GYM AND DIET PLANNER

A PROJECT REPORT [INTERNSHIP REPORT]

Submitted by

ROHIT KUMAR [RA2311056010279]
BHOOMI JAIN S [RA2311056010282]

Under the Guidance of

(DR. DHANALAKSHMI J)

(Assistant Professor, Data Science And Business Systems)

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in (DATA SCIENCE)

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

MAY 2025

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 21CSP302L - Project report titled “**GYM AND DIET PLANNER**” is the Bonafide work of “**ROHIT KUMAR [RA2311056010279], BHOOMI JAIN S[RA2311056010282]”** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

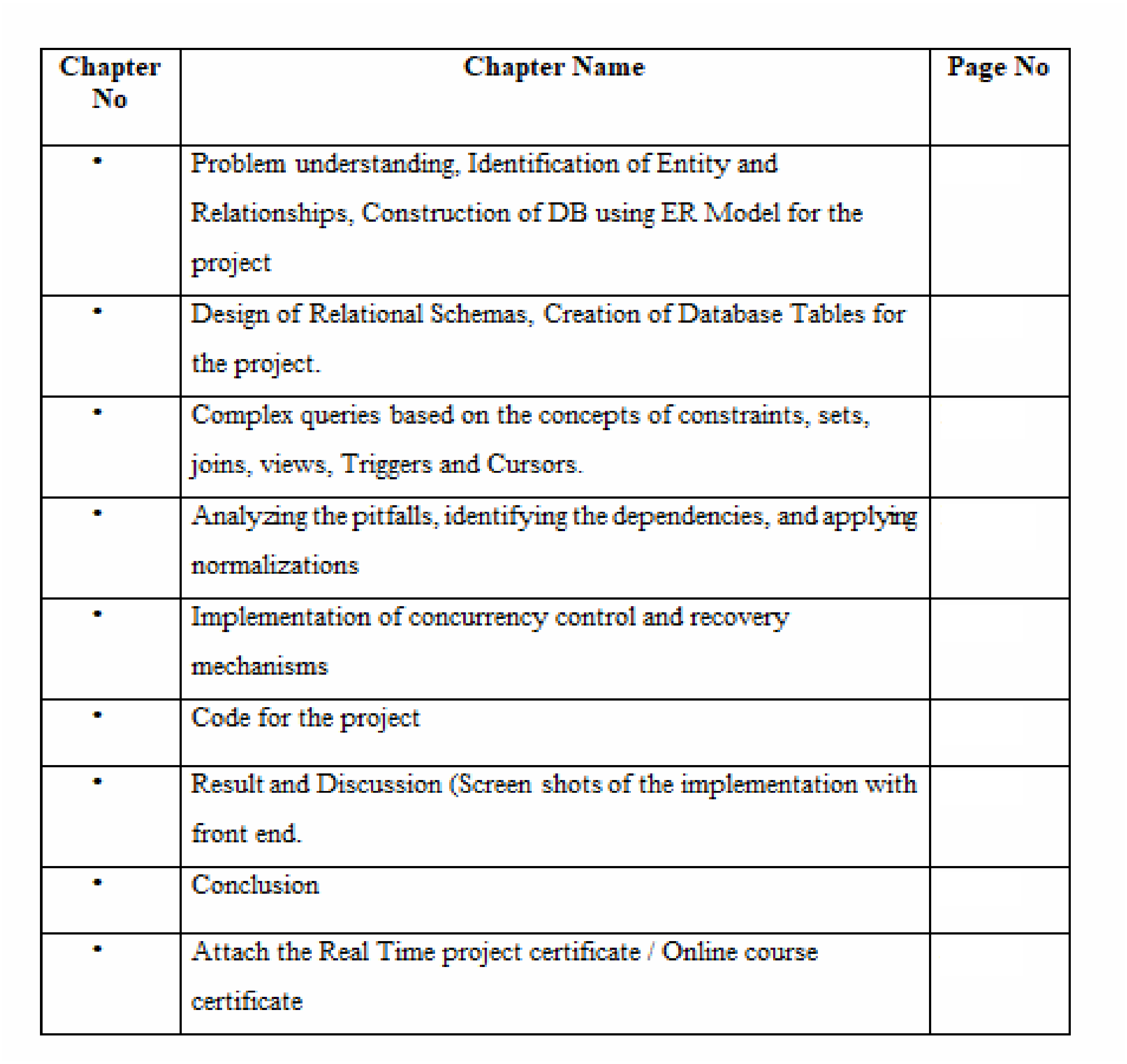
SIGNATURE	**SIGNATURE**
DR. DHANALAKSHMI J	**DR. V.KAVITHA**
ASSISTANT PROFESSOR DEPARTMENT OF Data Science and Business Systems	**PROFESSOR &HEAD** DEPARTMENT OF Data Science and Business Systems

ABSTRACT

This project aims to develop a user-friendly gym and diet planner that provides personalized recommendations for workouts and meal plans. The application utilizes data-driven algorithms to analyze user inputs and offer tailored guidance based on their specific needs and preferences.

By gathering information about an individual's fitness goals, current activity levels, and dietary habits, the application can generate personalized workout routines, meal plans, and progress tracking tools. This personalized approach helps users stay motivated and engaged, as they receive recommendations that are aligned with their unique circumstances and objectives. The app's ability to adapt to each user's evolving needs further enhances its effectiveness in supporting long-term lifestyle changes.

TABLE OF CONTENT



Chapter No	Chapter Name	Page No
•	Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project	
•	Design of Relational Schemas, Creation of Database Tables for the project.	
•	Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors.	
•	Analyzing the pitfalls, identifying the dependencies, and applying normalizations	
•	Implementation of concurrency control and recovery mechanisms	
•	Code for the project	
•	Result and Discussion (Screen shots of the implementation with front end.	
•	Conclusion	
•	Attach the Real Time project certificate / Online course certificate	

1. Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project

Introduction:

Motivation:

The demand for personalized fitness solutions is increasing. This trend is driven by the growing awareness of the importance of health and wellness among the general population. As people become more conscious of their physical and mental well-being, they seek out tools and services that can help them achieve their fitness goals.

In today's fast-paced world, individuals often struggle to find the time and motivation to maintain a consistent exercise routine. The availability of personalized fitness apps and programs can provide the necessary guidance and support to help users overcome these challenges. By offering customized workout plans, nutrition advice, and progress tracking, these solutions empower users to take control of their health and make sustainable lifestyle changes.

Objective:

This project aims to develop a user-friendly gym and diet planner that provides personalized recommendations for workouts and meal plans. The application utilizes data-driven algorithms to analyze user inputs and offer tailored guidance based on their specific needs and preferences.

By gathering information about an individual's fitness goals, current activity levels, and dietary habits, the application can generate personalized workout routines, meal plans, and progress tracking tools. The app's ability to adapt to each user's evolving needs further enhances its effectiveness in supporting long-term lifestyle changes.

Problem statement:

Existing fitness apps often provide generic recommendations that may not be suitable for everyone.
Maintaining motivation and consistency can be challenging, especially without personalized support.
Users may lack access to comprehensive data analysis and insights into their progress.

Challenges:

Developing this app requires creating a seamless, intuitive user experience that encourages long-term engagement. Collecting and analyzing user data to provide personalized recommendations is another key challenge.
The app must also adapt to users' evolving needs over time. Ensuring data security and privacy is critical. Finally, effective marketing and user engagement strategies are needed

to drive widespread adoption.
Additionally, the app will need to integrate with various fitness tracking devices and services to provide a comprehensive solution. Developing robust algorithms to analyze user data and deliver meaningful insights is crucial for the app's success.

Requirements :

The Gym and Diet Planner application will be primarily developed using the Java programming language, which provides the necessary features and capabilities to build a comprehensive and scalable application.
The application will utilize a relational database management system (RDBMS) such as MySQL Key SQL requirements include creating, reading, updating, and deleting data, using SQL joins to combine data, leveraging SQL functions for data manipulation and analysis, ensuring data integrity through SQL transactions, and optimizing database operations with SQL views and stored procedures.

Hardware Requirement:

Processor: Intel i3 or higher
RAM: 4 GB minimum
Storage: 500 GB HDD or higher
Display: 15-inch display with 1080p resolution
Other Hardware: Keyboard, mouse, and stable internet connection (for web access if needed).

Software Requirement :

Programming Language: JAVA (for backend development)
Database Management System: SQL (MySQL)
Operating System: Windows 10 or Linux-based OS
IDE/Editor: VS Code

Architecture and Design:

The Gym and Diet Planner System follows a client-server model, where the frontend communicates with the backend Java application that processes business logic and interacts with the SQL database.

Frontend:

Web interface for user registration, workout plans, diet tracking, and progress reports.
Optional mobile app for tracking on-the-go.
Simple CLI for basic operations.

Backend (Java):

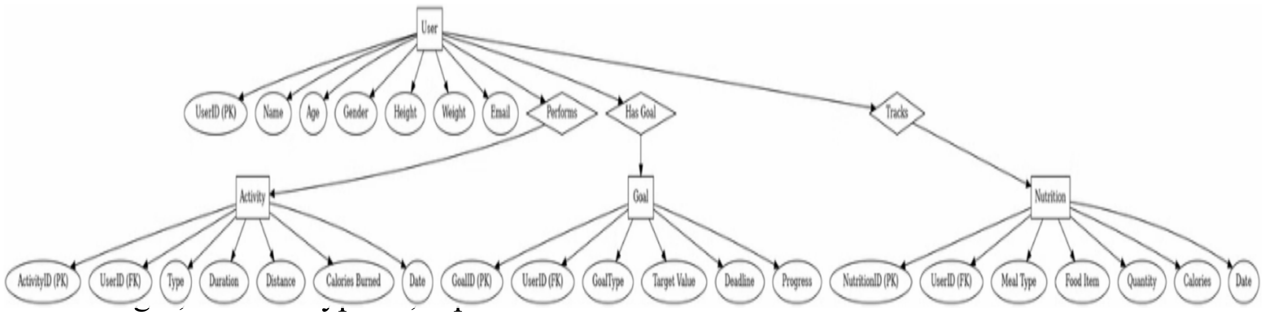
Manages workout routines, diet plans, and fitness reports.
RESTful APIs using Spring Boot for frontend-backend communication.
Algorithms for personalized fitness recommendations.

Database (SQL):

Stores data on users, workouts, diets, and progress.
SQL queries for CRUD operations.
Tables: Users, Workouts, DietPlans, Progress, Nutrition, Logs.

Workflow:

Users register and input data.
Java processes data for personalized plans.



Scalability:

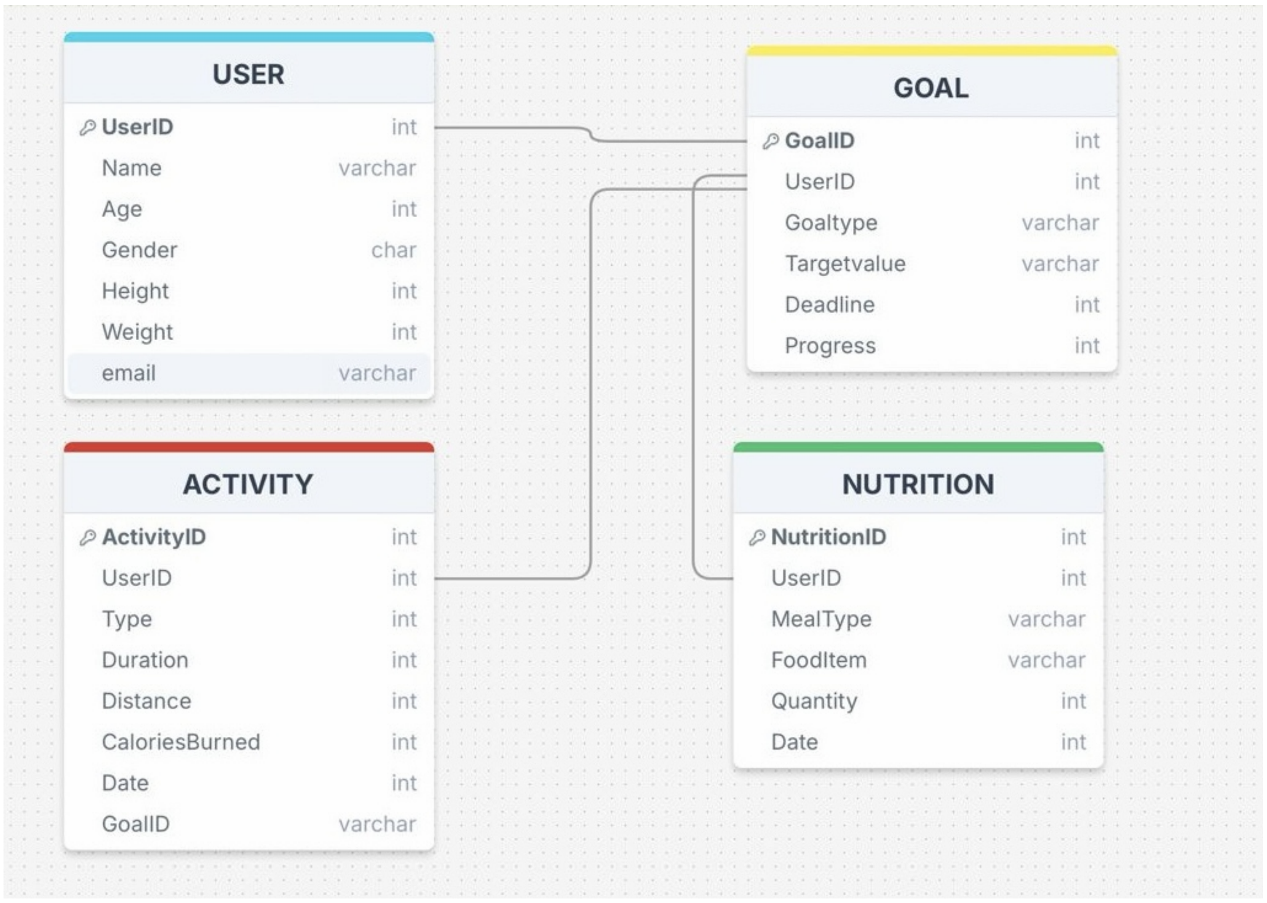
Scalable via microservices, distributed databases, load balancing, caching.

