FITNESS APP

Final Group Project

Group 1

Amandus Fernando 8989765

Ya-Jwu Jang 9015996

Gavril Koryakin 8986960

Sophie Obiyan 8975025

PROG8651-SEC1

Database Management

Conestoga College

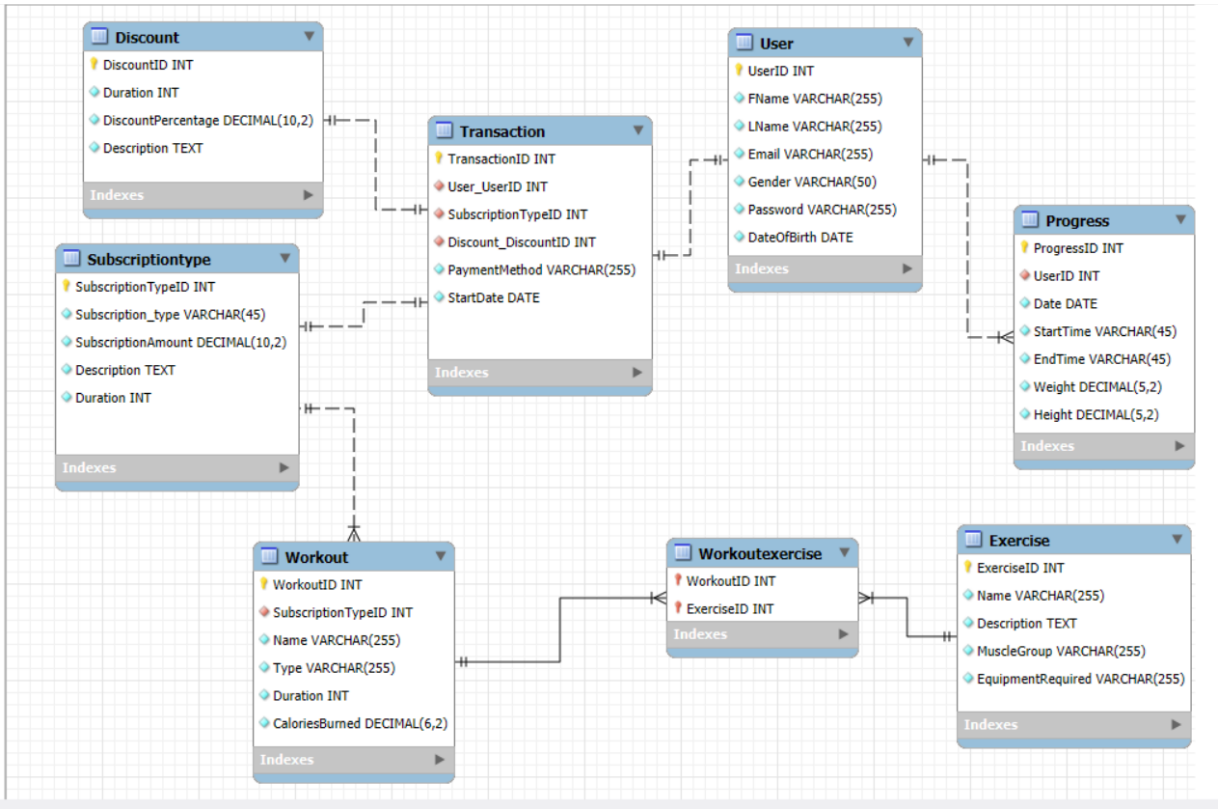
Yasser Elmankabady

DEC. 08 2024

1. **ER DIAGRAM** – SOPHIE OBIYAN 8975025

Our company in fitness apps helps people live healthier by enabling them to have a highly personalized fitness experience. Workouts are easy to track, progress is easily monitored, and subscriptions are easily managed, making it an incredibly smooth and fun user experience. In designing the ER diagram, the focus is on efficient data management, app scalability, and insights for users to achieve their goals.

ER DIAGRAM:



Company Goals:

- **Work Outs**: We provide users with exercise regimens specially crafted to satisfy their every goal and preferential cause. No 'one-size-fits-all' idea here!
- **Tracking progress** using the metrics of weight, height, and other fitness statistics can be done by the user.
- **Easy Subscription Management**: The subscription plans are flexible, available at a discount, and easy for users to choose from according to their needs.
- **User Engagement**: By using data-driven insights, we'll provide feedback and motivation to keep users on track.

