiple phone numbers in a single field.

##### Second Normal Form (2NF):

For 2NF:

* The database must be in 1NF.
* All non-key attributes must be fully dependent on the primary key (no partial dependencies).

Example: In the Driver table, attributes such as Availability Status and Driving Experience depend on the DriverID and not any part of a composite key (since we do not have composite keys in this case). All attributes are dependent on the full primary key, ensuring 2NF.

##### Third Normal Form (3NF):

For 3NF:

* The database must be in 2NF.
* There should be no transitive dependencies (non-key attributes should not depend on other non-key attributes).

Example: In the Ride table, all non-key attributes like Pickup Time, Drop-off Time, and Fare Amount are dependent on the RideID (the primary key), and not on each other. The same is true for the Payment table where Amount and Payment Method depend directly on PaymentID.

With the tables in 3NF, the database is optimised to avoid redundancy, ensure efficient query performance, and maintain data integrity.

**ER DIAGRAM FOR RIDE-HAILING APP DATABASE DESIGN**

