Food Delivery Platform Database

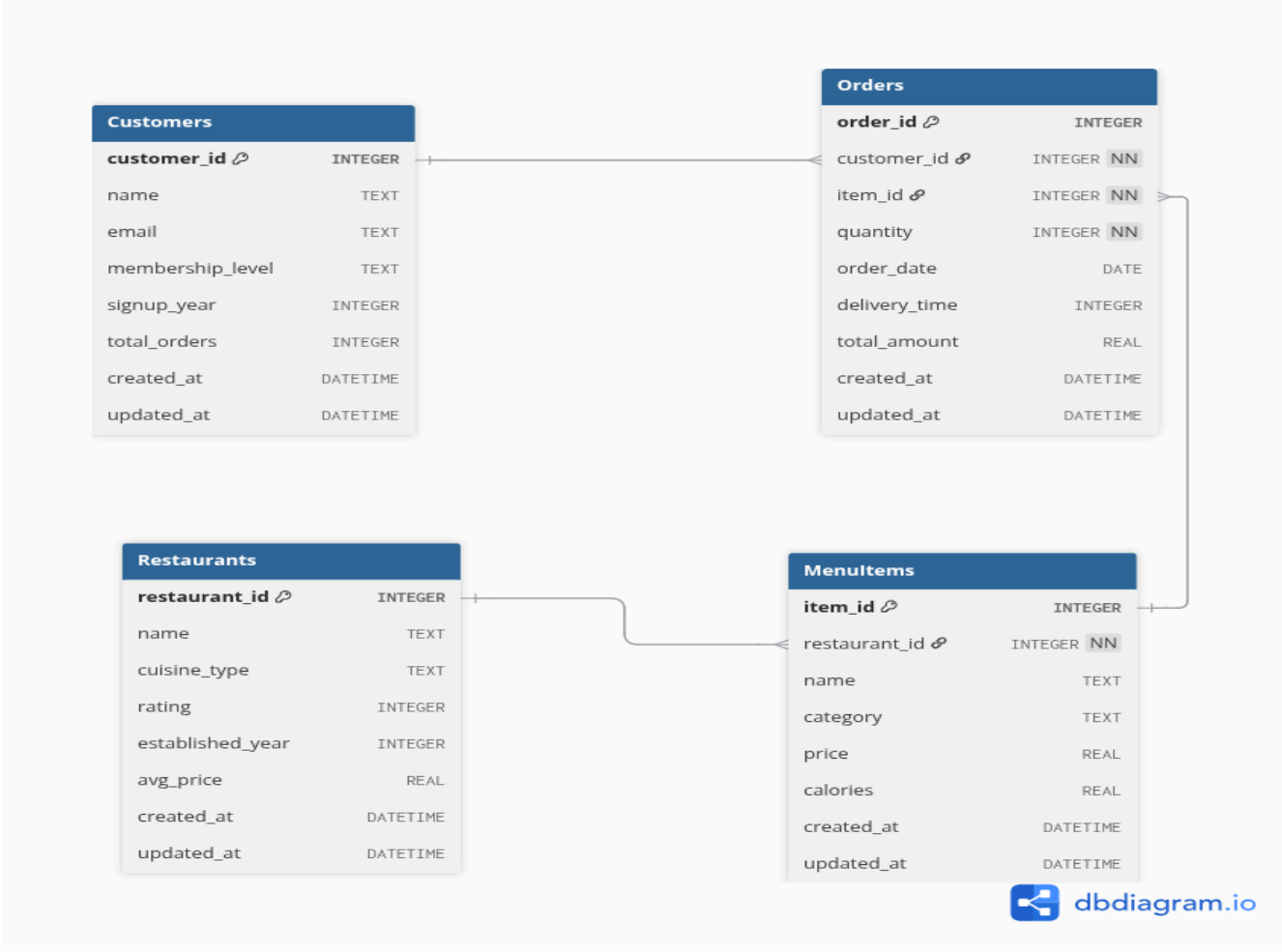
ABOSEDE RACHEAL ADEBISI
22101584

1. Introduction and Project Goal

This project created a database for a pretend **Food Delivery Platform** using **SQLite**. The main goal was to build a realistic, working database that follows good design rules. This report explains how the database was put together, why it was designed this way, and discusses important ethical points about the data.

2. Database Structure (Schema)

The database is built from four separate tables that are linked together.



2.1. Justification for Separate Tables

Four tables were used instead of one giant table for a simple reason: **efficiency and accuracy**. This concept is called **Normalization**.