Relationships in the ER Diagram

Our business Rule indicates the relationship per user and it states;

User – Transaction

Each transaction is tied to one user. This helps us track subscriptions and purchases easily.

- **Relationship:** One user can have one and only one transaction at a time (1:1).

Transaction – SubscriptionType

This setup lets us handle lots of users subscribing to different plans while keeping everything organized.

- **Relationship:** Each transaction is linked to one subscription type, and a subscription type can have one and only transactions (1:1).

Transaction – Discount

This makes it easy to apply discounts across different transactions.

- **Relationship:** A transaction can have one discount, and a discount can only apply to one transaction at a time (1:1).

SubscriptionType - Workout

This helps us connect workouts to specific user based on their subscriptiontype for better tracking and personalization.

- **Relationship:** A workout is tied to one subscriptiontype, and a subscriptiontype can include multiple workouts (1:M).

Workout – WorkoutExercise

This many-to-many relationship is managed through the WorkoutExercise table

- **Relationship:** A workout can have many exercises (1:M), and each exercise can be part of multiple workouts (M:N).

Exercise – WorkoutExercise

The WorkoutExercise table makes it easy to create diverse workout routines.

- **Relationship:** Exercises can show up in multiple workouts, and workouts can include multiple exercises (N:M).

User – Progress

This lets us track fitness metrics like weight and height over time for each user

- **Relationship:** A user can have multiple progress entries, but each progress entry belongs to one user (1:M).

2. **DDL AND NORMALIZATION -** Gavril Koryakin 8986960

2.1. **DDL**

In accordance with the ER diagram, a DDL statement was created. A **fitness_app** database and 8 tables were created: **Users**, **Progress**, **Transactions**, **Workout**, **WorkoutExercise**, **Discount**, **SubscriptionType**, and **Exercise**.

The **Users** table has 7 attributes: **UserID, FName, LName, Email, DateOfBirth, Gender, Password.**

- **UserID** is the PRIMARY KEY and AUTO_INCREMENT
- **Email**, **Password** are UNIQUE
- All attributes are NOT NULL

The **Exercise** table has 5 attributes: **ExerciseID**, **Name**, **Description**, **MuscleGroup**, **EquipmentRequired.**

- **ExerciseID** is the PRIMARY KEY and AUTO_INCREMENT
- All attributes are NOT NULL

The **Progress** table has 7 attributes: **ProgressID**, **UserID**, **Date**, **Start_Time**, **End_Time**, **Weight**, **Height.**

- **ProgressID** is the PRIMARY KEY and AUTO_INCREMENT
- **UserID** is a FOREIGN KEY referencing **User(UserID)**
- All attributes are NOT NULL

The **Discount** table has 4 attributes: **DiscountID**, **Duration**, **DiscountPercentage**, **Description.**

- **DiscountID** is the PRIMARY KEY and AUTO_INCREMENT
- All attributes are NOT NULL

The **SubscriptionType** table has 4 attributes: **SubscriptionTypeID**, **Subscription_type**, **SubscriptionAmount**, **Description.**

- **SubscriptionTypeID** is the PRIMARY KEY and AUTO_INCREMENT
- All attributes are NOT NULL

The **Transaction** table has 6 attributes: **TransactionID**, **User_UserID**, **Discount_DiscountID**, **SubscriptionTypeID**, **PaymentMethod**, **StartDate.**

- **TransactionID** is the PRIMARY KEY and AUTO_INCREMENT
- **User_UserID** is a FOREIGN KEY referencing **User(UserID)**
- **Discount_DiscountID** is a FOREIGN KEY referencing **Discount(DiscountID)**
- **SubscriptionTypeID** is a FOREIGN KEY referencing **SubscriptionType(SubscriptionTypeID)**
- All attributes are NOT NULL

The **Workout** table has 6 attributes: **WorkoutID**, **SubscriptionTypeID**, **Name**, **Type**, **Duration**, **CaloriesBurned.**

- **WorkoutID** is the PRIMARY KEY and AUTO_INCREMENT
- **SubscriptionTypeID** is a FOREIGN KEY referencing **SubscriptionType(SubscriptionTypeID)**
- All attributes are NOT NULL

The **WorkoutExercise** table has 2 attributes: **WorkoutID**, **ExerciseID**

- (**WorkoutID**, **ExerciseID**) together form the PRIMARY KEY
- **WorkoutID** is a FOREIGN KEY referencing **Workout(WorkoutID)**
- **ExerciseID** is a FOREIGN KEY referencing **Exercise(ExerciseID)**

- All attributes are NOT NULL

2.2. **NORMALIZATION**

The base database contains 5 tables: **Users**, **Progress**, **Transactions**, **Workout**, **WorkoutExercise**.
To normalize the database to 1NF (First Normal Form):

- Checked all duplicate groups and removed them.
- Identified all primary keys.
- Found all dependencies:
    - Transitive dependencies are found in the **Transactions** table.
    - **WorkoutExercise** has a partial dependency on a composite primary key.