This architecture ensures a robust, secure, and scalable Gym and Diet Planner System.

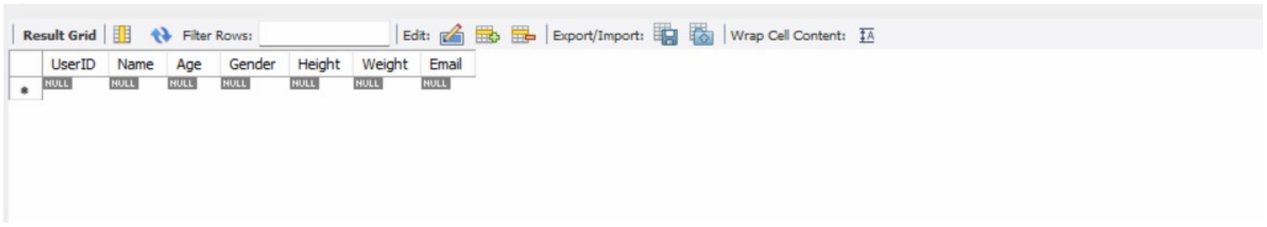
ER DIAGRAM:

2. DESIGN OF RELATIONAL SCHEMAS, CREATION OF DATABASE TABLES FOR THE PROJECT.

RELATIONAL SCHEMA

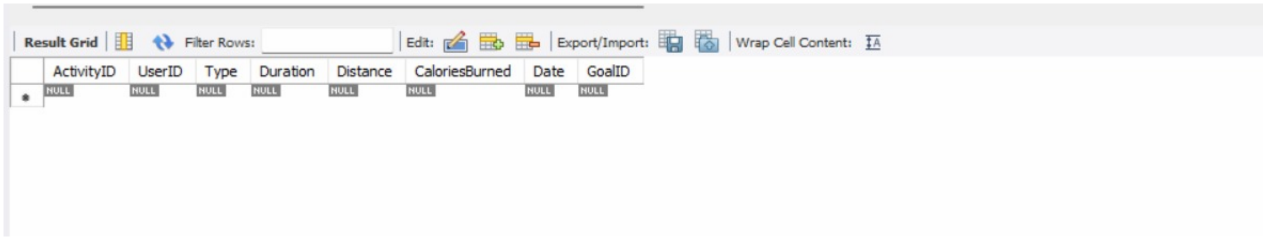


USER TABLE


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

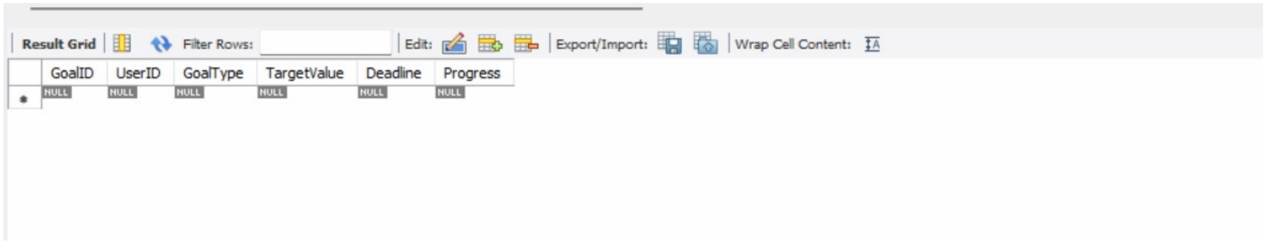
UserID	Name	Age	Gender	Height	Weight	Email
NULL	NULL	NULL	NULL	NULL	NULL	NULL

ACTIVITY TABLE


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

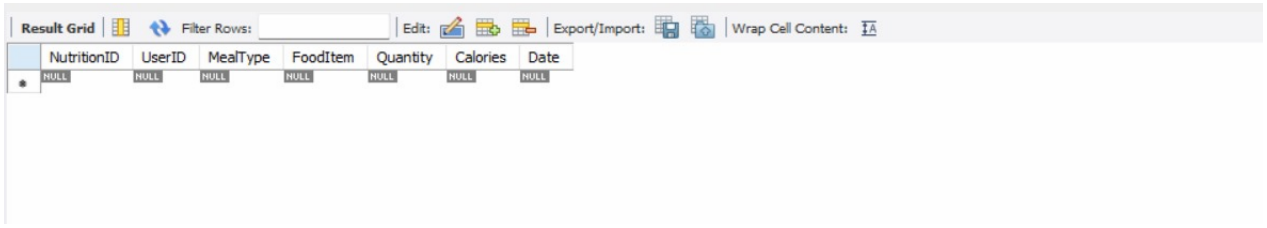
ActivityID	UserID	Type	Duration	Distance	CaloriesBurned	Date	GoalID
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

GOAL TABLE


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

GoalID	UserID	GoalType	TargetValue	Deadline	Progress
NULL	NULL	NULL	NULL	NULL	NULL

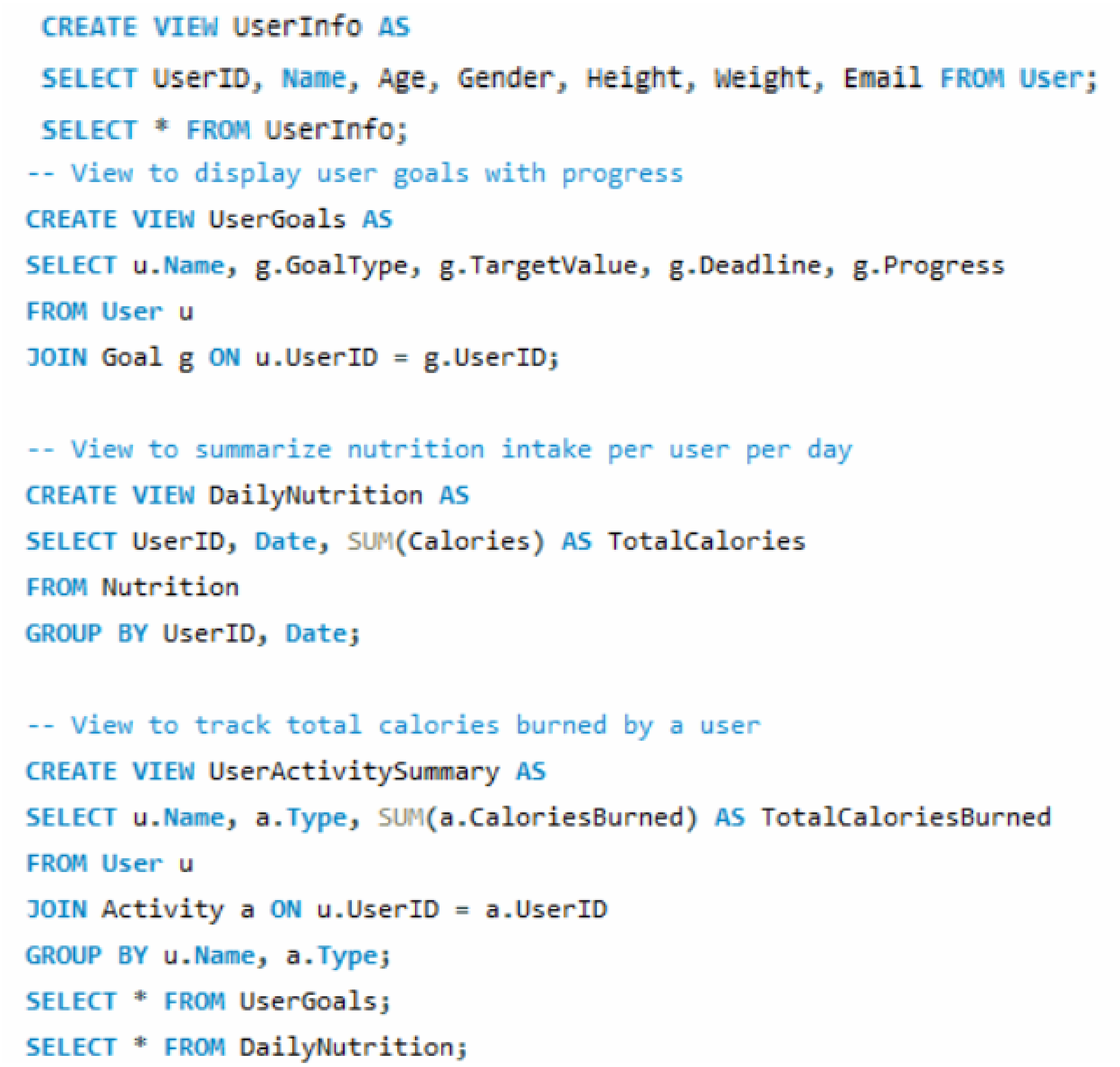
NUTRITION TABLE


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

NutritionID	UserID	MealType	FoodItem	Quantity	Calories	Date
NULL	NULL	NULL	NULL	NULL	NULL	NULL

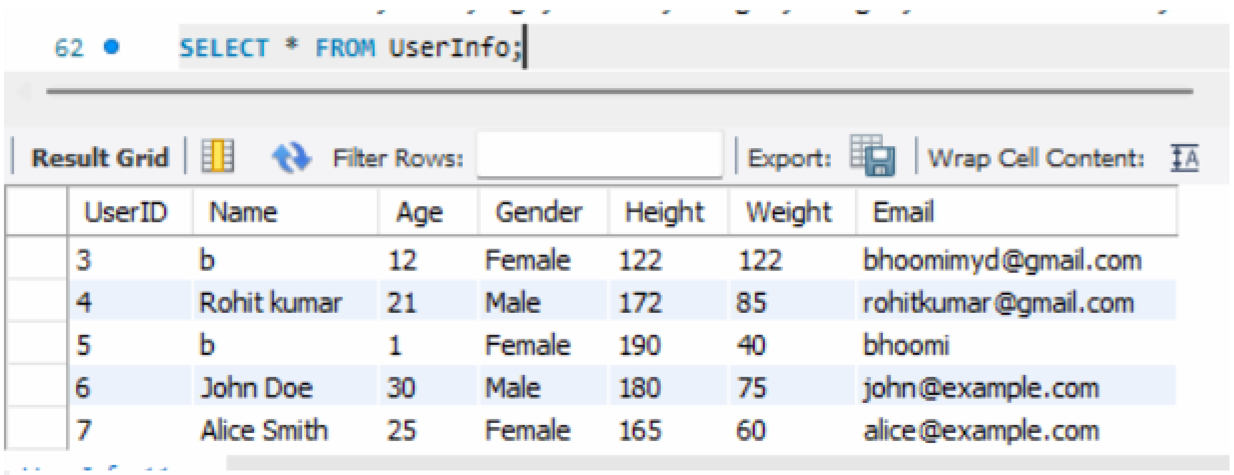
3. COMPLEX QUERIES BASED ON THE CONCEPTS OF CONSTRAINTS, SETS, JOINS, VIEWS, TRIGGERS AND CURSORS.

