**UNIVERSITY OF BUEA**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING (FET)**



**CEF 440: INTERNET PROGRAMMING (J2EE) AND MOBILE PROGRAMMING**

**TASK 6: DATABASE DESIGN AND IMPLEMENTATION**

**26-05-2023**

**NAMES OF GROUP MEMBERS**

| 1. AMBE MBONG-NWI NCHANG | FE20A007 |
| --- | --- |
| 1. ASONGNA FRANK TONGWA | FE20A012 |
| 1. BESINGI NAURA MABOLA | FE20A023 |
| 1. FONGE BERTIN AMIN-SHU | FE20A040 |
| 1. NKWETAKEM TABO BRUNO | FE20A091 |

**Course Instructor**

DR. VALERY NKEMENI

**Table of Content**

1. Introduction

1.1 Brief overview of CarryamGo mobile application

1.2 Importance of an efficient and well-designed database

1.3 Purpose of this report

1.4 Functional Requirements of our Database.

1.5 Non-functional Requirements of our Database.

1. Entity Relationship Diagram

2.1 Definition

2.2 ER Diagram for our System

2.3 Some abbreviations and their meanings

1. Logical Schema

3.1 Definition

3.2 Logical Schema for our System

1. Database Implementation

4.1 Database Creation

4.2 Table Creation

4.3 Table Definitions

4.4 Inserting Data

4.5 Measures taken to ensure data security and

Integrity Measures taken to ensure data security

and integrity

1. Conclusion

1. **Introduction**

**1.1 Brief overview of CarryamGo mobile application**

CarryamGo is a market management where customers can review prices of products and their location, whereas sellers can display these products thereby providing enough exposure for buyers to help them increase revenue and also helps sellers get the best available goods at good prices and with minimal waste of time. After all, time is money.

**1.2 Importance of an efficient and well-designed database**

* **Data Integrity:** A well-designed database ensures that data is accurate and consistent. This is important because decisions made based on inaccurate or inconsistent data can lead to costly mistakes and poor business outcomes.
* **Performance:** An efficient database design ensures that data can be accessed and processed quickly. This is important because slow performance can lead to poor user experience, lost productivity, and missed opportunities.
* **Scalability:** A well-designed database can easily scale to handle increasing amounts of data and users. This is important because as market participants grow, so does its data needs.
* **Security:** A well-designed database includes appropriate security measures to protect against unauthorized access, data breaches, and other security threats.
* **Cost**: A well-designed database can help reduce costs associated with data storage, processing, and maintenance. This is important because data-related expenses can quickly add up, especially as market participants grow.

**1.3 Purpose of this report**

**Purpose:**

The purpose of this report is to provide a detailed account of how the database was created, why certain decisions were made, and how the database is expected to function.

**1.4 Functional Requirements of our database.**

* **Data storage:** The database is able to store and organize data in a way that is efficient, secure, and scalable.
* **Data retrieval:** The database allows users to retrieve data quickly and easily, using search and filtering functions.
* **Data entry:** The database provides a user-friendly interface for entering new data, with validation and error-checking mechanisms to ensure accuracy.
* **Data manipulation:** The database allows users to manipulate data, such as sorting, grouping, and aggregating data, to support analysis and reporting.
* **User management:** The database includes user management features, such as user authentication, access controls, and user roles, to ensure data security and privacy.
* **Integration:** The database is able to integrate with other software and systems, such as data warehouses, analytics tools, and APIs, to support data exchange and interoperability.
* **Performance:** The database is able to handle large volumes of data and users, with fast response times and minimal downtime.

**1.5 Non-functional Requirements of our Database.**

* **Performance:** The database is able to handle a high volume of transactions, with fast response times and minimal downtime.
* **Scalability:** The database is scalable to accommodate future growth in data volume and user traffic.
* **Security:** The database is secure, with strong authentication, access controls, and encryption mechanisms to protect sensitive data
* **Usability:** The database is easy to use and navigate, with a user-friendly interface that supports efficient data entry and retrieval.
* **Compatibility:** The database is compatible with other software and systems, including data warehouses, analytics tools, and APIs.