For the 1NF-to-2NF conversion, the following steps were taken:

- Created new tables to eliminate partial dependencies: **WorkoutExercise**, **Exercise**
- Reassigned the corresponding dependent attributes.

For the 2NF-to-3NF conversion, the following steps were taken:

- Created new tables to eliminate transitive dependencies: **Discount**, **SubscriptionType**, **Transaction**
- Reassigned the corresponding dependent attributes.

Thus, the 3NF database consists of 8 tables (**Users, Progress, Transactions, Workout, WorkoutExercise, Discount, SubscriptionType, Exercise**), which:

- Are in table format, have no repeating groups, and have primary keys identified.
- Contain no partial dependencies.
- Contain no transitive dependencies.

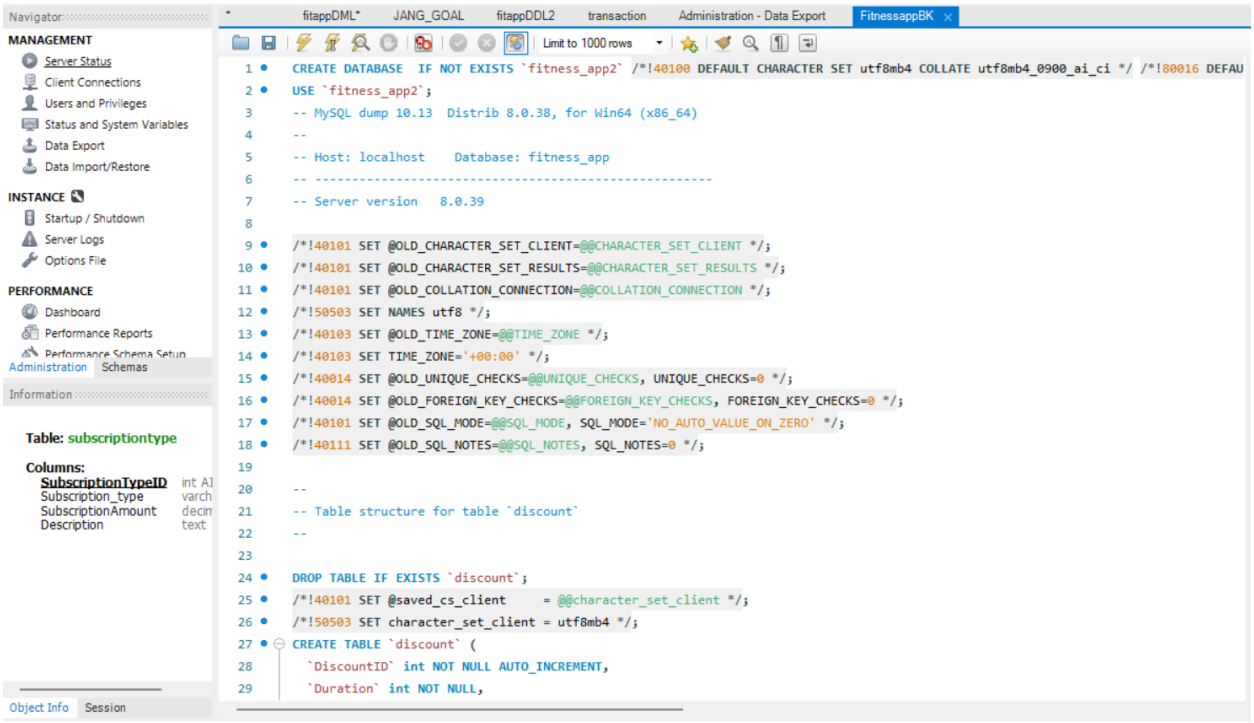
3. **DML AND BACKUP** – Ya-Jwu Jang 9015996