CREATING VIEWS



```
 CREATE VIEW UserInfo AS
 SELECT UserID, Name, Age, Gender, Height, Weight, Email FROM User;
 SELECT * FROM UserInfo;
-- View to display user goals with progress
CREATE VIEW UserGoals AS
SELECT u.Name, g.GoalType, g.TargetValue, g.Deadline, g.Progress
FROM User u
JOIN Goal g ON u.UserID = g.UserID;

-- View to summarize nutrition intake per user per day
CREATE VIEW DailyNutrition AS
SELECT UserID, Date, SUM(Calories) AS TotalCalories
FROM Nutrition
GROUP BY UserID, Date;

-- View to track total calories burned by a user
CREATE VIEW UserActivitySummary AS
SELECT u.Name, a.Type, SUM(a.CaloriesBurned) AS TotalCaloriesBurned
FROM User u
JOIN Activity a ON u.UserID = a.UserID
GROUP BY u.Name, a.Type;
SELECT * FROM UserGoals;
SELECT * FROM DailyNutrition;
```

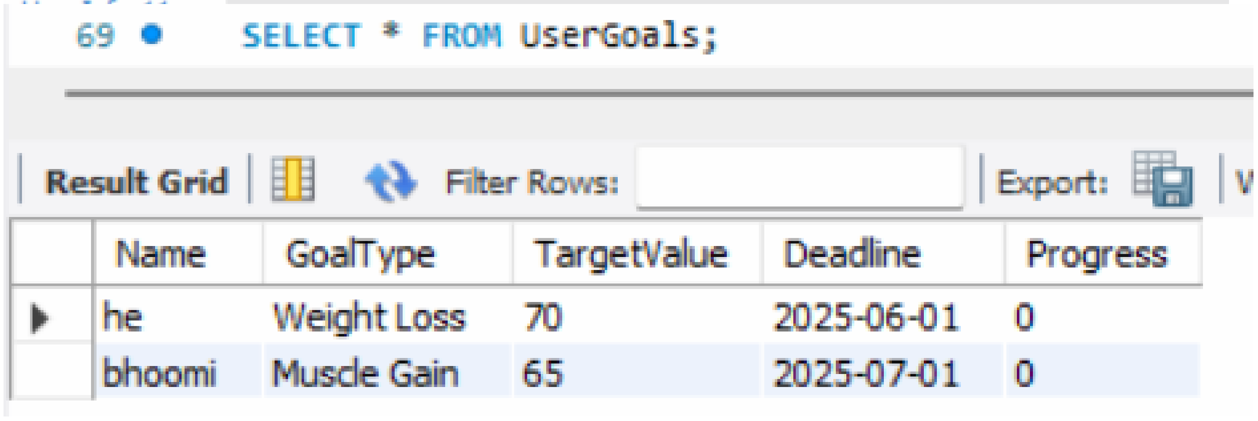
SAMPLE OUTPUT



```
SELECT * FROM UserInfo;
```

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

UserID	Name	Age	Gender	Height	Weight	Email
3	b	12	Female	122	122	bhoomimyd@gmail.com
4	Rohit kumar	21	Male	172	85	rohitkumar@gmail.com
5	b	1	Female	190	40	bhoomi
6	John Doe	30	Male	180	75	john@example.com
7	Alice Smith	25	Female	165	60	alice@example.com

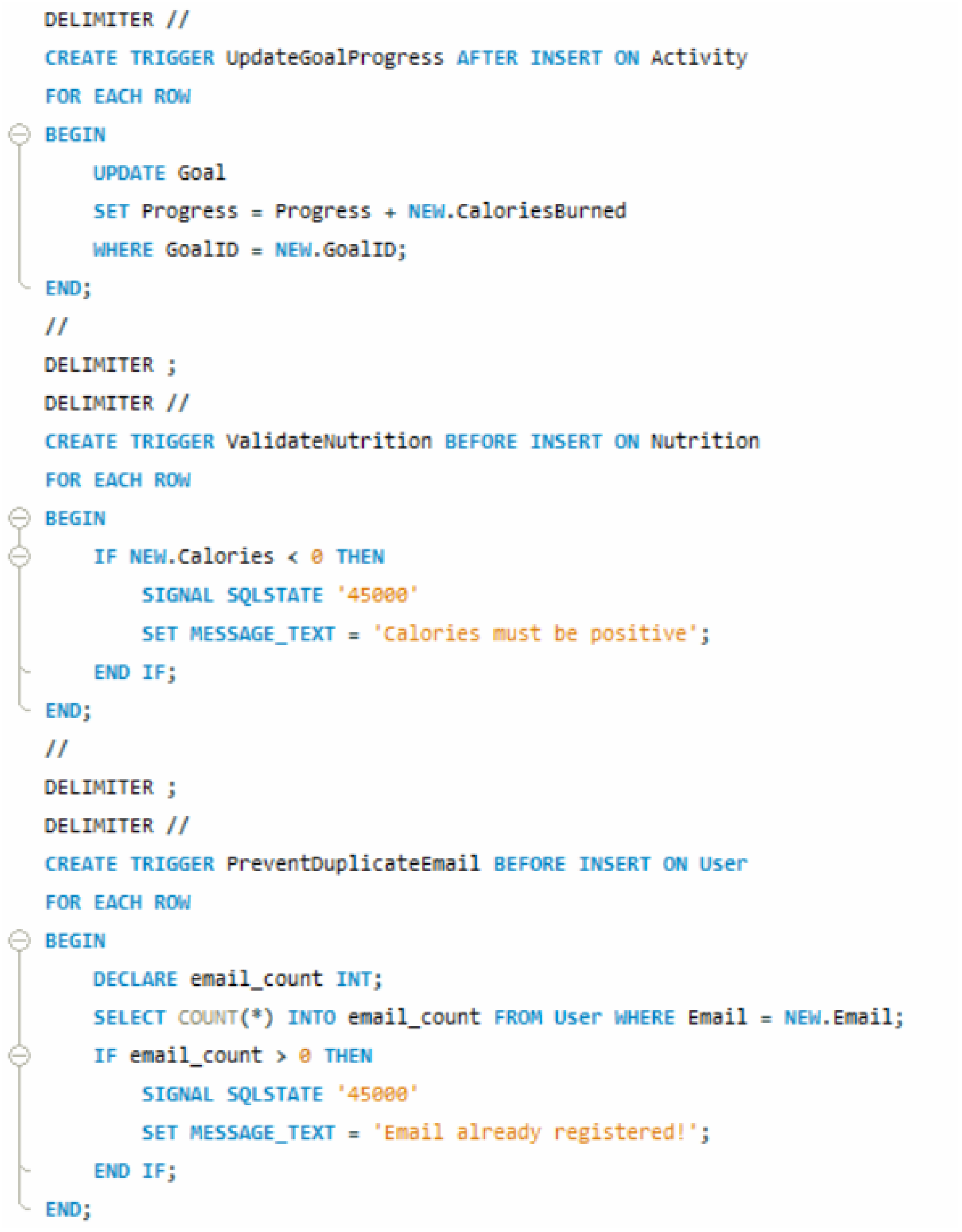


```
SELECT * FROM UserGoals;
```

Result Grid | Filter Rows: | Export:

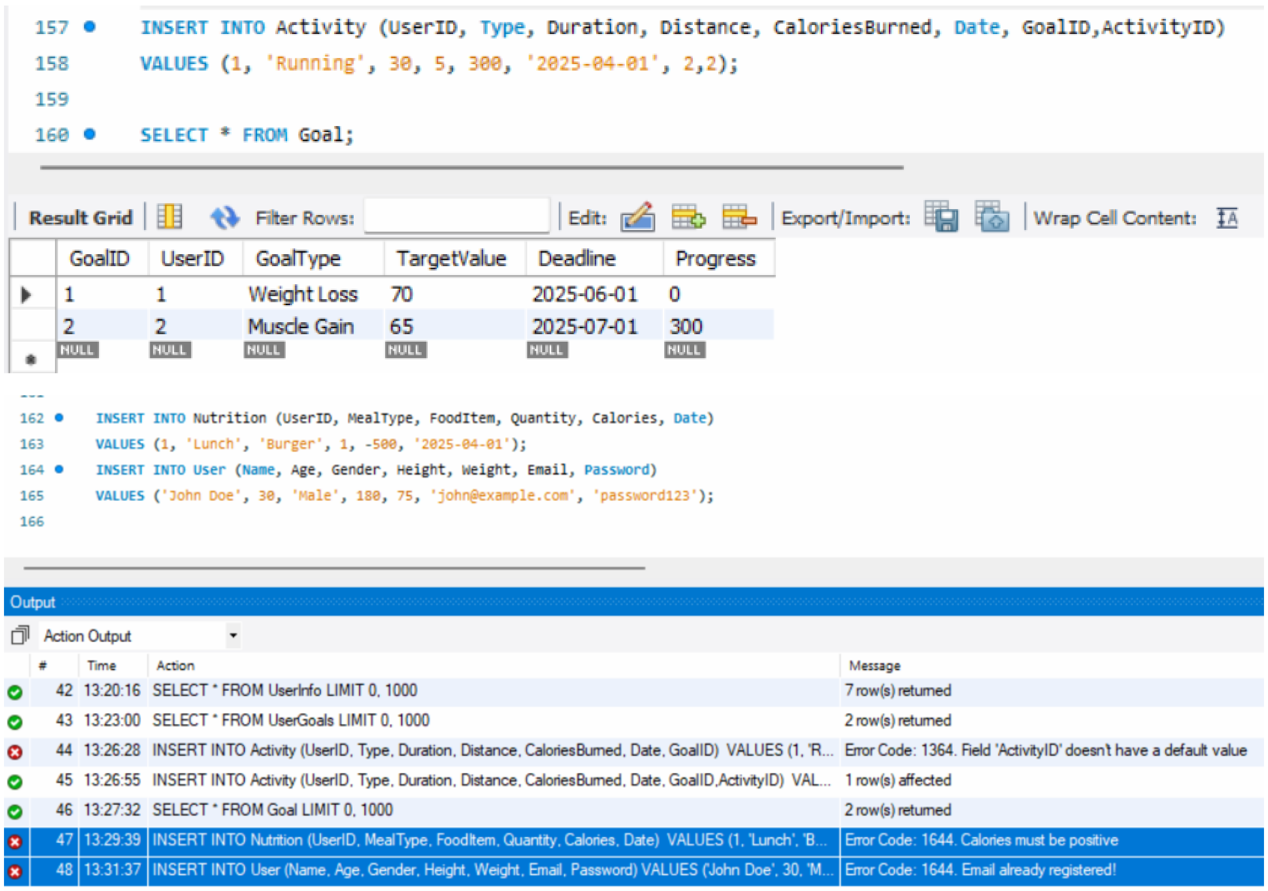
Name	GoalType	TargetValue	Deadline	Progress
he	Weight Loss	70	2025-06-01	0
bhoomi	Muscle Gain	65	2025-07-01	0

CREATING TRIGGERS



```sql
DELIMITER //
CREATE TRIGGER UpdateGoalProgress AFTER INSERT ON Activity
FOR EACH ROW
BEGIN
    UPDATE Goal
    SET Progress = Progress + NEW.CaloriesBurned
    WHERE GoalID = NEW.GoalID;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER ValidateNutrition BEFORE INSERT ON Nutrition
FOR EACH ROW
BEGIN
    IF NEW.Calories < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Calories must be positive';
    END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER PreventDuplicateEmail BEFORE INSERT ON User
FOR EACH ROW
BEGIN
    DECLARE email_count INT;
    SELECT COUNT(*) INTO email_count FROM User WHERE Email = NEW.Email;
    IF email_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Email already registered!';
    END IF;
END;
```

