***Report on the Optimization of Delivery Services: Design and Implementation of a Robust Restaurant Delivery System Database***

***Zyad Fri, Jaafar Yeffou and Oumaima kouhail***

***General Introduction:***

The rapid evolution of the food delivery industry has necessitated the development of efficient and robust systems to meet the increasing demands of customers. This project focuses on the design and implementation of a sophisticated Restaurant Delivery System Database aimed at optimizing delivery services. The database is meticulously crafted to support seamless operations, ensuring timely and accurate order fulfillment while providing a foundation for effective management and decision making.

***Database Schema Overview:***

The Restaurant **Delivery** System Database features a meticulously designed schema that encapsulates the intricacies of modern food delivery services. The cornerstone of this schema is the Delivery entity, capturing essential details such as delivery ID, date, time, and delivery status. It establishes a crucial link to the **Vehicle** entity through a foreign key reference, ensuring seamless coordination with the delivery fleet. Complementing this, the **Orders** entity manages individual customer orders, incorporating attributes like order ID, customer ID, order city, and establishing a foreign key relationship with the Customer entity.

The complex many-to-many relationship between deliveries and orders is elegantly handled by the **ComposedBy** entity, facilitating accurate association and tracking. Our **Customer** entity is thoughtfully designed to store comprehensive user information, including name, address, contact details, and secure login credentials. **Payment** transactions are efficiently recorded in the **Pays** entity, which links both the broader payment system and specific orders. Meanwhile, the Payment entity manages payment details such as payment ID and date.

The **Contains** entity is pivotal in tracking the composition of orders, detailing the inclusion of **meals** and establishing foreign key connections with both Orders and **Meal** entities. Individual **restaurants** are represented in the Restaurants entity, capturing crucial information like name and location. The **Meal** entity encompasses a variety of food items available, linking back to the corresponding restaurant. Two specialized entities, **Payment\_Cash** and **PaymentOnline**, cater to different transaction modes—cash and online payments, respectively. These entities manage financial details, including credit card information and transaction timing.

Ensuring the smooth functioning of the delivery system, the **Delivery\_Personal** entity stores vital information about delivery personnel, including ID, name, city, and phone number. It establishes a foreign key relationship with the Vehicle entity, creating a seamless connection between delivery personnel and assigned vehicles. Finally, the Vehicle entity itself encompasses critical attributes such as vehicle ID, maximum meal capacity, speed, and operational status.

***Comprehensive Summary of Project Tasks and Achievements***

***Task 1: Functional Dependencies and Minimal Cover***

**Delivery Personnel Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency 1: {Pid} → {Name, DpCity, DP\_PhoneNumber, Is\_fired}

Functional Dependency 2: {Vid} → {Pid}

* **Minimal cover**

{Pid} → {PName, DpCity, DP\_Phone\_Number, Is\_fired}

{Vid} → {Pid}

**Vehicle Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency: {Vid} → {Max\_ Meals, Speed, Is\_working}

* **Minimal cover**

{Vid} → {Max\_Meals, Speed, Is\_working}

**Delivery Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency: {Did} → {Ddate, Vid, Dtime, Is\_delivered}

* **Minimal cover**

{Did} → {Ddate, Vid, Dtime, Is\_delivered}

**orders Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency: {Order\_id}→{Cid, Order\_city }

* **Minimal cover**

{Order\_id}→{Cid, Order\_city }

**Meal Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency: {Rid, Mname}→{Price}

* **Minimal cover**

{Rid, Mname}→{Price}

**Restaurant Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency: {Rid} → {Rname, RCity}

* **Minimal cover**

{Rid} → {Rname, RCity}

**Customer** **Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency 1: {Cid} → {Address, Cname, Email, Phone\_Number, Pass\_word}