1. **Entity-Relationship Diagram**

**2.1 Definition:**

An Entity-Relationship (ER) diagram is a graphical representation of the entities, attributes and relationships among data objects in a database.

**2.2 ER Diagram for our System**

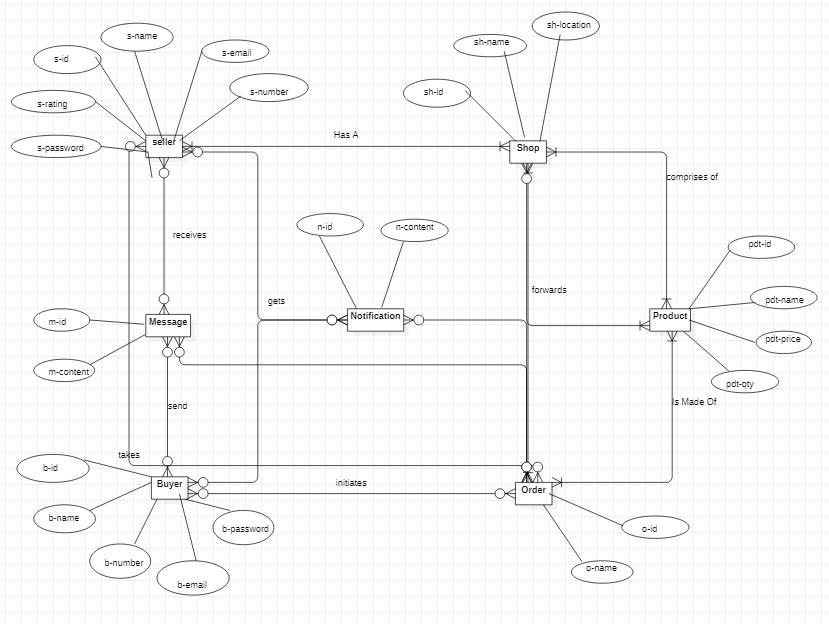


Fig 1: ER Diagram

There are 7 entities and 8 relations in this system with the entities being Buyer, Seller, Shop, Product, Order, Message and Notification.

* Buyer is a person who purchases products from a shop.
* Seller is a person that sells products to buyers.
* Shop is a physical location indicated online in the app where buyers can order products from sellers.
* Product is an item that is available for purchase from a shop.
* Order is a record of a buyer's purchase of a product from a shop.
* Message is a communication between a buyer and seller.
* Notification is a message that is sent to a buyer or seller about an order or other event.

The relationships between these entities are as follows:

* Buyers can initiate orders with multiple shops.
* Sellers can sell products to multiple buyers through their various shops..
* Products can be sold by multiple shops.
* Orders can contain multiple products.
* Messages can be sent from buyers to sellers, or from sellers to buyers.
* Notifications can be sent to buyers or sellers about an order, or about other events such as a product being out of stock.

**2.3 Some abbreviations and their meanings**

| **Abbreviation** | **Meaning** |
| --- | --- |
| s-id | Seller ID |
| m-id | Message ID |
| b-id | Buyer ID |
| o-id | Order ID |
| n-id | Notification ID |
| sh-id | Shop ID |
| pdt-id | Product ID |

1. **Logical Schema**

**3.1 Definition:**

It defines the logical structure of the database, including the entities, attributes, relationships, and constraints that define the data model.

**3.2 Logical Schema for our System**

* Seller(s-id, s-name, s-email, s-number, s-password, s-rating)
* Message(m-d, m-text)
* Receives(\*s-id, \*m-id)
* Buyer(b-id, b-name, b-number, b-email, b-password)
* Send(\*b-id, \*m-id)
* Order(o-id, o-name)
* Initiates(\*b-id, \*o-id)
* Takes(\*s-id, \*o-id)
* Notification(n-id, n-content)
* Shop(sh-id, sh-name, sh-location)
* Has A(\*s-id, \*sh-id)
* Products(pdt-id, pdt-name, pdt-price, pdt-qty)
* Comprises Of(\*sh-id, \*pdt-id)
* Is Made Of(\*o-id, \*pdt-id)
* Gets(\*b-id, \*s-id, \*n-id)
* Forwards(\*m-id, \*n-id, \*sh-id, \*o-id, \*pdt-id)

1. **Database Implementation**

The provided database code represents the schema definition and table creation statements for a database called "Carayamgo" in PostgreSQL. Let's go through each section of the code to understand it in detail:

**4.1 Database Creation:**

* The code begins with the creation of the database using the statement `CREATE DATABASE Carayamgo;`.
* This statement creates a new database named "Carayamgo".