SAMPLE OUTPUT



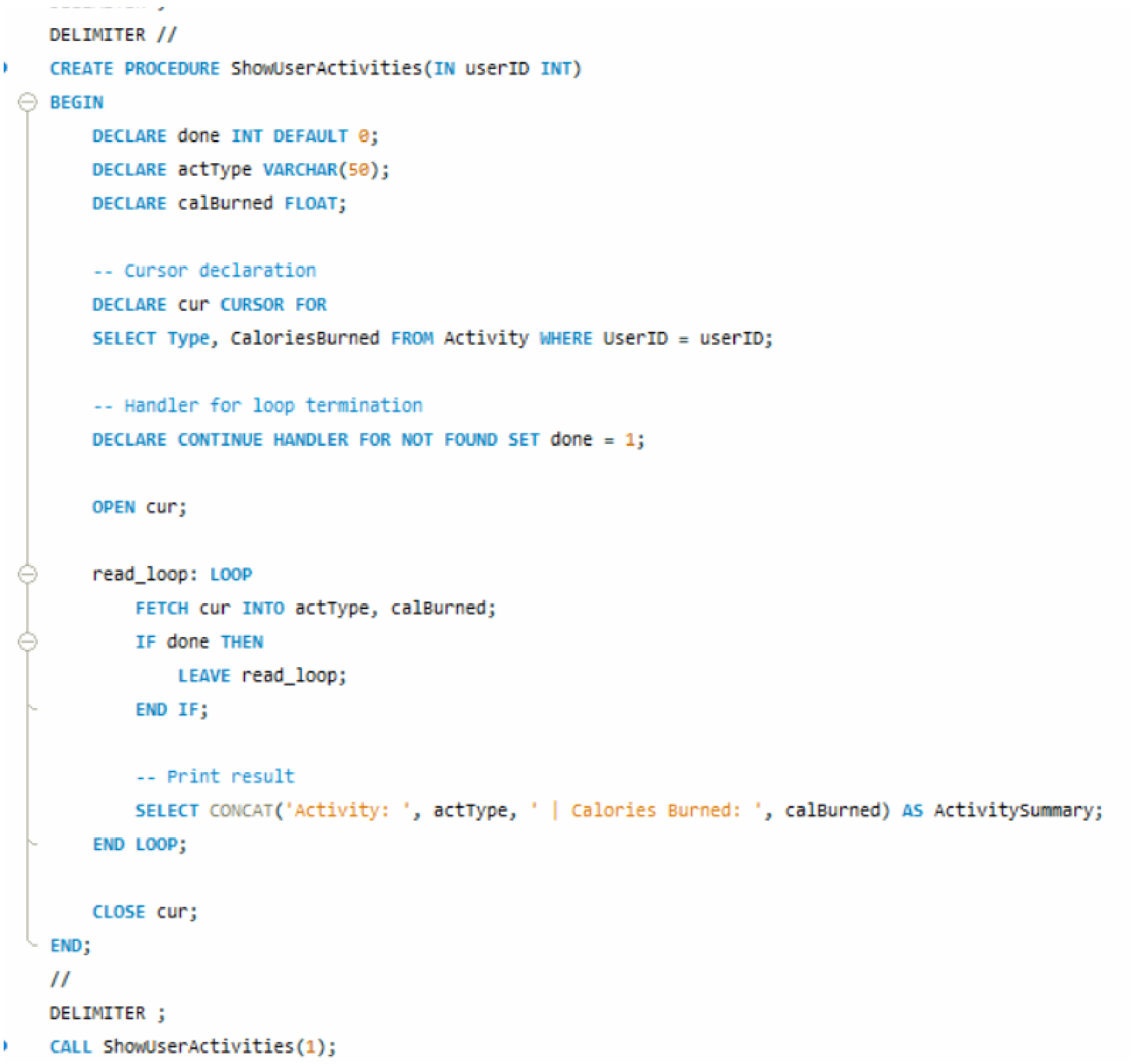
```sql
INSERT INTO Activity (UserID, Type, Duration, Distance, CaloriesBurned, Date, GoalID,ActivityID)
VALUES (1, 'Running', 30, 5, 300, '2025-04-01', 2,2);

SELECT * FROM Goal;
```

GoalID	UserID	GoalType	TargetValue	Deadline	Progress
1	1	Weight Loss	70	2025-06-01	0
2	2	Muscle Gain	65	2025-07-01	300
NULL	NULL	NULL	NULL	NULL	NULL

```sql
INSERT INTO Nutrition (UserID, MealType, FoodItem, Quantity, Calories, Date)
VALUES (1, 'Lunch', 'Burger', 1, -500, '2025-04-01');
INSERT INTO User (Name, Age, Gender, Height, Weight, Email, Password)
VALUES ('John Doe', 30, 'Male', 180, 75, 'john@example.com', 'password123');
```

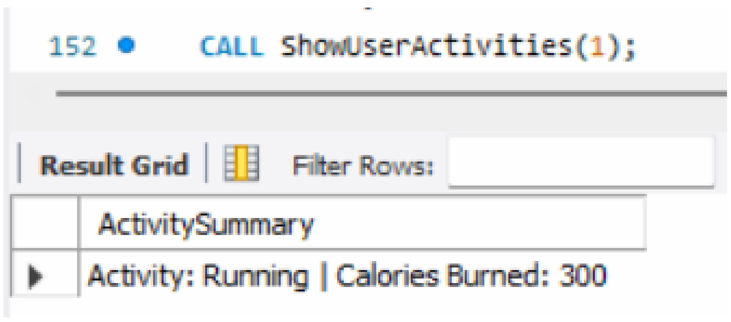
```
#   Time      Action                                                                                     Message
42  13:20:16  SELECT * FROM UserInfo LIMIT 0, 1000                                                       7 row(s) returned
43  13:23:00  SELECT * FROM UserGoals LIMIT 0, 1000                                                      2 row(s) returned
44  13:26:28  INSERT INTO Activity (UserID, Type, Duration, Distance, CaloriesBurned, Date, GoalID) VALUES (1, 'R...  Error Code: 1364. Field 'ActivityID' doesn't have a default value
45  13:26:55  INSERT INTO Activity (UserID, Type, Duration, Distance, CaloriesBurned, Date, GoalID,ActivityID) VAL...  1 row(s) affected
46  13:27:32  SELECT * FROM Goal LIMIT 0, 1000                                                           2 row(s) returned
47  13:29:39  INSERT INTO Nutrition (UserID, MealType, FoodItem, Quantity, Calories, Date) VALUES (1, 'Lunch', 'B...  Error Code: 1644. Calories must be positive
48  13:31:37  INSERT INTO User (Name, Age, Gender, Height, Weight, Email, Password) VALUES ('John Doe', 30, 'M...  Error Code: 1644. Email already registered!
```

CREATING PROCEDURES



```
DELIMITER //
CREATE PROCEDURE ShowUserActivities(IN userID INT)
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE actType VARCHAR(50);
    DECLARE calBurned FLOAT;

    -- Cursor declaration
    DECLARE cur CURSOR FOR
    SELECT Type, CaloriesBurned FROM Activity WHERE UserID = userID;

    -- Handler for loop termination
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO actType, calBurned;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Print result
        SELECT CONCAT('Activity: ', actType, ' | Calories Burned: ', calBurned) AS ActivitySummary;
    END LOOP;

    CLOSE cur;
END;
//
DELIMITER ;
CALL ShowUserActivities(1);
```

SAMPLE OUTPUT



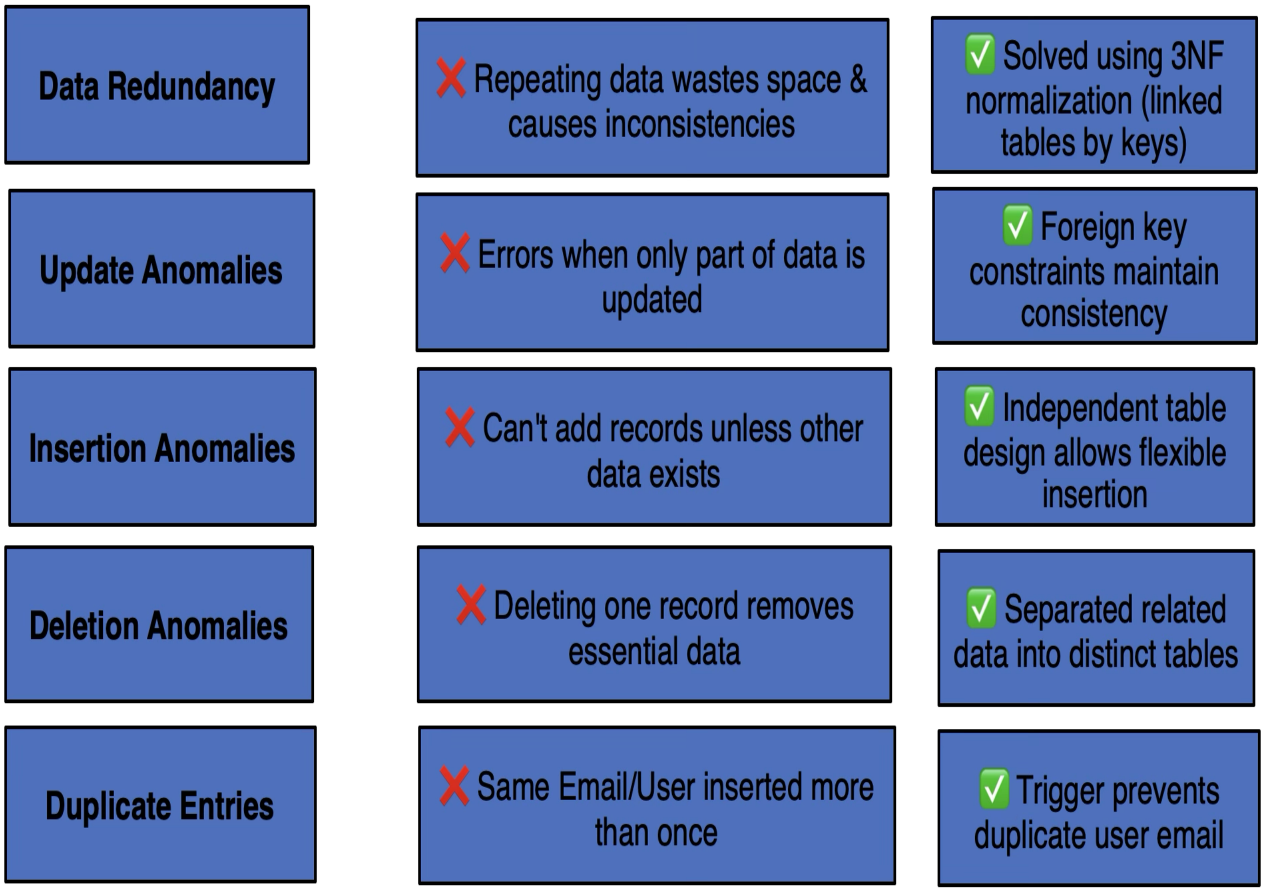
```
CALL ShowUserActivities(1);
```

Result Grid | Filter Rows:

```
ActivitySummary
Activity: Running | Calories Burned: 300
```

4. ANALYZING THE PITFALLS, IDENTIFYING THE DEPENDENCIES, AND APPLYING NORMALIZATIONS

PITFALL



Data Redundancy	❌ Repeating data wastes space & causes inconsistencies	✅ Solved using 3NF normalization (linked tables by keys)
Update Anomalies	❌ Errors when only part of data is updated	✅ Foreign key constraints maintain consistency
Insertion Anomalies	❌ Can't add records unless other data exists	✅ Independent table design allows flexible insertion
Deletion Anomalies	❌ Deleting one record removes essential data	✅ Separated related data into distinct tables
Duplicate Entries	❌ Same Email/User inserted more than once	✅ Trigger prevents duplicate user email

DEPENDENCY

1.FUNCTIONAL DEPENDENCY

When one column uniquely determines another column

EXAMPLE:

User Table :

UserID → Name, Age, Gender, Height, Weight, Email, Password

(Each attribute is uniquely identified by UserID)

Goal Table

GoalID → UserID, GoalType, TargetValue, Deadline, Progress

No transitive dependencies (e.g., Progress doesn’t depend on GoalType)

Activity Table

ActivityID → UserID, Type, Duration, Distance, CaloriesBurned, Date, GoalID

Fully functionally dependent on the primary key ActivityID

Nutrition Table

NutritionID → UserID, MealType, FoodItem, Quantity, Calories, Date

All attributes depend only on NutritionID, with no indirect dependency

2. TRANSITIVE DEPENDENCY

When one column depends indirectly on the primary key through another coloumn.

EXAMPLE:

If Progress can be calculated from activities, then:

GoalID → Progress (transitively through Activity table)

NORMALIZING

1NF ensures that:

Each field contains only atomic values (no lists or sets).
Each record (row) is unique.

Let's Check :-

User Table: Each column already holds atomic values (like Age, Gender, Height, etc.).
Activity Table: It also holds atomic values for Type, Duration, etc.
Same for Goal and Nutrition table.
Thus, all tables are already in 1NF.

To achieve 2NF, we need to:

Eliminate partial dependencies, i.e., make sure non-key attributes depend on the entire primary key, not just part of it.
This is particularly relevant when we have a composite primary key.

The UserID, Activity, Goal, Nutrition Table has no partial dependencies. Therefore, there's no need to change anything for 2NF. All tables meet the requirements of 2NF as well.

To achieve 3NF, we need to:

Remove transitive dependencies, i.e., non-key attributes should depend only on the primary key, not on other non-key attributes.

User Table: All non-key attributes (Name, Age, Gender, etc.) depend only on the primary key UserID. No transitive dependencies.
Goal Table: Attributes like GoalType, TargetValue, etc. depend directly on GoalID. No partial or transitive dependencies.
Activity Table: Each column depends solely on ActivityID. Fully normalized with no redundant data.
Nutrition Table: MealType, FoodItem, Calories, etc. depend on NutritionID. No derived or transitive attributes.