Functional Dependency 2: {Email} → {Cid}

* **Minimal cover**

{Cid} → {Address, Cname, Email, Phone\_Number, Pass\_word}

{Email} → {Cid}

**Payment**  **Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency : {PyID} → {PDate}

* **Minimal cover**

{PyID} → {PDate}

**Payment\_Cash**  **Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency: {PCid} → {Budget, PCPrice, PID, PYid}

* **Minimal cover**

{PCid} → {Budget, PCPrice, PID, PYid}

**Payment\_Online**  **Entity:**

* **Functional Dependencies (FDS):**

Functional Dependency: {PoID} → {POPrice, Credit\_Card, Ptime, PYid}

* **Minimal cover**

{PoID} → {POPrice, Credit\_Card, Ptime, PYid}

**Composed\_by** **Relationship:**

* **Functional Dependencies (FDS):**

Functional Dependency: {Did, Order\_id} → {Did, Order\_id}

* **Minimal Cover:**

The "ComposedBy" relationship has only one trivial functional dependency, making the minimal cover effectively empty.

**Contains Relationship:**

* **Functional Dependencies (FDS):**

Functional Dependency: {Order\_id, {Rid, Mname}} → {Order\_id, Rid, MName}

* **Minimal Cover:**

The "Contains" relationship has only one trivial functional dependency, making the minimal cover effectively empty.

**Pays Relationship:**

* **Functional Dependencies (FDS):**

Functional Dependency: {PyId, Order\_id} → {PyId, Order\_id}

* **Minimal Cover:**

The "Pays" relationship has only one trivial functional dependency, making the minimal cover effectively empty.

***Summary:***

In the first task, the focus was on identifying and formalizing the functional dependencies within the entities of the Restaurant Delivery System Database. The minimal cover, a set of minimal functional dependencies, was established for key attributes in each entity, ensuring a comprehensive and non-redundant representation of the data. For the "Delivery Personnel" entity, dependencies such as {Pid} → {Name, DpCity, DPPhoneNumber, Is\_fired} and {Vid} → {Pid} were identified and deemed minimal. Similar processes were applied to entities like "Vehicle," "Delivery," "Orders," "Meal," "Restaurant," "Customer," "Payment," "Payment\_Cash," and "Payment\_Online," resulting in minimal covers that guarantee optimal normalization and maintain the integrity of the database.

***Taske2 BCNF Compliance Assessment for Database Schemas***

*Restaurant Delivery System Database Schemas Overview*

* Delivery (Did: INTEGER, Vid: INTEGER, DDate: DATE, Dtime:TIME, Is\_delivered :BOOLEAN)

Foreign Key: Vid references Vehicle

* Orders (Order\_id : INTEGER, Cid: INTEGER, Order\_city: VARCHAR(200))

Foreign Key: Cid references Customer

* ComposedBy (Did: INTEGER, Order\_id: INTEGER)

Foreign Key: \_Did\_ references Delivery, Foreign Key: Order\_id references Order

* Customer (Cid: INTEGER, CName: VARCHAR(20),Address: VARCHAR(30), Phone\_Number: VARCHAR(20), Pass\_word: VARCHAR(20), Email: VARCHAR(200))
* Pays (PyId: INTEGER, Order\_id: INTEGER)

Foreign Key: \_PyId\_ references Payment, Foreign Key: \_Order\_id\_ references Order

* Payment (PyId : INTEGER, PDate: DATE)
* Contains (Ordr\_id: INTEGER, Rid: INTEGER, MName: VARCHAR(200))

Foreign Key: Oid references Order, Foreign Key: (Rid, MName) references Meals

* Restaurants (Rid: INTEGER, RName: VARCHAR(20), RCity: VARCHAR(200))
* Meal (Mname: Varchar (20), Rid: Integer, price:REAL)

Foreign Key: \_Rid\_ references Restaurants

* Payment\_Cash (Pcid: INTEGER, Pcprice: REAL, Budget: REAL, Pid: INTEGER,pyid:integer)