**4.2 Table Creation:**

* The code then proceeds to create multiple tables within the "Carayamgo" database.
* Each table is created using the `CREATE TABLE` statement, followed by the table name and column definitions.

**4.3 Table Definitions:**

**Seller Table:**

* The `Seller` table represents sellers in the system.
* It includes columns such as `seller\_id`, `seller\_name`, `seller\_email`, `seller\_number`, `seller\_address`, `seller\_password`, `seller\_rating`, and `created\_at`.

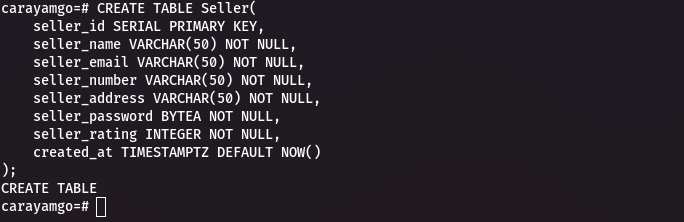


Fig 2: Creating the Seller Table

**Buyer Table:**

* The `Buyer` table represents buyers in the system.
* It includes columns such as `buyer\_id`, `buyer\_name`, `buyer\_number`, `buyer\_email`, `buyer\_address`, `buyer\_password`, `buyer\_rating`, and `created\_at`.

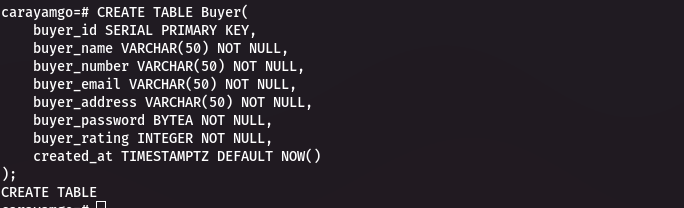


Fig 3: Creating the Buyer Table

**Product Table:**

* The `Product` table represents products offered by sellers.
* It includes columns such as `product\_id`, `product\_name`, `product\_price`, `product\_quantity`, `seller\_id`, and `created\_at`.
* The `seller\_id` column references the `seller\_id` column in the `Seller` table.

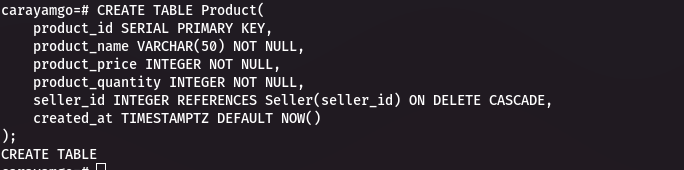


Fig 4: Creating the Product table

**Message Table:**

* The `Message` table represents messages sent between sellers and buyers.
* It includes columns such as `message\_id`, `message\_text`, `sender\_id`, `receiver\_id`, and `sent\_at`.
* The `sender\_id` column references the `seller\_id` column in the `Seller` table, and the `receiver\_id` column references the `buyer\_id` column in the `Buyer` table.

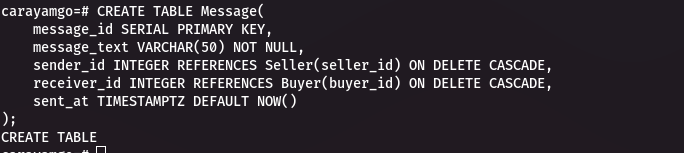


Fig 5: Creating the message table

**Shop Table:**

* The `Shop` table represents shops associated with sellers and products.
* It includes columns such as `shop\_id`, `shop\_name`, `shop\_location`, `product\_id`, `seller\_id`, and `created\_at`.
* The `product\_id` column references the `product\_id` column in the `Product` table, and the `seller\_id` column references the `seller\_id` column in the `Seller` table.