5. IMPLEMENTATION OF CONCURRENCY CONTROL AND RECOVERY MECHANISMS

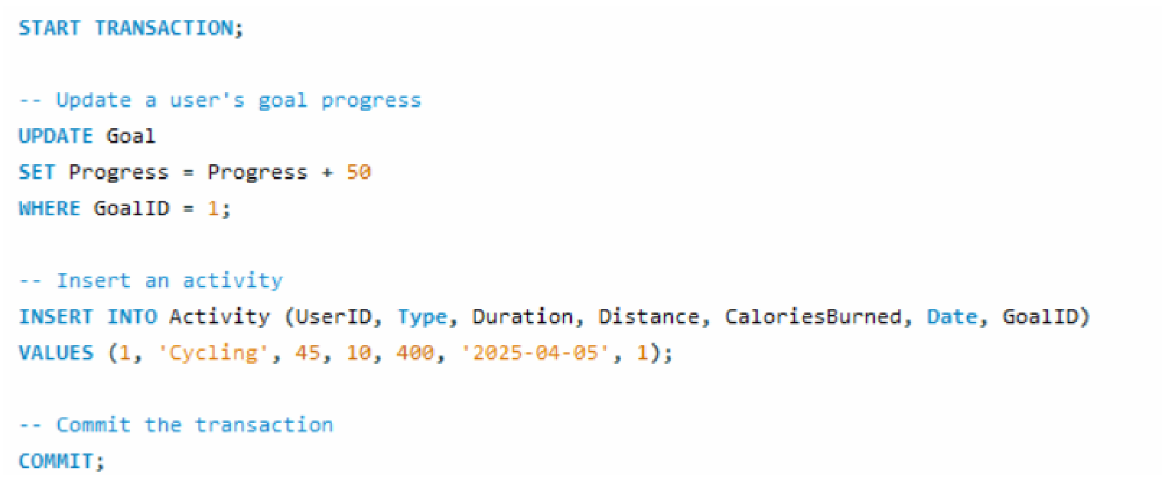
CONCURRENCY AND TRANSACTIONS:

You can use transactions to ensure that data is consistent and isolated, even when multiple users are performing actions at the same time.

Concurrency example:

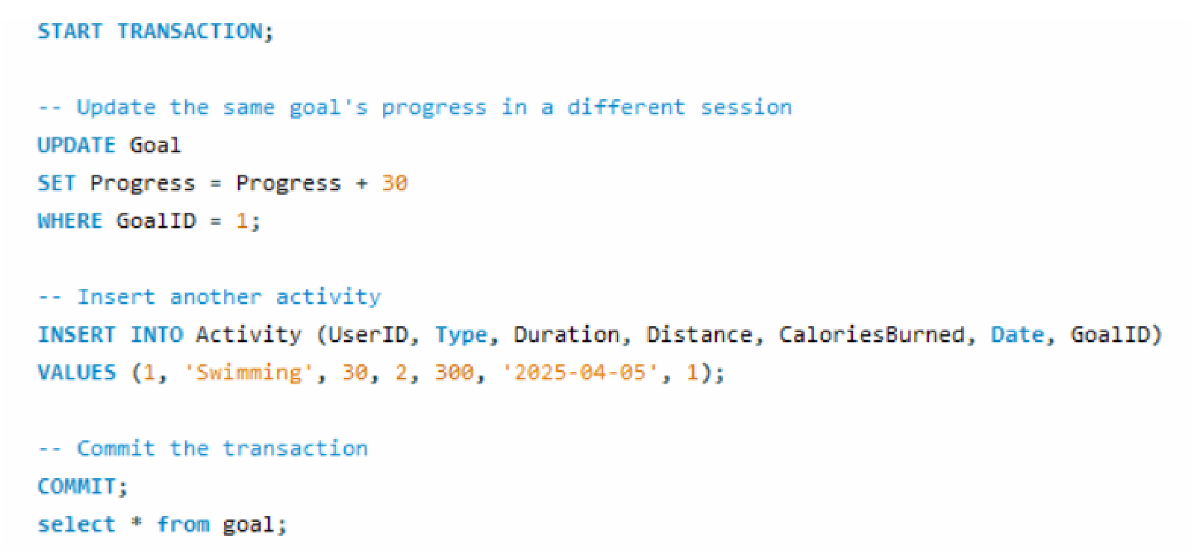
1. Opening two different sessions in mysql.
2. Performing actions in each session.

Transaction 1:



```sql
START TRANSACTION;

-- Update a user's goal progress
UPDATE Goal
SET Progress = Progress + 50
WHERE GoalID = 1;

-- Insert an activity
INSERT INTO Activity (UserID, Type, Duration, Distance, CaloriesBurned, Date, GoalID)
VALUES (1, 'Cycling', 45, 10, 400, '2025-04-05', 1);

-- Commit the transaction
COMMIT;
```

Transaction 2:



```sql
START TRANSACTION;

-- Update the same goal's progress in a different session
UPDATE Goal
SET Progress = Progress + 30
WHERE GoalID = 1;

-- Insert another activity
INSERT INTO Activity (UserID, Type, Duration, Distance, CaloriesBurned, Date, GoalID)
VALUES (1, 'Swimming', 30, 2, 300, '2025-04-05', 1);

-- Commit the transaction
COMMIT;
select * from goal;
```

Transaction Isolation Levels:

You can set the isolation level to control how the transactions interact with each other.

Example:

```
-- Set isolation level to SERIALIZABLE (Highest level of isolation)
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Handling Concurrency Issues:

- **Deadlock**: If the transactions try to access the same data in conflicting ways, a deadlock may occur. MySQL will automatically detect and resolve this by rolling back one of the transactions.

- **Lost Updates**: Ensure proper isolation to avoid situations where one transaction's changes overwrite another's updates unintentionally.

-
- For example, if both sessions simultaneously update the same goal's progress without locking the data, one session's changes might be overwritten by the other.

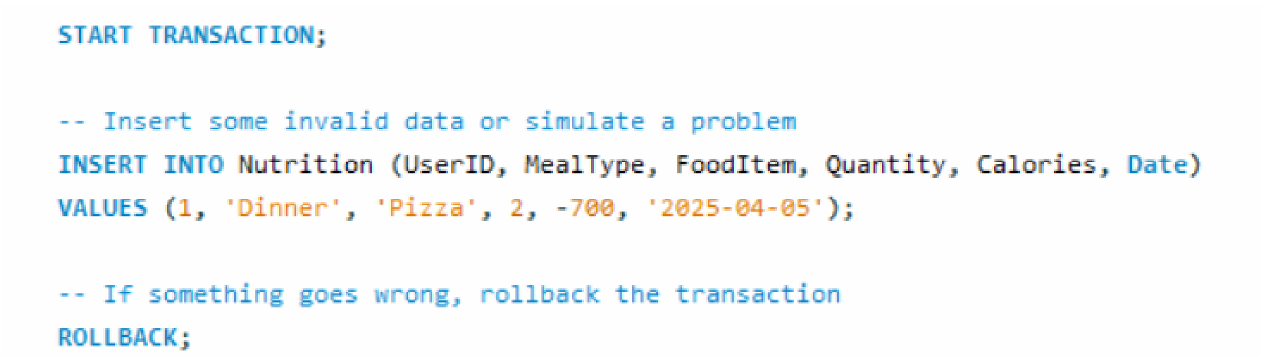
Locking Data:

To avoid conflicts between transactions, you can use locking mechanisms like SELECT FOR UPDATE to lock the row you are working on. This will prevent other transactions from modifying the same data until the current transaction is committed.

```
START TRANSACTION;

-- Lock the row for GoalID 1 to prevent other transactions from updating it simultaneously
SELECT * FROM Goal WHERE GoalID = 1 FOR UPDATE;

-- Now perform updates on Goal
UPDATE Goal
SET Progress = Progress + 50
WHERE GoalID = 1;

-- Commit the transaction
COMMIT;
```

Rollback Example:

If any errors occur or you want to revert the changes made in the current transaction, you can issue a ROLLBACK statement.



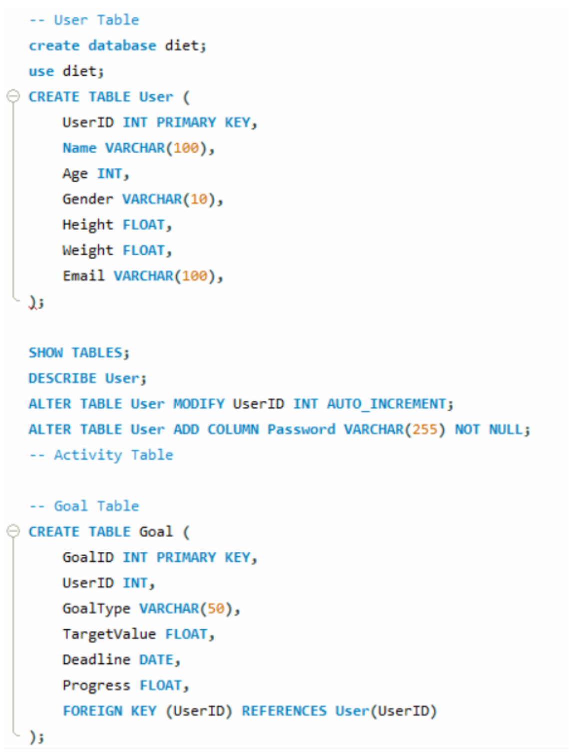
```sql
START TRANSACTION;

-- Insert some invalid data or simulate a problem
INSERT INTO Nutrition (UserID, MealType, FoodItem, Quantity, Calories, Date)
VALUES (1, 'Dinner', 'Pizza', 2, -700, '2025-04-05');

-- If something goes wrong, rollback the transaction
ROLLBACK;
```

6. CODE

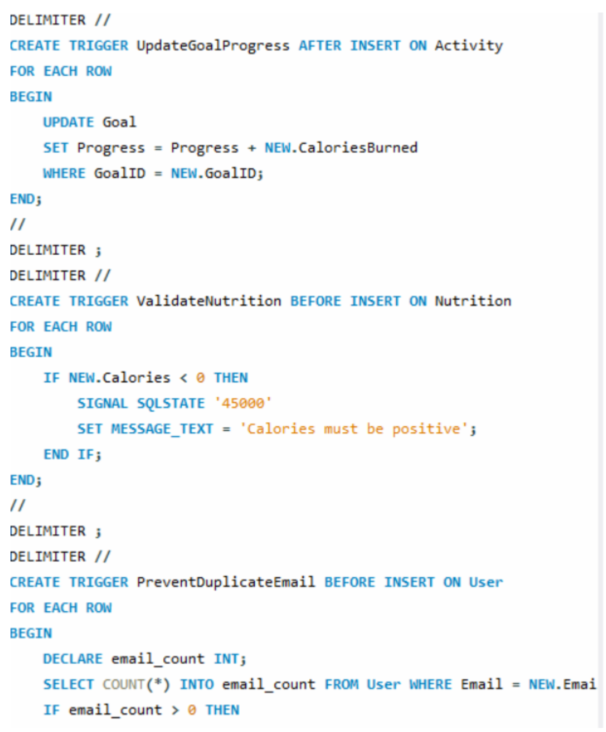
SAMPLE CODE:

```
package dietplanner;

  import javax.swing.*;
import java.awt.*;
import java.sql.*;

public class DietPlanner extends JFrame {

    private static final String DB_URL = "jdbc:mysql://localhost:3306/diet";
    private static final String DB_USER = "root";
    private static final String DB_PASSWORD = "root";

    private CardLayout cardLayout;
    private JPanel mainPanel;

    private int loggedInUserId = -1;
    private int loggedInGoalId = -1;

    public DietPlanner() {
        setTitle("Diet Planner");
        setSize(400, 300);
        setLocation(600, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        cardLayout = new CardLayout();
        mainPanel = new JPanel(cardLayout);

        mainPanel.add(createLoginPanel(), "Login");
        mainPanel.add(createRegisterPanel(), "Register");
        mainPanel.add(createGoalPanel(), "SetGoal");
        mainPanel.add(createDashboardPanel(), "Dashboard");

        add(mainPanel);
        cardLayout.show(mainPanel, "Login");

        setVisible(true);
    }

    private JPanel createLoginPanel() {
        JPanel panel = new JPanel(new GridLayout(4, 2, 10, 10));

        JLabel emailLabel = new JLabel("Email:");
        JTextField emailField = new JTextField();

        JLabel passwordLabel = new JLabel("Password:");
        JPasswordField passwordField = new JPasswordField();

