WORKOUT SPLIT MANAGER

Project Report

Submitted by:

Vihaan Dumont – 3761518
Darin Thomson – 3776723
Abdelrahman Abdelsadek – 3764220
Dwyane Luy – 3775996

Abstract

Workout Split Manager is a Java-based project designed to automate and personalize workout routines for gym users. This project demonstrates how relational database design, Java programming, and JavaFX GUI integration can come together to simulate a real-world gym management system. Each user is assigned a 5-day routine based on their fitness goals and membership plans, showcasing the practical implementation of database normalization, ER modelling, and randomized data generation.

Introduction

The Workout Split Manager aims to bring structure and personalization to fitness routines using a desktop-based Java application. Users are assigned routines tailored to their fitness goals, such as Muscle Building, Weight Loss, Endurance, or General Fitness. Utilizing SQLite as the backend and JavaFX for the user interface, the system creates a seamless and interactive environment for managing exercises, users, and workout plans.

System Description

The application comprises five interrelated tables: Users, MuscleGroup, Exercise, Workouts, and WorkoutExercise. Using JDBC, these tables are connected and populated dynamically. The data includes 15 users with randomized 5-day workout routines incorporating various exercises, sets, reps, and durations. The JavaFX GUI allows users to interact with the data using table views and buttons for navigation. The GUI includes scenes for viewing all users, exercises, and workouts, with specific plans dynamically generated for each user.

ER Diagram Overview

The ER design follows a normalized structure, where foreign key constraints enforce data integrity between tables. WorkoutExercise handles many-to-many relationships between Workouts and Exercise. Each Exercise belongs to a specific MuscleGroup, enabling structured categorization. Workouts are associated with Users via UserID foreign keys, and each user is assigned multiple workouts over five days.

List of DDL Statements

CREATE TABLE Users (...);

```
CREATE TABLE IF NOT EXISTS Users (
    UserID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    Email TEXT NOT NULL,
    PhoneNumber TEXT NOT NULL,
    workout_goal TEXT,
    membership_plan TEXT
);
```

CREATE TABLE MuscleGroup (…);

```
CREATE TABLE IF NOT EXISTS MuscleGroup (
    MuscleGroupID INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL
);
```

CREATE TABLE Exercise (…);

```
CREATE TABLE IF NOT EXISTS Exercise (
    ExerciseID INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    muscle_group_id INTEGER,
    FOREIGN KEY (muscle_group_id) REFERENCES MuscleGroup(MuscleGroupID)
);
```

CREATE TABLE Workouts (…);

```
CREATE TABLE IF NOT EXISTS Workouts (
    WorkoutID INTEGER PRIMARY KEY AUTOINCREMENT,
    UserID INTEGER,
    DayNumber INTEGER,
    WorkoutDay TEXT,
    ExerciseName TEXT,
    Sets INTEGER,
    Reps INTEGER,
    Duration INTEGER,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

CREATE TABLE WorkoutExercise (…);