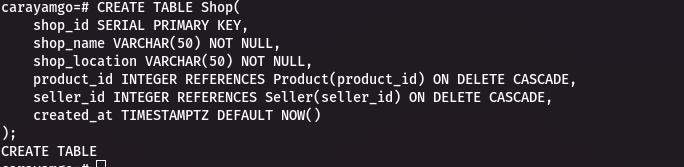


Fig 6: Creating the Shop Table

**Orders Table:**

* The `Orders` table represents orders placed by buyers for products.
* It includes columns such as `orders\_id`, `orders\_name`, `orders\_amount`, `product\_id`, `seller\_id`, `buyer\_id`, and `created\_at`.
* The `product\_id` column references the `product\_id` column in the `Product` table, the `seller\_id` column references the `seller\_id` column in the `Seller` table, and the `buyer\_id` column references the `buyer\_id` column in the `Buyer` table.

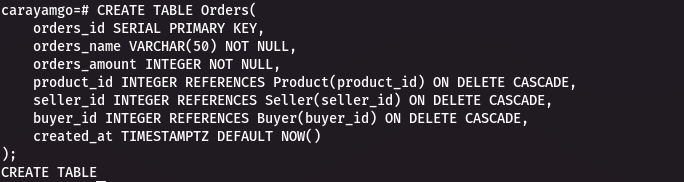


Fig 7: Creating the Orders Table

**Notification Table:**

* The `Notification` table represents notifications sent to recipients.
* It includes columns such as `notification\_id`, `notification\_content`, `recipient\_id`, `recipient\_role`, and `created\_at`.
* The `recipient\_id` column is a reference to the ID of the recipient (seller or buyer), and the `recipient\_role` represents the role of the recipient.
* The foreign key constraint for `recipient\_id` references the `seller\_id` column in the `Seller` table and the `buyer\_id` column in the `Buyer` table.

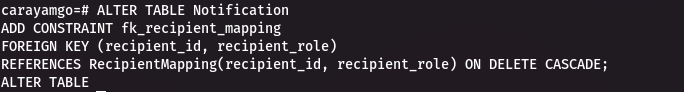


Fig 8: Creating the Notification Table

**RecipientMapping Table:**

* The `RecipientMapping` table is a separate table used for mapping the recipient ID and role.
* It includes columns `recipient\_id` and `recipient\_role`.
* The primary key of this table is defined on `recipient\_id` and `recipient\_role`.
* The foreign key constraints within the `RecipientMapping` table reference the `seller\_id` column in the `Seller` table and the `buyer\_id` column in the `Buyer` table.

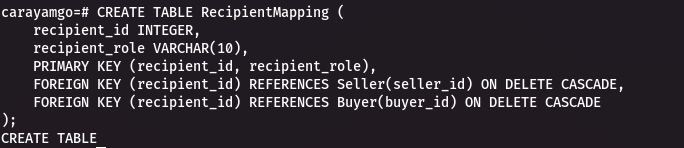


Fig 9: Creating the RecipientMapping Table

**Admin Table:**

* The `Admin` table represents administrators in the system.
* It includes columns such as `admin\_id`, `admin\_name`, `admin\_email`, `admin\_password`, and `created\_at`.

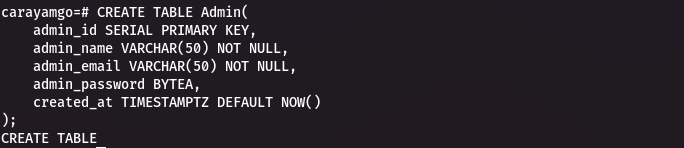


Fig 10: Creating the Admin Table

**4. Constraints and Relationships:**

* The tables include various constraints and relationships defined through foreign key references. These constraints ensure data integrity and enforce referential integrity between related tables.
* The `ON DELETE CASCADE` option specifies that when a referenced row is deleted, the corresponding rows in the referencing table will also be deleted.

This database schema design aims to provide a structure for the "Carayamgo" application, encompassing entities such as sellers, buyers, products, shops, messages, orders, notifications, and administrators. The relationships between tables help establish connections and maintain data consistency throughout the system.