        JButton loginButton = new JButton("Login");
        JButton registerButton = new JButton("Register");

        panel.add(emailLabel);
        panel.add(emailField);
        panel.add(passwordLabel);
        panel.add(passwordField);
        panel.add(loginButton);
        panel.add(registerButton);
        loginButton.addActionListener(e -> {
            String email = emailField.getText();
            String password = new String(passwordField.getPassword());
            int userId = loginUser(email, password);
            if (userId != -1) {
                loggedInUserId = userId;
                loggedInGoalId = fetchGoalId(userId);
                System.out.println("Login successful. UserID: " + loggedInUserId + ", GoalID: " + loggedInGoalId);
                JOptionPane.showMessageDialog(this, "Login successful!");
                cardLayout.show(mainPanel, "Dashboard");
            } else {
                JOptionPane.showMessageDialog(this, "Invalid email or password.");
            }
        });

        registerButton.addActionListener(e -> cardLayout.show(mainPanel, "Register"));

        return panel;
    }

    private JPanel createRegisterPanel() {
        JPanel panel = new JPanel(new GridLayout(8, 2, 10, 10));

        JLabel nameLabel = new JLabel("Name:");
        JTextField nameField = new JTextField();

        JLabel ageLabel = new JLabel("Age:");
        JTextField ageField = new JTextField();

        JLabel genderLabel = new JLabel("Gender:");
        String[] genders = {"Male", "Female", "Other"};
        JComboBox<String> genderBox = new JComboBox<>(genders);

        JLabel heightLabel = new JLabel("Height (cm):");
        JTextField heightField = new JTextField();

        JLabel weightLabel = new JLabel("Weight (kg):");
        JTextField weightField = new JTextField();

        JLabel emailLabel = new JLabel("Email:");
        JTextField emailField = new JTextField();

        JLabel passwordLabel = new JLabel("Password:");
        JPasswordField passwordField = new JPasswordField();

        JButton registerButton = new JButton("Register");
        JButton backButton = new JButton("Back");

        panel.add(nameLabel);
        panel.add(nameField);
        panel.add(ageLabel);
        panel.add(ageField);
        panel.add(genderLabel);
        panel.add(genderBox);
        panel.add(heightLabel);
        panel.add(heightField);
        panel.add(weightLabel);
        panel.add(weightField);
        panel.add(emailLabel);
        panel.add(emailField);
        panel.add(passwordLabel);
        panel.add(passwordField);
        panel.add(registerButton);
        panel.add(backButton);

        registerButton.addActionListener(e -> {
            try {
                String name = nameField.getText();
                String email = emailField.getText();
                String password = new String(passwordField.getPassword());
                int age = Integer.parseInt(ageField.getText());
                float height = Float.parseFloat(heightField.getText());
                float weight = Float.parseFloat(weightField.getText());
                String gender = (String) genderBox.getSelectedItem();

                if (registerUser(name, age, gender, height, weight, email, password)) {
                    loggedInUserId = getLastInsertedUserId();
                    System.out.println("Registered. UserID: " + loggedInUserId);
                    JOptionPane.showMessageDialog(this, "Registration successful! Now set your goal.");
                    cardLayout.show(mainPanel, "SetGoal");
                } else {
                    JOptionPane.showMessageDialog(this, "Registration failed!");
                }
            } catch (Exception ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(this, "Invalid input. Please fill all fields correctly.");
            }
        });

        backButton.addActionListener(e -> cardLayout.show(mainPanel, "Login"));

        return panel;
    }

    private JPanel createGoalPanel() {
        JPanel panel = new JPanel(new GridLayout(5, 2, 10, 10));

        JLabel goalLabel = new JLabel("Goal Type:");
        String[] goalOptions = {"Weight Loss", "Muscle Gain", "Maintain Fitness"};
        JComboBox<String> goalBox = new JComboBox<>(goalOptions);

        JLabel targetLabel = new JLabel("Target Weight (kg):");
        JTextField targetField = new JTextField();

        JLabel deadlineLabel = new JLabel("Deadline (YYYY-MM-DD):");
        JTextField deadlineField = new JTextField();

        JButton saveGoalButton = new JButton("Save Goal");

        panel.add(goalLabel);
        panel.add(goalBox);
        panel.add(targetLabel);
        panel.add(targetField);
        panel.add(deadlineLabel);
        panel.add(deadlineField);
        panel.add(new JLabel());
        panel.add(saveGoalButton);

        saveGoalButton.addActionListener(e -> {
            try {
                String goalType = (String) goalBox.getSelectedItem();
                float targetValue = Float.parseFloat(targetField.getText());
                String deadline = deadlineField.getText();
                if (saveUserGoal(goalType, targetValue, deadline)) {
                    loggedInGoalId = fetchGoalId(loggedInUserId);
                    System.out.println("Goal saved. GoalID: " + loggedInGoalId);
                    JOptionPane.showMessageDialog(this, "Goal saved successfully!");
                    cardLayout.show(mainPanel, "Dashboard");
                } else {
                    JOptionPane.showMessageDialog(this, "Failed to save goal!");
                }
            } catch (Exception ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(this, "Invalid goal input.");
            }
        });

        return panel;
    }

    private JPanel createDashboardPanel() {
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

        JLabel label = new JLabel("Welcome to your Dashboard!");
        JButton manageActivityButton = new JButton("Manage Activity");
        JButton manageNutritionButton = new JButton("Manage Nutrition");
        JButton viewProgressButton = new JButton("View Progress");

        panel.add(label);
        panel.add(manageActivityButton);
        panel.add(manageNutritionButton);
        panel.add(viewProgressButton);

        manageActivityButton.addActionListener(e -> manageActivity());
        manageNutritionButton.addActionListener(e -> manageNutrition());
        viewProgressButton.addActionListener(e -> viewProgress());

        return panel;
    }

    private boolean registerUser(String name, int age, String gender, float height, float weight, String email, String
password) {
        String query = "INSERT INTO User (Name, Age, Gender, Height, Weight, Email, Password) VALUES (?,
?, ?, ?, ?, ?, ?)";

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
             PreparedStatement stmt = conn.prepareStatement(query)) {

            stmt.setString(1, name);
            stmt.setInt(2, age);
            stmt.setString(3, gender);
            stmt.setFloat(4, height);
            stmt.setFloat(5, weight);
            stmt.setString(6, email);
            stmt.setString(7, password);

            return stmt.executeUpdate() > 0;

        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
    private int loginUser(String email, String password) {
        String query = "SELECT UserID FROM User WHERE Email = ? AND Password = ?";
        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
             PreparedStatement stmt = conn.prepareStatement(query)) {

            stmt.setString(1, email);
            stmt.setString(2, password);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                return rs.getInt("UserID");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return -1;
    }

    private boolean saveUserGoal(String goalType, float targetValue, String deadline) {
        String query = "INSERT INTO Goal (UserID, GoalType, TargetValue, Deadline, Progress) VALUES (?, ?,
?, ?, 0)";
        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
             PreparedStatement stmt = conn.prepareStatement(query)) {

            stmt.setInt(1, loggedInUserId);
            stmt.setString(2, goalType);
            stmt.setFloat(3, targetValue);
            stmt.setString(4, deadline);

            return stmt.executeUpdate() > 0;

        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    private int getLastInsertedUserId() {
        String query = "SELECT UserID FROM User ORDER BY UserID DESC LIMIT 1";
        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {

            if (rs.next()) {
                return rs.getInt("UserID");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return -1;
    }

    private int fetchGoalId(int userId) {
        String query = "SELECT GoalID FROM Goal WHERE UserID = ? ORDER BY GoalID DESC LIMIT 1";
        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
             PreparedStatement stmt = conn.prepareStatement(query)) {

            stmt.setInt(1, userId);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return rs.getInt("GoalID");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return -1;
    }

    private void manageActivity() {
        System.out.println("Attempting to open ActivityManager with UserID: " + loggedInUserId + ", GoalID: " +
loggedInGoalId);
        if (loggedInUserId != -1 && loggedInGoalId != -1) {
            new ActivityManager(loggedInUserId, loggedInGoalId);
            this.setVisible(false);
        } else {
            JOptionPane.showMessageDialog(this, "Missing user or goal ID!");
        }
    }

    private void manageNutrition() {
        if (loggedInUserId != -1) {
            new ManageNutrition(loggedInUserId);
            this.setVisible(false);
        } else {
            JOptionPane.showMessageDialog(this, "Missing user ID!");
        }
    }

    private void viewProgress() {
    if (loggedInUserId != -1) {
        new ViewProgress(loggedInUserId); // Open the ViewProgress window for the logged-in user
        this.setVisible(false);  // Hide the current window
    } else {
        JOptionPane.showMessageDialog(this, "No user logged in. Please log in first.");
    }
}


    public static void main(String[] args) {
        SwingUtilities.invokeLater(DietPlanner::new);
    }
}
```

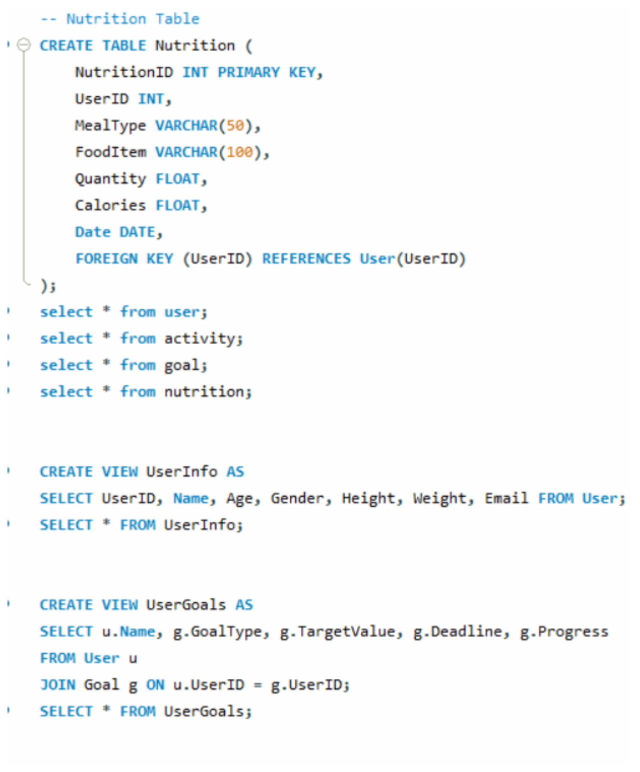
SQL QUERIES:



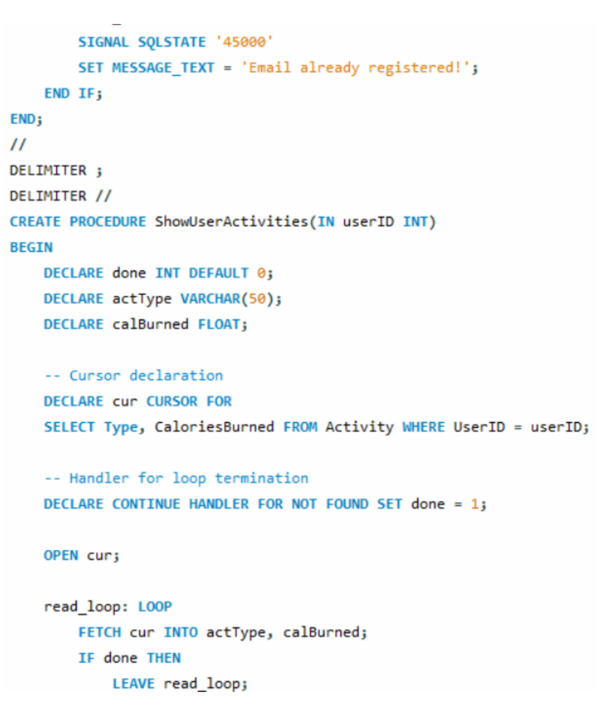
```sql
-- User Table
create database diet;
use diet;
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    Height FLOAT,
    Weight FLOAT,
    Email VARCHAR(100),
);

SHOW TABLES;
DESCRIBE User;
ALTER TABLE User MODIFY UserID INT AUTO_INCREMENT;
ALTER TABLE User ADD COLUMN Password VARCHAR(255) NOT NULL;
-- Activity Table

-- Goal Table
CREATE TABLE Goal (
    GoalID INT PRIMARY KEY,
    UserID INT,
    GoalType VARCHAR(50),
    TargetValue FLOAT,
    Deadline DATE,
    Progress FLOAT,
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```



```sql
DELIMITER //
CREATE TRIGGER UpdateGoalProgress AFTER INSERT ON Activity
FOR EACH ROW
BEGIN
    UPDATE Goal
    SET Progress = Progress + NEW.CaloriesBurned
    WHERE GoalID = NEW.GoalID;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER ValidateNutrition BEFORE INSERT ON Nutrition
FOR EACH ROW
BEGIN
    IF NEW.Calories < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Calories must be positive';
    END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE TRIGGER PreventDuplicateEmail BEFORE INSERT ON User
FOR EACH ROW
BEGIN
    DECLARE email_count INT;
    SELECT COUNT(*) INTO email_count FROM User WHERE Email = NEW.Emai
    IF email_count > 0 THEN
```



```sql
-- Nutrition Table
CREATE TABLE Nutrition (
    NutritionID INT PRIMARY KEY,
    UserID INT,
    MealType VARCHAR(50),
    FoodItem VARCHAR(100),
    Quantity FLOAT,
    Calories FLOAT,
    Date DATE,
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
select * from user;
select * from activity;
select * from goal;
select * from nutrition;


CREATE VIEW UserInfo AS
SELECT UserID, Name, Age, Gender, Height, Weight, Email FROM User;
SELECT * FROM UserInfo;


CREATE VIEW UserGoals AS
SELECT u.Name, g.GoalType, g.TargetValue, g.Deadline, g.Progress
FROM User u
JOIN Goal g ON u.UserID = g.UserID;
SELECT * FROM UserGoals;
```