- **Avoids Repeating Data:** Imagine having one table. Every time a customer orders, their name, address, and email have to be typed again. With separate tables, the customer's details are stored once in the `Customers` table. The `Orders` table just uses a small number, the `customer_id`, to point back to the right customer. This saves space and prevents mistakes.
- **Keeps Data Consistent:** If a restaurant changes its address, it only needs to be updated in one place (the `Restaurants` table). If the data was repeated across thousands of order records, we might miss one, leading to wrong information.
- **Table Relationships:** Using these links (called **Foreign Keys**) between tables was a specific requirement to show a deeper understanding of database design.

2.2. Data Constraints

Constraints are rules built into the database to keep the data clean and reliable.

Constraint	Table & Attribute	Purpose
Primary Key (PK)	`customer_id`, `restaurant_id`, `item_id`, `order_id`	Ensures every record has a unique ID. This is the main way to find a specific piece of data.
Foreign Key (FK)	`customer_id` in `Orders`, `restaurant_id` in `MenuItems` and `Orders` tables	Enforces the links between tables. It makes sure you can't create an order for a customer or restaurant that doesn't exist.
UNIQUE	`email` in `Customers` table	Guarantees that no two customers can sign up with the same email address.
NOT NULL	Various columns (e.g., `name`, `price`)	Ensures that important fields are never left empty.

2.3. Missing Data Justification

Missing values were deliberately introduced in the range of 2% to 10% to simulate realistic operational scenarios

1. Missing customer emails due to incomplete registration
2. Missing restaurant ratings for newly onboarded restaurants
3. Missing menu item calories due to restaurants not providing nutritional labels
4. Missing delivery_time due to system failures

2.4 Duplicate Data Justification

Duplicate entries (1% to 3%) were introduced to reflect real-world issues

1. Customers with the same name
2. Restaurant franchises sharing identical names
3. Duplicate orders caused by accidental multiple submissions

3. How the Data Was Generated

The database needed to contain at least 1,000 realistic records. Since this is a project and not a real company, a special method was used to create fake, but believable, data.

3.1. Tools Used

1. **`schema.sql`:** This file contained the SQL commands (the database language) to build the empty tables and set up all the constraints mentioned above.
2. **`generate_data.py`:** This Python script was the engine for creating the data.
3. **Faker Library:** This is a powerful tool used inside the Python script. It generates realistic fake information, such as names, addresses, email addresses, and phone numbers.

3.2. The Generation Process

The Python script followed these steps:

1. **Setup:** It connected to the empty SQLite database file (**`data.db`**).
2. **Generating Core Data:** It used Faker to create a set of fake customers and restaurants. For example, it would generate a random name, a random email (guaranteed to be unique by the **`UNIQUE`** constraint), and a random address for each customer.
3. **Generating Menu Items:** It created menu items for each restaurant, ensuring that the **`price`** was a realistic number and the **`category`** (like 'Dessert' or 'Main' course) was chosen from a set list.

4. **Generating Orders (The Bulk Data):** This was the most important step, as it created over 1,000 records. For each order, the script randomly picked an existing `customer_id` and `restaurant_id` to ensure the Foreign Key links were valid. It also calculated a random `total_amount` and assigned a random `delivery_time` (in minutes). This process ensured the database was large and the data was randomized across all appropriate types (nominal, ordinal, interval, ratio).

4. Ethical and Data Privacy Discussion

Even though this is a project with fake data, it is important to think about the real-world implications of running a food delivery database.

4.1. Data Minimization

In a real platform, only the data that is absolutely necessary should be collected. For example, a customer's address needs to be collected because it is needed for delivering the food. Also collecting their email for communication and notification. Collecting extra and unnecessary data increases the risk if the database is ever hacked.

4.2. Anonymization and Pseudonymization

Since the project uses **synthetic (fake) data**, it has already achieved a high level of **anonymization**. The names, emails, and addresses are not real people's information.
In a real company, if data is used for analysis (like finding out which cuisine is most popular), the best practice is to **pseudonymize** the data. This means replacing the real customer names and emails with a fake ID before giving the data to analysts. This protects the customer's identity while still allowing the company to study trends.

4.3. Data Security

The database contains sensitive information, such as customer addresses and order history. In a real system, the following security measures would be mandatory:

- **Encryption:** All sensitive data (especially passwords and payment information, if stored) must be scrambled (encrypted) so that if a hacker steals the database file, they cannot read the information.
- **Access Control:** Only specific employees should be allowed to view certain parts of the data. For example, a delivery driver only needs the customer's address, not their entire order history.

By considering these ethical and privacy points, which ensures that the database design is not only technically sound but also responsible and compliant with privacy standards.

5. Conclusion

The Food Delivery Platform database successfully models a real-world system using a well-normalized, multi-table structure. The use of Python and the Faker library allowed for the efficient creation of a large, realistic dataset that adheres to all data type requirements. The design choices, from table separation to constraint implementation, were made to ensure data integrity and efficiency, while the ethical discussion highlights the importance of privacy in any data-driven application.