- **DML**

  In our databases, we insert the following data:

  1. 20 **users**
  2. 30 types of **exercise**
  3. 50 records of users' **progress**
  4. 25 records of users' **transactions**
     - 5 users have already started with new transactions
  5. 10 **workout** plans
  6. 30 **workout-exercise** (each workout contains 3 exercises)
  7. 4 types of **discounts**
     - 5% discount for 30 days
     - 10% discount for 90 days
     - 15% discount for 180 days
     - 20% discount for 365 days
  8. 3 levels of **subscriptions**
     - Free plan: Access to 1 workout plan at no cost.
     - Premium: Access to 3 workout plans for CAD 49.99 per month
     - VIP: All access to workout plans for CAD 99.99 per month

- **Backup:** Use the Data Export to backup the FitnessApp database named FitnessappBK



4. **DATA MART-** Amandus Fernando-8989765

The Star schema consists of four-dimension tables and one fact table:

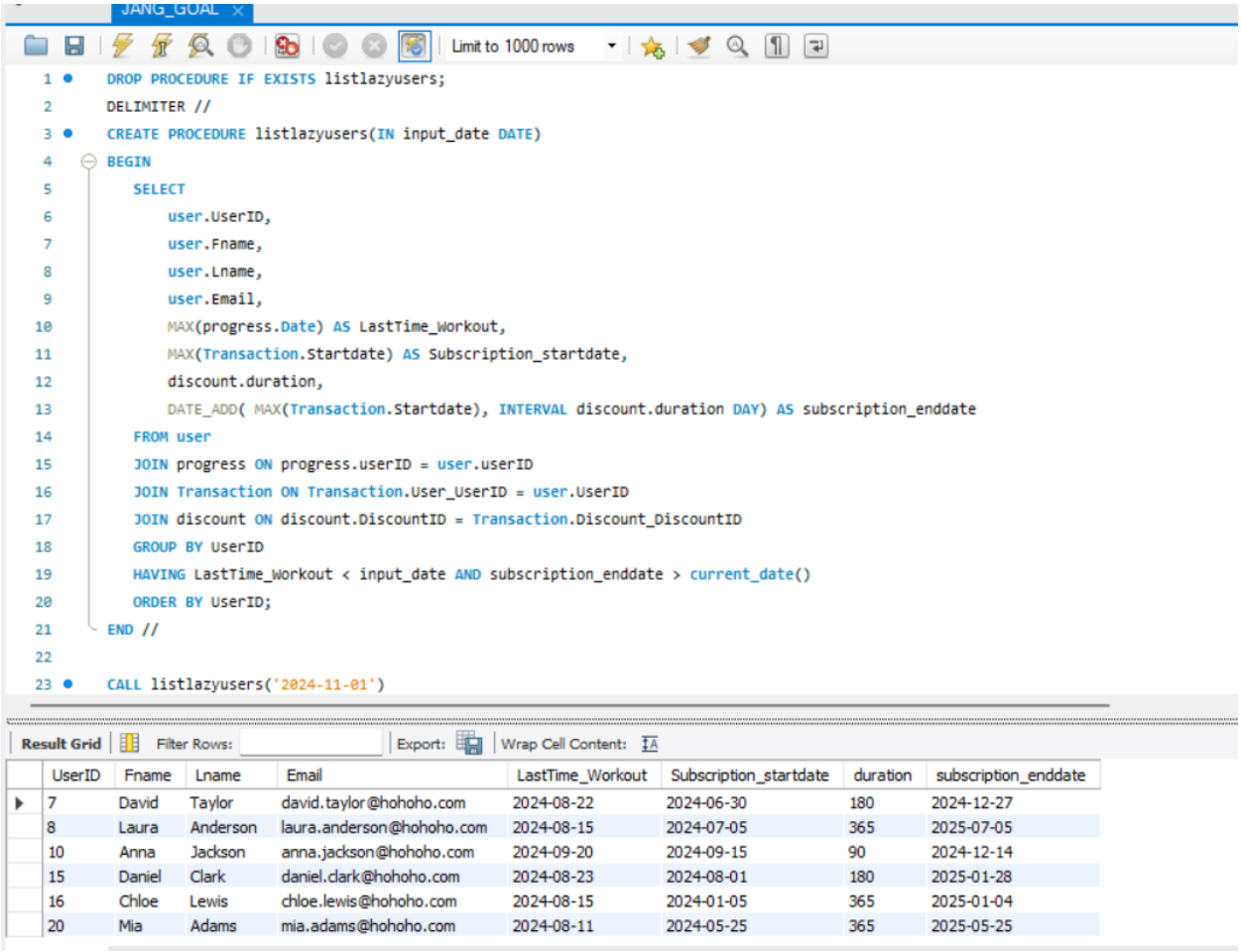
- **Dimension Tables**:
    - **DimUser**: Contains user details such as UserID, FirstName, LastName, Gender, and DateOfBirth.
    - **DimWorkout**: Represents workout details, including WorkoutID, Name, Type, and Duration.
    - **DimSubscriptionType**: Details subscription types with SubscriptionTypeID, SubscriptionType, and SubscriptionAmount.
    - **DimDate**: A date dimension to support temporal queries, containing attributes like FullDate, Day, Month, Year, and DayOfWeek.
- **Fact Table**:
    - **FactAnalytics**: Stores analytics data, including UserID, DateID, SubscriptionTypeID, WorkoutID, CaloriesBurned, and DiscountValue.
- **Schema Creation**:
    - A schema dmart is created for housing the star schema.