**4.4 Inserting Data:**

The provided code snippet demonstrates the insertion of data into various tables in the Carayamgo database. Let's go through each step:

**1. Inserting data into the Seller table:**

In this step, data is inserted into the `Seller` table. Each row represents a seller and contains attributes such as `seller\_name`, `seller\_email`, `seller\_number`, `seller\_address`, `seller\_password`, and `seller\_rating`. The `seller\_password` attribute stores the hashed password using the **MD5 algorithm**. The `md5` function is used to calculate the hash, and the resulting hash value is converted to the `bytea` type and concatenated with the prefix `\\x`.

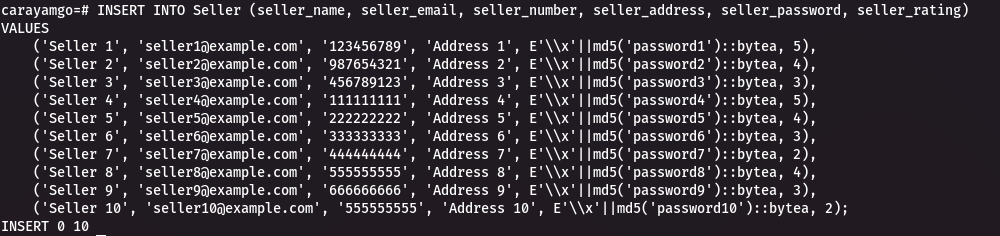


Fig 11: Inserting Data into the Seller Table

**2. Inserting data into the Buyer table:**

Similar to the previous step, data is inserted into the `Buyer` table. Each row represents a buyer and contains attributes such as `buyer\_name`, `buyer\_number`, `buyer\_email`, `buyer\_address`, `buyer\_password`, and `buyer\_rating`. The `buyer\_password` attribute stores the hashed password using the same approach as in the Seller table.

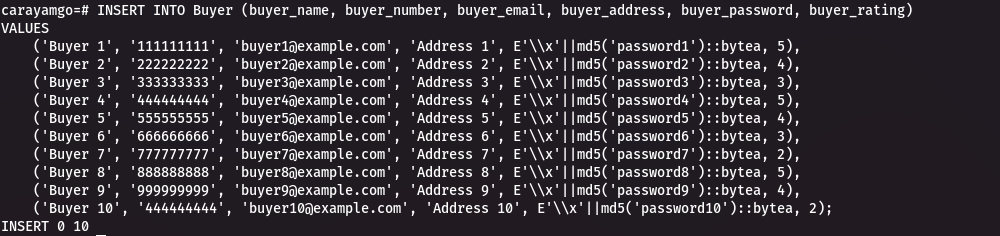


Fig 12: Inserting Data into the Buyer Table

**3. Inserting data into the Product table:**

This step involves inserting data into the `Product` table, which represents various products available in the system. Each row represents a product and includes attributes such as `product\_name`, `product\_price`, `product\_quantity`, `seller\_id`. The `seller\_id` attribute is a foreign key referencing the corresponding seller who listed the product.

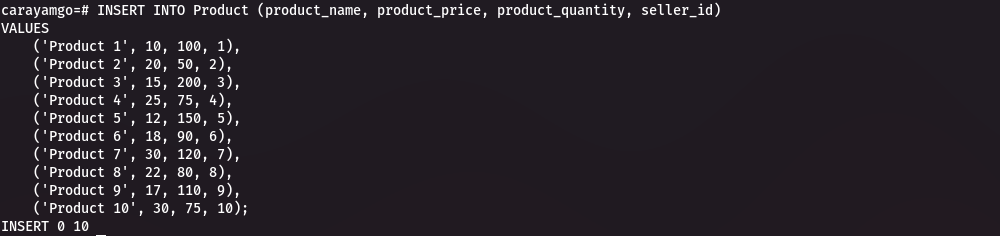


Fig 13: Inserting Data into the Product Table

**4. Inserting data into the Message table:**

The `Message` table is used to store messages exchanged between users. Each row represents a message and contains attributes such as `message\_text`, `sender\_id`, and `receiver\_id`, representing the message content and the sender and receiver of the message, respectively.