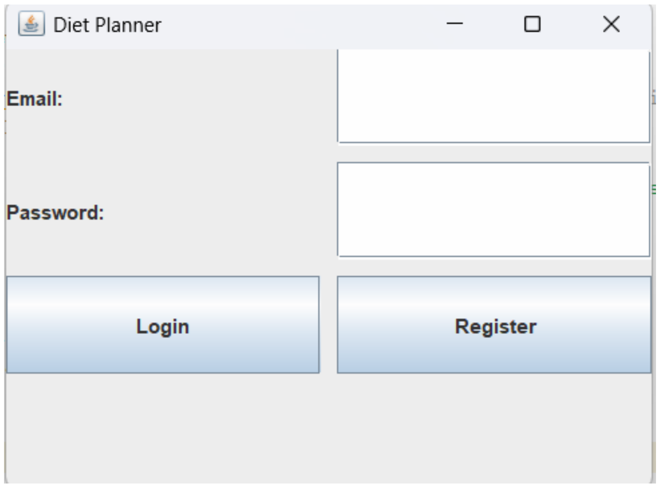


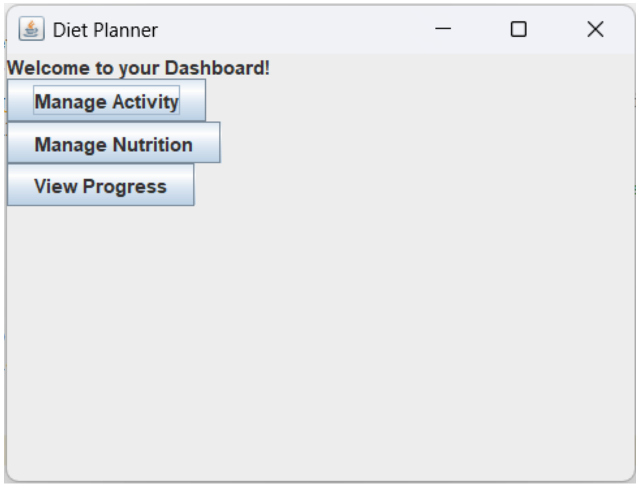
```sql
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Email already registered!';
    END IF;
END;
//
DELIMITER ;
DELIMITER //
CREATE PROCEDURE ShowUserActivities(IN userID INT)
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE actType VARCHAR(50);
    DECLARE calBurned FLOAT;

    -- Cursor declaration
    DECLARE cur CURSOR FOR
    SELECT Type, CaloriesBurned FROM Activity WHERE UserID = userID;

    -- Handler for loop termination
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO actType, calBurned;
        IF done THEN
            LEAVE read_loop;
```

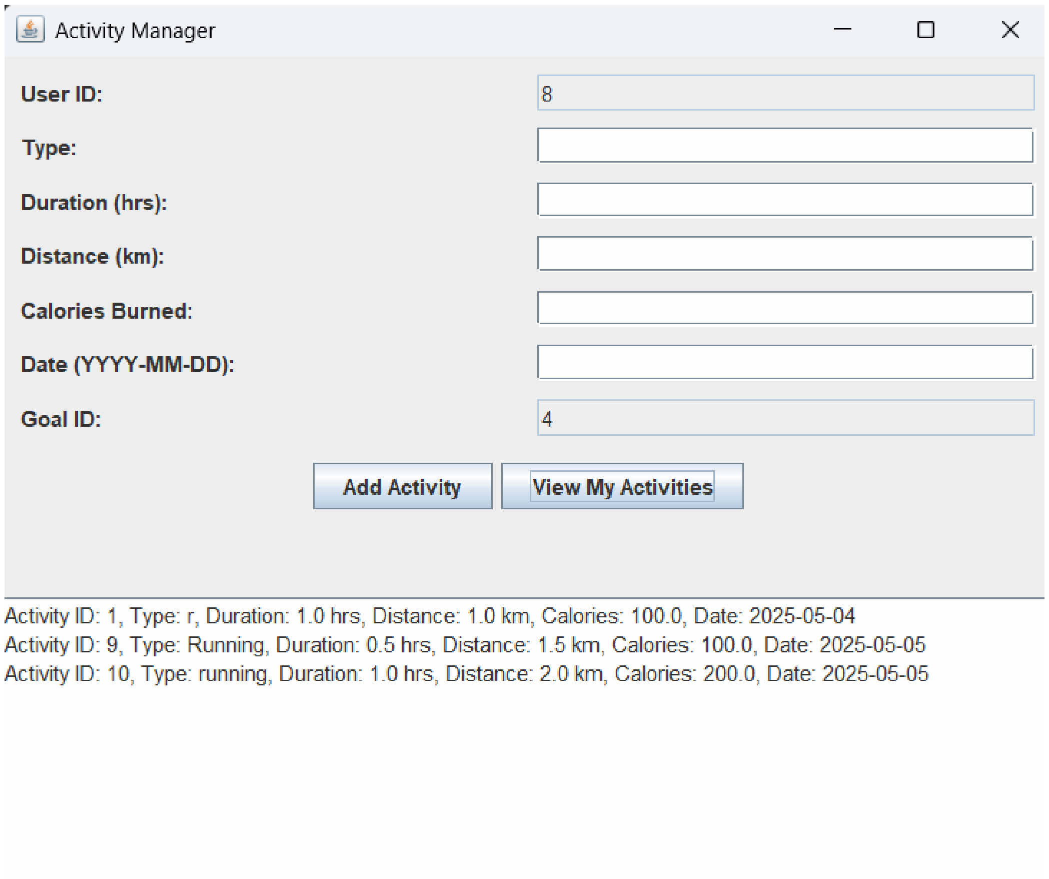
```sql
DELIMITER //
select * from goal;
ALTER TABLE Nutrition MODIFY NutritionID INT AUTO_INCREMENT;

START TRANSACTION;

-- Update a user's goal progress
UPDATE Goal
SET Progress = Progress + 50
WHERE GoalID = 1;

-- Insert an activity
INSERT INTO Activity (UserID, Type, Duration, Distance, CaloriesBurned, Date, GoalID)
VALUES (1, 'Cycling', 45, 10, 400, '2025-04-05', 1);

-- Commit the transaction
COMMIT;
START TRANSACTION;

-- Update the same goal's progress in a different session
UPDATE Goal
SET Progress = Progress + 30
WHERE GoalID = 1;
```

7. RESULT AND DISCUSSION

RESULT

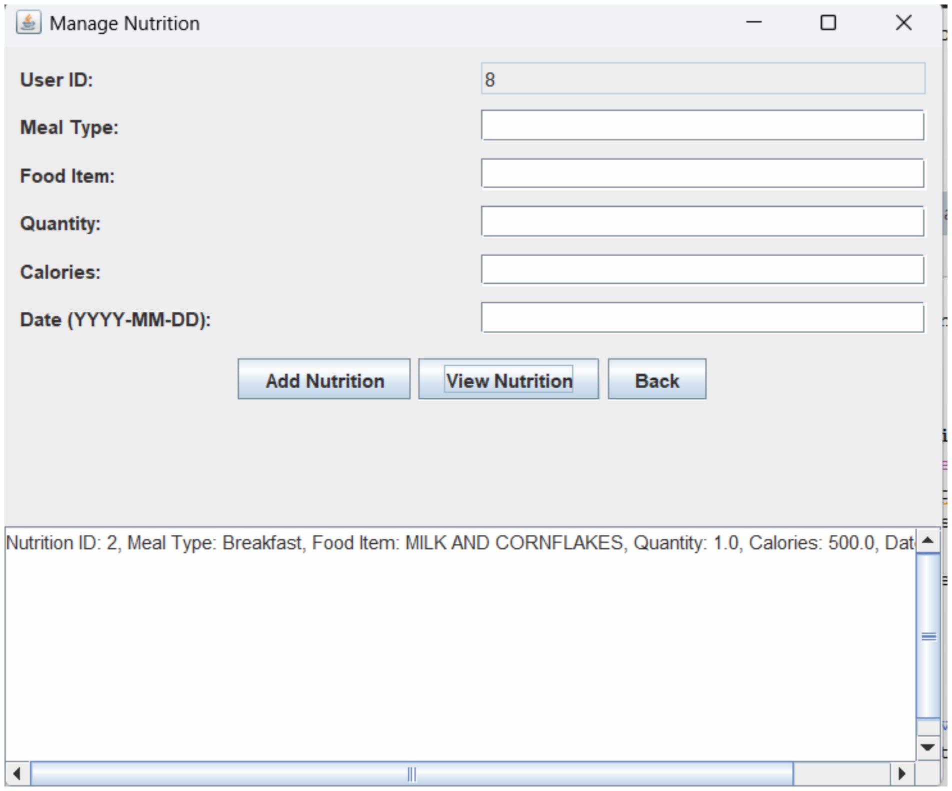






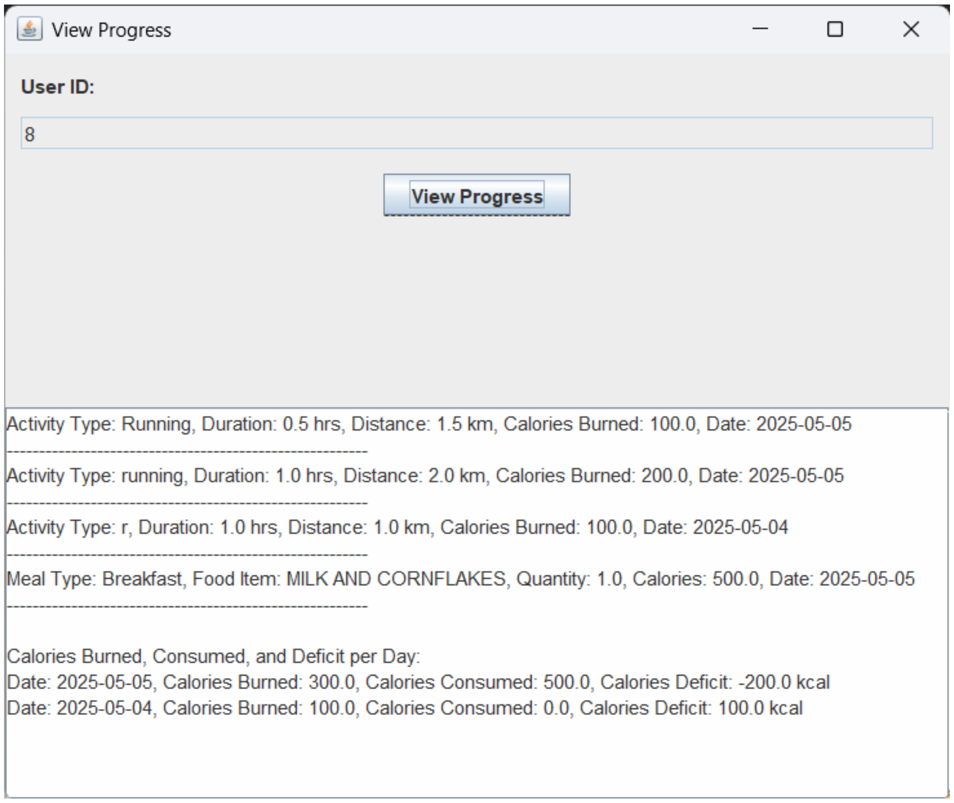
User ID: 8
Type:
Duration (hrs):
Distance (km):
Calories Burned:
Date (YYYY-MM-DD):
Goal ID: 4

```
Activity ID: 1, Type: r, Duration: 1.0 hrs, Distance: 1.0 km, Calories: 100.0, Date: 2025-05-04
Activity ID: 9, Type: Running, Duration: 0.5 hrs, Distance: 1.5 km, Calories: 100.0, Date: 2025-05-05
Activity ID: 10, Type: running, Duration: 1.0 hrs, Distance: 2.0 km, Calories: 200.0, Date: 2025-05-05
```