    - Dimension and fact tables are defined with appropriate primary and foreign keys to establish relationships.

- **Data Transformation and Loading**:
    - Data is extracted from the source (fitness_app) and transformed into a format suitable for dimensional modeling.
    - Insert operations ensure data deduplication using SELECT DISTINCT.
- **Data Population**:
    - **DimUser**: Populates user attributes.
    - **DimWorkout**: Populates workout details.
    - **DimSubscriptionType**: Inserts subscription-related data.
    - **DimDate**: Converts transactional dates into a date dimension with relevant breakdowns (day, month, year, etc.).
    - **FactAnalytics**: Aggregates data from multiple tables to store analytical metrics like CaloriesBurned (summed) and DiscountValue (calculated using discount duration and subscription amount).
- **Aggregation and Calculations**:
    - Metrics like total calories burned and discount values are computed during the ETL.

5. GOALS

- JANG:
    - Create a procedure to list customers who have not been active since Input_date and are still subscribed. And then we will email them to remind them to come back to work out!
    - Input 2024-11-01
    - Result: 6 users



```
DROP PROCEDURE IF EXISTS listlazyusers;
DELIMITER //
CREATE PROCEDURE listlazyusers(IN input_date DATE)
BEGIN
   SELECT
       user.UserID,
       user.Fname,
       user.Lname,
       user.Email,
       MAX(progress.Date) AS LastTime_Workout,
       MAX(Transaction.Startdate) AS Subscription_startdate,
       discount.duration,
       DATE_ADD( MAX(Transaction.Startdate), INTERVAL discount.duration DAY) AS subscription_enddate
   FROM user
   JOIN progress ON progress.userID = user.userID
   JOIN Transaction ON Transaction.User_UserID = user.UserID
   JOIN discount ON discount.DiscountID = Transaction.Discount_DiscountID
   GROUP BY UserID
   HAVING LastTime_Workout < input_date AND subscription_enddate > current_date()
   ORDER BY UserID;
END //

CALL listlazyusers('2024-11-01')
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

UserID	Fname	Lname	Email	LastTime_Workout	Subscription_startdate	duration	subscription_enddate
7	David	Taylor	david.taylor@hohoho.com	2024-08-22	2024-06-30	180	2024-12-27
8	Laura	Anderson	laura.anderson@hohoho.com	2024-08-15	2024-07-05	365	2025-07-05
10	Anna	Jackson	anna.jackson@hohoho.com	2024-09-20	2024-09-15	90	2024-12-14
15	Daniel	Clark	daniel.clark@hohoho.com	2024-08-23	2024-08-01	180	2025-01-28
16	Chloe	Lewis	chloe.lewis@hohoho.com	2024-08-15	2024-01-05	365	2025-01-04
20	Mia	Adams	mia.adams@hohoho.com	2024-08-11	2024-05-25	365	2025-05-25

- **AMANDUS:**
  - To track and identify users who have not met their fitness goals within the current month, based on two key performance indicators: total calories burned and workout frequency.
  - Result: 2 Users who didn't meet either the calorie or workout goal are selected.