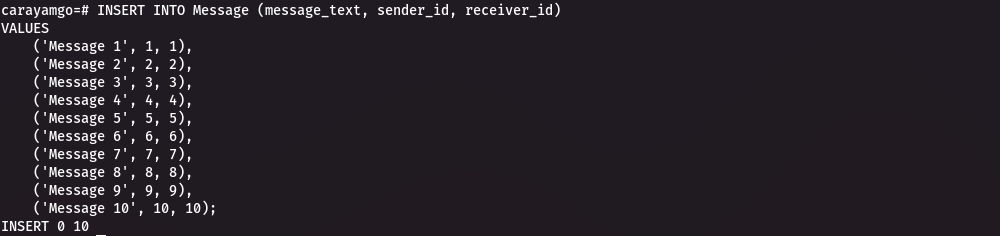


Fig 14: Inserting Data into the Message Table

**5. Inserting data into the Shop table:**

In this step, data is inserted into the `Shop` table, which represents various shops associated with sellers. Each row includes attributes such as `shop\_name`, `shop\_location`, `product\_id`, and `seller\_id`. The `product\_id` and `seller\_id` attributes are foreign keys referencing the respective product and seller.

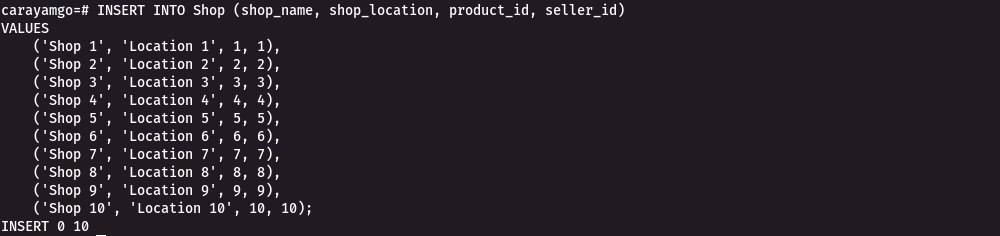


Fig 15: Inserting Data into the Shop Table

**6. Inserting data into the Orders table:**

The `Orders` table is used to store information about orders placed by buyers. Each row represents an order and includes attributes such as `orders\_name`, `orders\_amount`, `product\_id`, `seller\_id`, and `buyer\_id`. The `product\_id`, `seller\_id`, and `buyer\_id` attributes are foreign keys referencing the respective product, seller, and buyer.

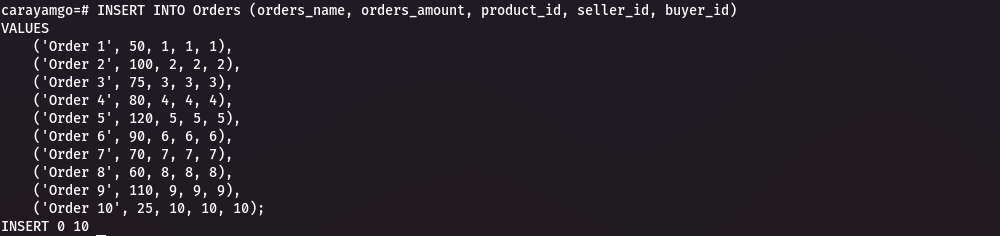


Fig 16: Inserting Data into the Orders Table

**7. Inserting data into the Notification table:**

This step involves inserting data into the `Notification` table, which stores notifications sent to users. Each row represents a notification and includes attributes such as `notification\_content`, `recipient\_id`, and `recipient\_role`. The `recipient\_id` attribute refers to the user receiving the notification, and the `recipient\_role` indicates the role of the recipient (seller or buyer).