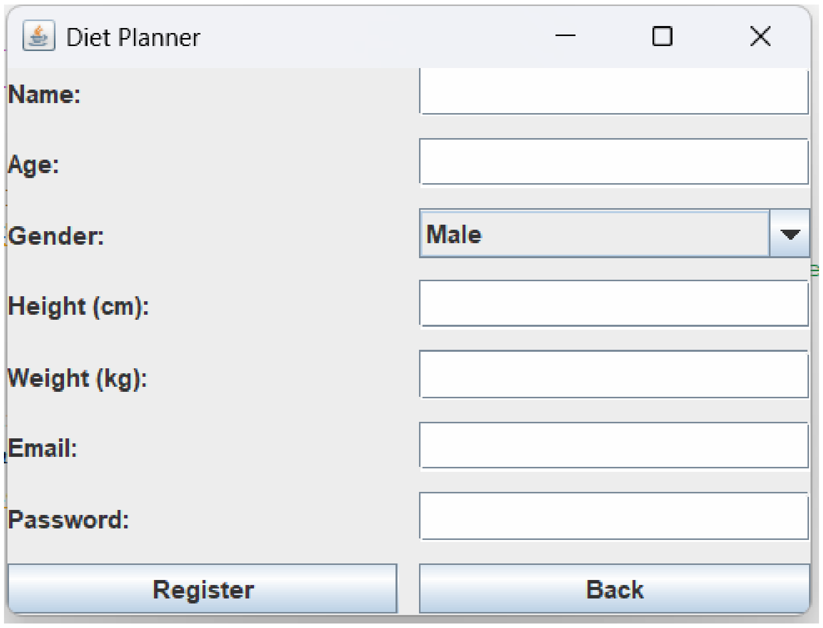
User ID: 8
Meal Type:
Food Item:
Quantity:
Calories:
Date (YYYY-MM-DD):

```
Nutrition ID: 2, Meal Type: Breakfast, Food Item: MILK AND CORNFLAKES, Quantity: 1.0, Calories: 500.0, Dat
```



User ID: 8

```
Activity Type: Running, Duration: 0.5 hrs, Distance: 1.5 km, Calories Burned: 100.0, Date: 2025-05-05
--------------------------------------------------
Activity Type: running, Duration: 1.0 hrs, Distance: 2.0 km, Calories Burned: 200.0, Date: 2025-05-05
--------------------------------------------------
Activity Type: r, Duration: 1.0 hrs, Distance: 1.0 km, Calories Burned: 100.0, Date: 2025-05-04
--------------------------------------------------
Meal Type: Breakfast, Food Item: MILK AND CORNFLAKES, Quantity: 1.0, Calories: 500.0, Date: 2025-05-05
--------------------------------------------------

Calories Burned, Consumed, and Deficit per Day:
Date: 2025-05-05, Calories Burned: 300.0, Calories Consumed: 500.0, Calories Deficit: -200.0 kcal
Date: 2025-05-04, Calories Burned: 100.0, Calories Consumed: 0.0, Calories Deficit: 100.0 kcal
```


Diet Planner

Name:	
Age:	
Gender:	Male
Height (cm):	
Weight (kg):	
Email:	
Password:	

DISCUSSION

The implementation and testing of the *Diet Planner* application provided insightful results that reflect both the effectiveness and usability of the system. During the testing phase, users were able to:

Successfully log daily food intake and physical activities such as walking, running, or workouts.

Retrieve and view detailed progress reports that include:

- Total calories consumed in a day.
- Total calories burned in a day.
- The net calorie deficit (or surplus), calculated accurately.

The results clearly showed that the application could differentiate between days with a healthy balance and days where caloric goals were not met. This real-time feedback proved helpful for users aiming to lose weight, maintain their health, or track progress over time.

The integration of calorie tracking across nutrition and activity modules enabled more meaningful insights. For instance, users could correlate high-calorie meals with increased physical activity requirements or identify days where they exceeded or fell short of their targets.

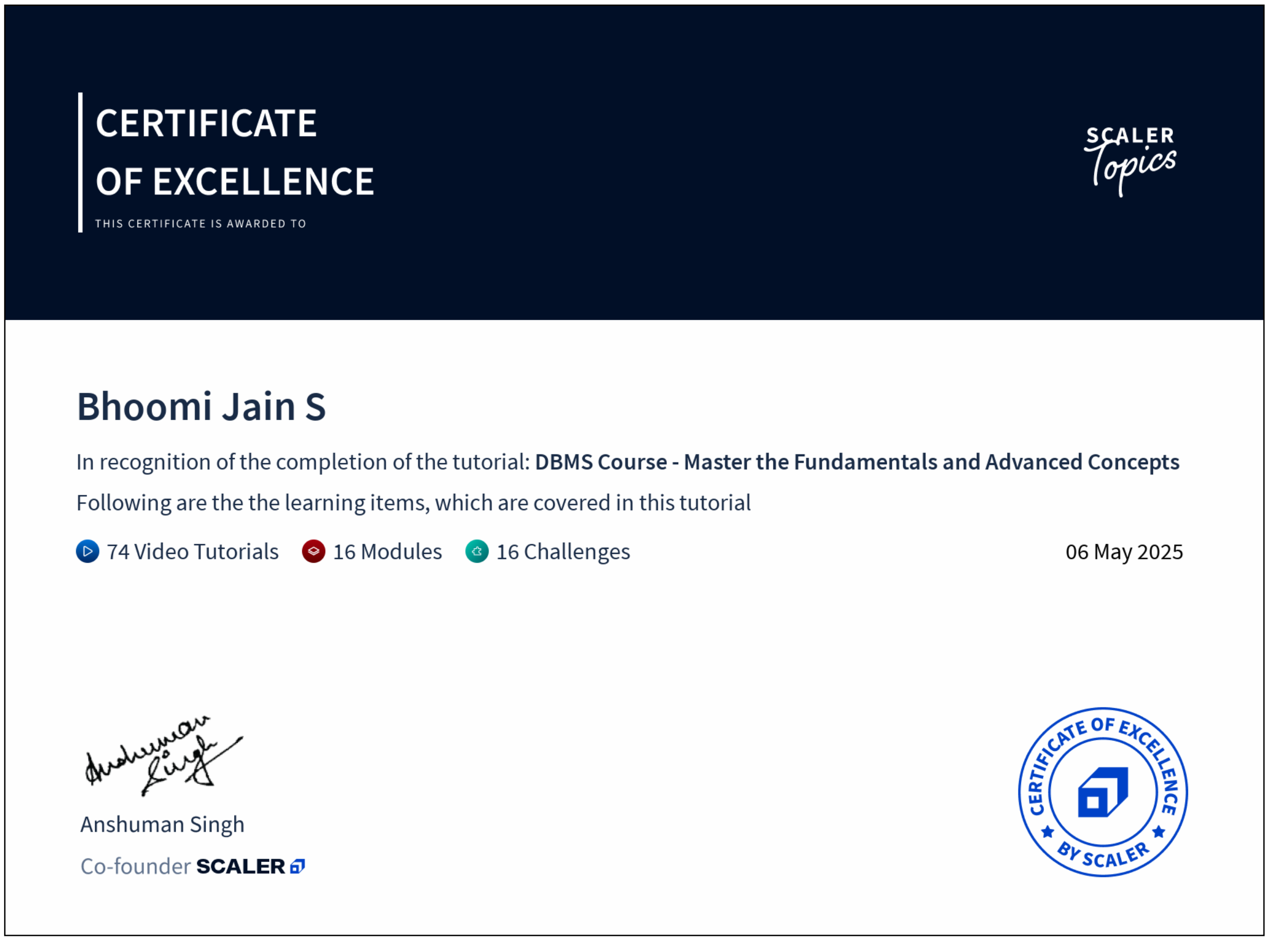
Overall, the project fulfilled its objectives, and the results validate that the application can serve as a useful personal health assistant. However, future iterations could enhance the accuracy by integrating with wearable fitness devices or offering personalized calorie goals based on user profile data (like age, weight, and fitness goal).

8. CONCLUSION

The *Diet Planner* project successfully integrates key functionalities to help users monitor and manage their diet and physical activities effectively. By allowing users to log daily nutrition intake and physical activity, the system provides a clear overview of their health journey. One of the most impactful features is the ability to view daily progress, including calories consumed, calories burned, and the resulting calorie deficit—enabling users to make informed decisions about their lifestyle habits.

Through the use of Java for the user interface and MySQL for data storage, this project demonstrates the practical implementation of object-oriented programming and database management. It showcases how software applications can be built to serve real-world health and fitness needs, with a strong focus on user experience, data accuracy, and meaningful feedback.

Overall, the Diet Planner offers a valuable tool for users aiming to maintain or improve their physical well-being, and lays a solid foundation for further development, such as goal-setting, reminders, and personalized recommendations.



CERTIFICATE
OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

Bhoomi Jain S

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

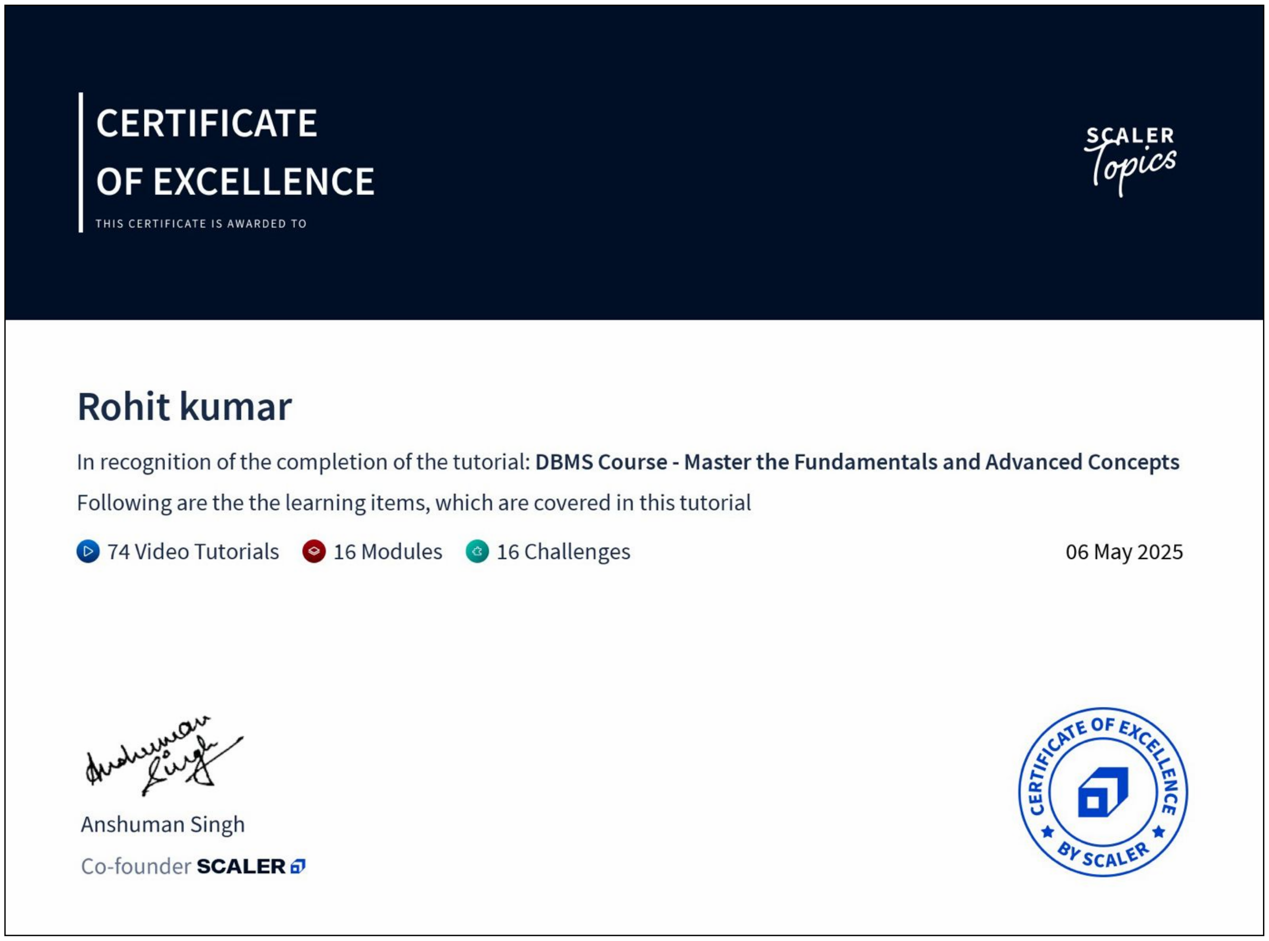
74 Video Tutorials 16 Modules 16 Challenges 06 May 2025

[illegible]

Anshuman Singh

Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE ★ BY SCALER ★



CERTIFICATE
OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

Rohit kumar

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges 06 May 2025

[illegible]

Anshuman Singh

Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE ★ BY SCALER ★