Foreign Key: \_Pid\_ references DeliveryPersonal

Foreign Key: \_Pyid\_ references Payment

* PaymentOnline (Poid: INTEGER, POPrice: REAL, Credit\_Card:INTEGER,Ptime:TIME,pyid:Integer)

Foreign Key: \_Pyid\_ references Payment

* Delivery\_Personal (Pid: INTEGER, Vid: INTEGER , Is\_fired:BOOLEAN ,DPCity:VARCHAR(20), PName: VARCHAR(20), DP\_Phone\_Number: VARCHAR(20))

Foreign Key: \_Vid\_ references Vehicle

* Vehicle (Vid: INTEGER, Max\_meals: INTEGER, Speed: REAL, Is\_working:BOOLEAN)

***Summary:***

In this task, we meticulously examined and scrutinized the schemas of all entities and relationships in our Restaurant Delivery System database. The objective was to ensure that each schema adhered to the Boyce-Codd Normal Form (BCNF). The analysis revealed that all the schemas were inherently dependency-preserving BCNF, indicating a robust and normalized structure. Consequently, no decomposition was deemed necessary, affirming the efficiency and coherence of the initial schema design.

***Task4.Database Definition Language (DDL) Script and Views for Access Control***

* ***Comprehensive DDL Script and Access Control Views***

In this task, I present the DDL script file, um6p-cs-introdb-project4-ddl.sql, encapsulating SQL commands for creating tables, indexes, constraints, and triggers within the Restaurant Delivery System database. Additionally, I've introduced strategically crafted views for access control, allowing specific users to interact with designated subsets of data. The views are tailored to fulfill business requirements and have been granted relevant privileges using Data Control Language (DCL) commands. The script is thorough and well-documented, offering a comprehensive foundation for the database's structure and security measures.

* **Indexes Overview:**

**idx\_Email on Customer(Email):**

* **Purpose**: This index is designed to enhance performance for login operations and searching for a customer by their email.
* **Explanation**: The index on the Email column in the Customer table facilitates rapid retrieval of customer information when logging in or searching for a customer based on their email address.

**idx\_Vid on Delivery\_Personal(Vid):**

* **Purpose**: This index is created to expedite the retrieval of delivery personnel details based on the vehicle they are assigned.
* **Explanation**: By indexing the Vid column in the Delivery\_Personal table, operations that involve searching for delivery personnel based on their associated vehicle can be performed more efficiently.

**idx\_Combine\_CID\_OID on Orders(Order\_id, Cid):**

* **Purpose**: This compound index is beneficial for various operations, including retrieving orders for a specific customer and supporting join operations.
* **Explanation**: The compound index on Order\_id and Cid in the Orders table enhances the speed of queries that involve fetching orders for a particular customer or joining operations that involve the Orders table.

**idx\_Restaurant\_Name on Restaurant(RName):**

* **Purpose**: This index aims to facilitate quick retrieval of restaurant details when searching by name.
* **Explanation**: The index on the RName column in the Restaurant table streamlines operations that involve searching for restaurant details based on their names, a common scenario in real-world applications.

**idx\_Combine\_Rid\_MName on Meal(Rid, MName):**

* **Purpose**: This compound index is useful for operations such as fetching meals offered by a specific restaurant and supporting join operations.
* **Explanation**: The compound index on Rid and MName in the Meal table optimizes queries that involve fetching meals offered by a particular restaurant or join operations that involve the Meal table.
* ***Triggers Overview:***

**Update Delivery Personal Trigger:**

* **Purpose**: This trigger ensures that if a delivery person is marked as fired (Is\_fired is TRUE) and has an associated vehicle (Vid is not NULL), the vehicle is assigned to another delivery person if available.
* **Explanation**: If a delivery person is fired and has an associated vehicle, this trigger updates the Vid of an available non-fired delivery person to the fired person's vehicle. It also sets the vehicle to NULL for the fired person.