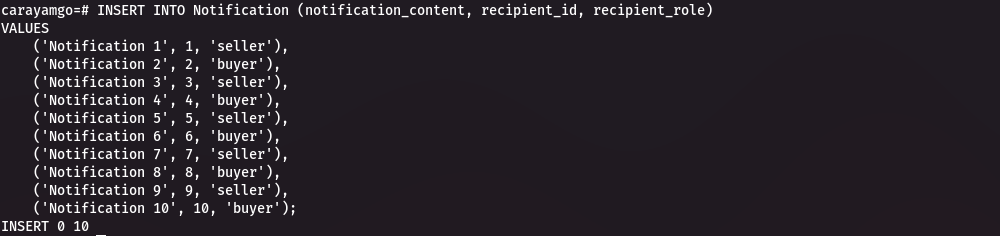


Fig 17: Inserting Data into the Notification Table

**8. Inserting data into the RecipientMapping table:**

The `RecipientMapping` table is used to map recipients to their roles. Each row includes attributes such as `recipient\_id` and `recipient\_role`, indicating the recipient's ID and their corresponding role (seller or buyer).

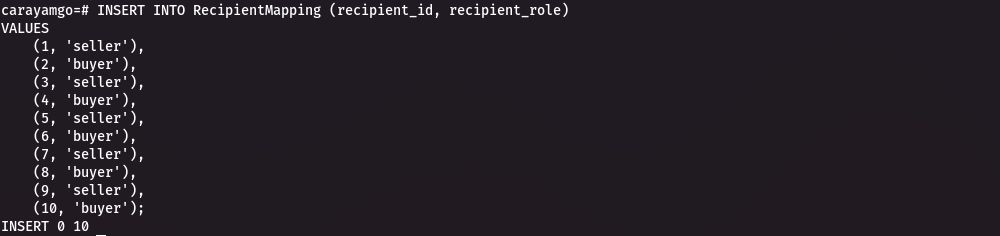


Fig 18: Inserting Data into the RecipientMapping Table

**9. Inserting data into the Admin table:**

In this step, data is inserted into the `Admin` table, which represents administrators of the system. Each row includes attributes such as `admin\_name`, `admin\_email`, and `admin\_password`. The `admin\_password` attribute stores the hashed password using the same approach as in the Seller and Buyer tables.

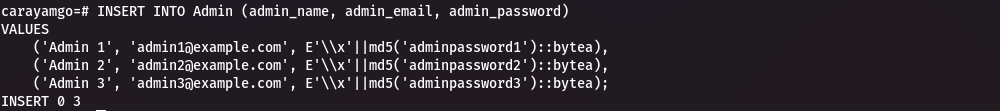


Fig 19: Inserting Data into the Admin Table

**4.5 Measures taken to ensure data security and integrityMeasures taken to ensure data security and integrity:**

Data security and integrity are crucial aspects of any mobile application, including Carryamgo. Below are some of the measures that we took to ensure data security and integrity in our Carryamgo mobile application:

1. **Secure Authentication**: Implement a robust authentication system to ensure that only authorized users can access the application. This includes secure password storage using strong hashing algorithms like md5.The overall purpose of hashing passwords and storing them in a hashed format is to enhance security. Hashing is a one-way process, making it difficult to retrieve the original password from the stored hash. This helps protect user passwords in case of a data breach where the database is compromised.

The provided code demonstrates the usage of the MD5 hashing algorithm in PostgreSQL to generate the hash values for passwords. The expression `E'\\x'||md5('password')::bytea` calculates the MD5 hash of the password string, converts it to the `bytea` data type, and concatenates it with the prefix `\\x`. The resulting value is then stored in the respective password attribute in the tables.

By employing this approach, the system ensures that user passwords are not stored in plaintext, adding an extra layer of security to safeguard user accounts.

1. **Data Minimization:** Collect and store only the data that is necessary for the application's functionality. Avoid storing sensitive user information that is not required, and regularly review and purge unnecessary data to minimize potential risks.

**5. Conclusion**

In conclusion, the security and integrity of user data in the Carryamgo mobile application are paramount. Our application implements a comprehensive set of security measures, including secure authentication, encryption, data minimization, and communication, to safeguard user information, prevent unauthorized access, and mitigate data breaches. Regular security audits and updates identify and address vulnerabilities promptly. User input validation, secure password storage, and session management further enhance the application's security. Carryamgo complies with data protection regulations, respects user privacy, and obtains proper consent, demonstrating its commitment to user trust and legal requirements. By prioritizing data security, Carryamgo provides users with a safe environment for transactions and remains dedicated to maintaining security best practices and addressing emerging threats.