```
Limit to 1000 rows
133     SELECT
134         U.UserID,
135         U.FirstName,
136         U.LastName,
137         SUM(FA.CaloriesBurned) AS TotalCaloriesBurned,
138         COUNT(FA.DateID) AS WorkoutFrequency,
139         D.Month AS Month,
140         D.Year AS Year
141     FROM DimUser U
142     LEFT JOIN FactAnalytics FA ON U.UserID = FA.UserID
143     LEFT JOIN DimDate D ON FA.DateID = D.DateID
144     WHERE D.Month = MONTH(CURDATE())   -- Current month
145         AND D.Year = YEAR(CURDATE())   -- Current year
146     GROUP BY
147         U.UserID, D.Month, D.Year
148     HAVING
149         (SUM(FA.CaloriesBurned) < calorie_threshold
150          OR COUNT(FA.DateID) < workout_threshold)
151     ORDER BY
152         U.UserID;
153
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

UserID	FirstName	LastName	TotalCaloriesBurned	WorkoutFrequency	Month	Year
14	Olivia	Garcia	4800.00	3	12	2024
15	Daniel	Clark	1200.00	1	12	2024

- **Gavril Koryakin 8986960:**

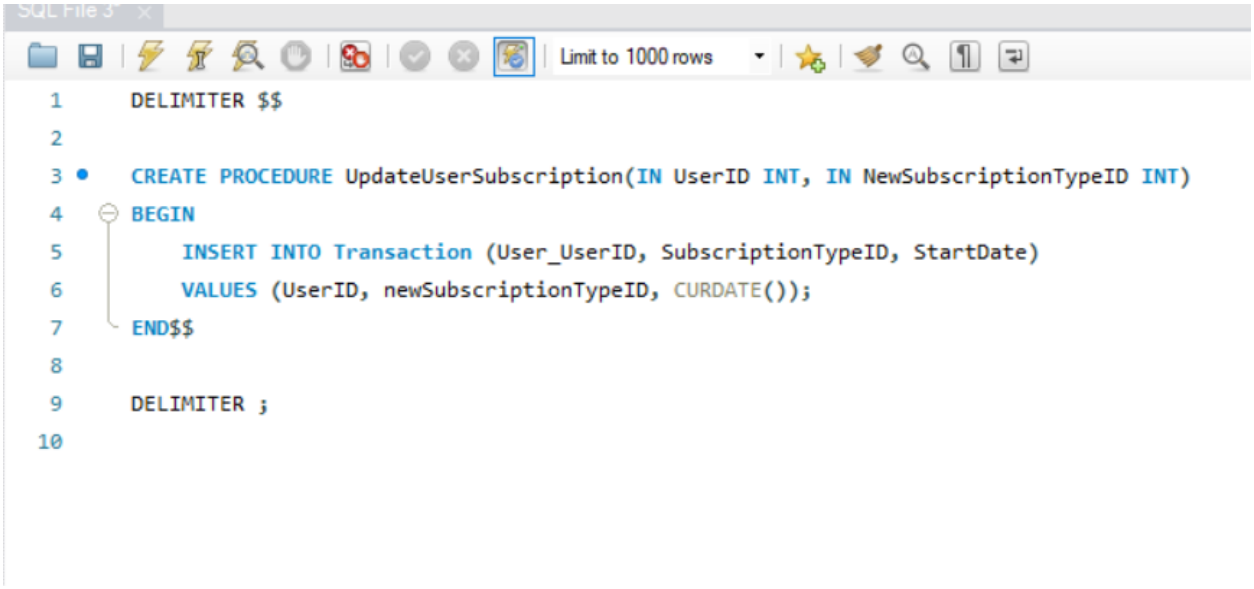
The procedure AVGDiscountMonth returns the average discount percentage applied to transactions within a specified month.

```
DELIMITER $$

CREATE PROCEDURE AVGDiscountMonth(IN input_month INT)

BEGIN

SELECT AVG(d.DiscountPercentage) AS AvgDiscount

FROM Transaction t

JOIN Discount d ON t.Discount_DiscountID = d.DiscountID

WHERE MONTH(t.StartDate) = input_month;
```

END$$

DELIMITER ;

- Sophie Obiyan:



  - Managing subscription updates for users:
    The goal of this procedure is to ensure that changes to a user's subscription are recorded systematically without altering historical data. Each time a user renews, upgrades, or modifies their subscription, a new record is inserted into the Transaction table. This approach allows the system to maintain a complete and chronological history of all user subscriptions, supporting future analytics, audits, and user activity tracking.