**Check Delivery Date Trigger:**

* **Purpose**: This trigger checks the delivery date before insertion to ensure that it is not in the future.
* **Explanation**: Before inserting a new record into the Delivery table, this trigger checks whether the delivery date (DDate) is in the future. If so, it raises an error to prevent the insertion.

**Check Payment Date Trigger:**

* **Purpose**: This trigger checks the payment date before insertion to ensure that it is not in the future.
* **Explanation**: Before inserting a new record into the Payment table, this trigger checks whether the payment date (PDate) is in the future. If so, it raises an error to prevent the insertion.

**Vehicle Shut Down Trigger:**

* **Purpose**: This trigger manages the business logic when a vehicle is marked as shut down (Is\_working is set to FALSE).
* **Explanation**: If a vehicle is marked as shut down and was previously working, this trigger reassigns deliveries from the shut-down vehicle to another available vehicle, but only if a driver is assigned to the shut-down vehicle and an available vehicle exists.

***Database Security Script Overview:***

**Delivery Personnel Privileges:**

* **Description**: This section focuses on granting privileges to delivery personnel, allowing them to update the delivery status and select deliveries.

**Restaurant Manager Privileges:**

* **Description**: This part grants restaurant managers full privileges to make changes in their respective meals to boost their business. However, access to the meal RID is restricted.

**Customer Privileges:**

* **Description**: This section provides customers with the privilege to track the history of their orders, ensuring privacy by limiting their ability to delete or modify orders.

***Task5.Data Insertion, and Logging***

* **Data Insertion**
* **Objective**: Populate the database with data, either synthetic or provided by the customer.
* **Actions** Taken: Executed SQL commands for inserting or importing data into the database (um6p-cs-introdb-project4-data.sql).
* **Outcome**: The database now contains relevant data necessary for testing, analysis, or application development.

***Logging Data Insertion***

* **Objective**: Maintain a MySQL log file (um6p-cs-introdb-project4-data-log.txt) to demonstrate the successful loading/insertion of data.
* **Actions** Taken: Recorded the execution status of data insertion commands in a log file, providing insights into the success or failure of the process.
* **Outcome**: A documented log confirming the successful insertion of data, ensuring data integrity and aiding in troubleshooting potential issues.

**Challenges and Reflections on Deliverable 2: A Comprehensive Database Implementation**

Undertaking the second deliverable of our database project proved to be an intricate process, laden with challenges and valuable learning experiences. One of the foremost hurdles we encountered was the meticulous task of populating data. It entailed not only inserting synthetic information but also establishing meaningful relationships between primary and foreign keys. This process, though time-consuming, was crucial to creating a database reflective of real-world scenarios. Additionally, ensuring the integrity of the database involved grappling with a multitude of constraints that mimic the complexities encountered in practical applications.

The conundrum of maintaining referential integrity, enforcing unique constraints, and handling errors in the code became apparent as we delved deeper into the implementation. The challenge lay not just in making the database functional but in making it resilient to potential real-world scenarios, where data integrity is paramount. Managing constraints and coding errors demanded a keen eye for detail and a commitment to delivering a robust and error-tolerant system.

The meticulous task of achieving a minimal cover for our relations presented yet another challenge. While striving for the Boyce-Codd Normal Form (BCNF), we had to navigate the intricacies of functional dependencies, ensuring our schema design was both comprehensive and optimized. This process, vital for database performance, demanded an understanding of the theoretical underpinnings as well as a keen awareness of the real-world implications of our choices.

Estimating the time investment for each subtask proved to be an exercise in realistic planning. The population of data consumed a significant portion of our allocated time, requiring two dedicated days to establish a robust foundation. Determining the minimal cover and achieving BCNF, though time-intensive, were indispensable steps in creating an optimized and efficient database schema. The Data Definition Language (DDL) and Data Control Language (DCL) tasks, spanning a day and two days respectively, contributed to the holistic development of the database, ensuring not only structural soundness but also security and access control.