```
CREATE TABLE IF NOT EXISTS WorkoutExercise (
    WorkoutExerciseID INTEGER PRIMARY KEY AUTOINCREMENT,
    WorkoutID INTEGER,
    ExerciseID INTEGER,
    Sets INTEGER NOT NULL,
    Reps INTEGER NOT NULL,
    Duration INTEGER NOT NULL,
    FOREIGN KEY (WorkoutID) REFERENCES Workouts(WorkoutID),
    FOREIGN KEY (ExerciseID) REFERENCES Exercise(ExerciseID)
);
```

Java Source Code Summary

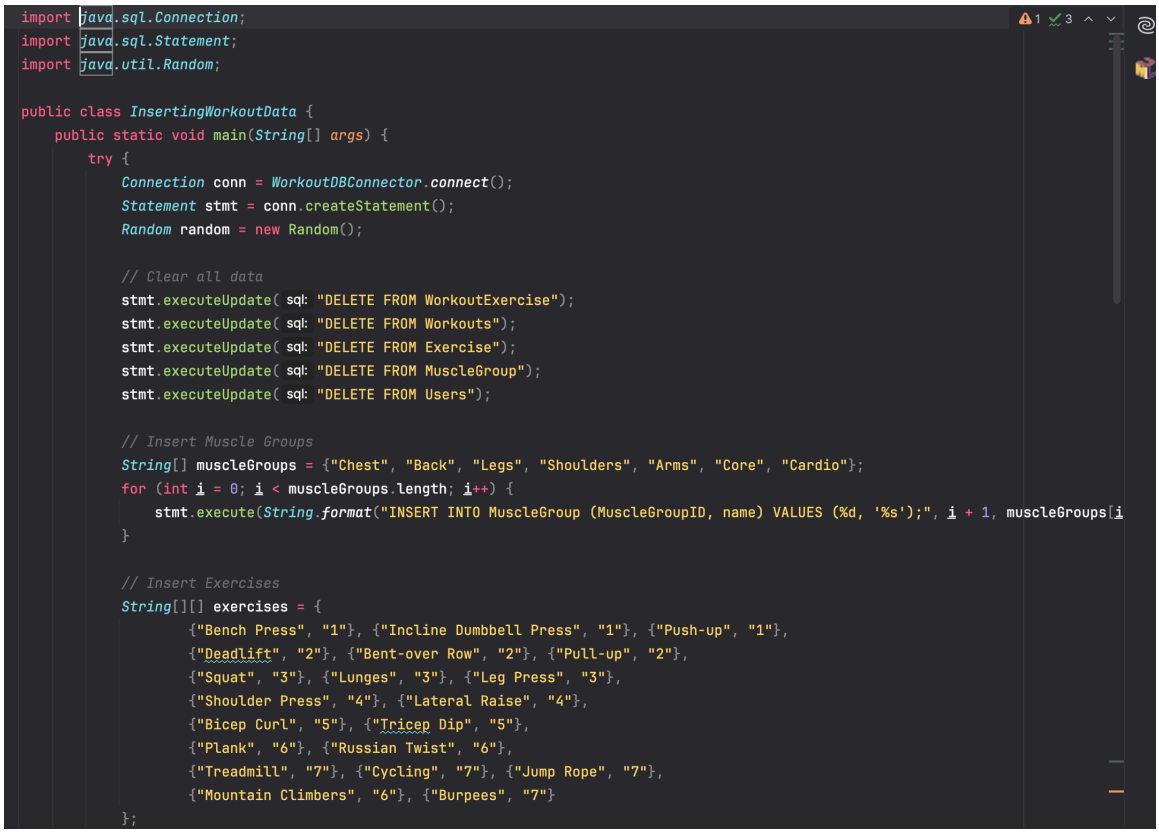
1. CreatingWorkoutTables.java – Defines schema for all workout-related tables.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class CreatingWorkoutTables {
    public static void main(String[] args) {
        try {
            Connection conn = DriverManager.getConnection( url: "jdbc:sqlite:workout_split.db");


            Statement stmt = conn.createStatement();

            // Users Table
            stmt.execute( sql: """
                CREATE TABLE IF NOT EXISTS Users (
                    UserID INTEGER PRIMARY KEY AUTOINCREMENT,
                    Name TEXT NOT NULL,
                    Email TEXT NOT NULL,
                    PhoneNumber TEXT NOT NULL,
                    workout_goal TEXT,
                    membership_plan TEXT
                );
            """);

            // MuscleGroup Table
            stmt.execute( sql: """
                CREATE TABLE IF NOT EXISTS MuscleGroup (
                    MuscleGroupID INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL
                );
            """);
```

```
    // Exercise Table
    stmt.execute( sql: """
        CREATE TABLE IF NOT EXISTS Exercise (
            ExerciseID INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            muscle_group_id INTEGER,
            FOREIGN KEY (muscle_group_id) REFERENCES MuscleGroup(MuscleGroupID)
        );
    """);

    // Workouts Table
    stmt.execute( sql: """
        CREATE TABLE IF NOT EXISTS Workouts (
            WorkoutID INTEGER PRIMARY KEY AUTOINCREMENT,
            UserID INTEGER,
            DayNumber INTEGER,
            WorkoutDay TEXT,
            ExerciseName TEXT,
            Sets INTEGER,
            Reps INTEGER,
            Duration INTEGER,
            FOREIGN KEY (UserID) REFERENCES Users(UserID)
        );
    """);
```

```
            // WorkoutExercise Table
            stmt.execute( sql: """
                CREATE TABLE IF NOT EXISTS WorkoutExercise (
                    WorkoutExerciseID INTEGER PRIMARY KEY AUTOINCREMENT,
                    WorkoutID INTEGER,
                    ExerciseID INTEGER,
                    Sets INTEGER NOT NULL,
                    Reps INTEGER NOT NULL,
                    Duration INTEGER NOT NULL,
                    FOREIGN KEY (WorkoutID) REFERENCES Workouts(WorkoutID),
                    FOREIGN KEY (ExerciseID) REFERENCES Exercise(ExerciseID)
                );
            """);

            System.out.println("Workout tables created successfully.");
            conn.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2. InsertingWorkoutData.java – Populates the database with randomized data for users and exercises.



```java
import java.sql.Connection;
import java.sql.Statement;
import java.util.Random;

public class InsertingWorkoutData {
    public static void main(String[] args) {
        try {
            Connection conn = WorkoutDBConnector.connect();
            Statement stmt = conn.createStatement();
            Random random = new Random();

            // Clear all data
            stmt.executeUpdate( sql: "DELETE FROM WorkoutExercise");
            stmt.executeUpdate( sql: "DELETE FROM Workouts");
            stmt.executeUpdate( sql: "DELETE FROM Exercise");
            stmt.executeUpdate( sql: "DELETE FROM MuscleGroup");
            stmt.executeUpdate( sql: "DELETE FROM Users");

            // Insert Muscle Groups
            String[] muscleGroups = {"Chest", "Back", "Legs", "Shoulders", "Arms", "Core", "Cardio"};
            for (int i = 0; i < muscleGroups.length; i++) {
                stmt.execute(String.format("INSERT INTO MuscleGroup (MuscleGroupID, name) VALUES (%d, '%s');", i + 1, muscleGroups[i
            }

            // Insert Exercises
            String[][] exercises = {
                    {"Bench Press", "1"}, {"Incline Dumbbell Press", "1"}, {"Push-up", "1"},
                    {"Deadlift", "2"}, {"Bent-over Row", "2"}, {"Pull-up", "2"},
                    {"Squat", "3"}, {"Lunges", "3"}, {"Leg Press", "3"},
                    {"Shoulder Press", "4"}, {"Lateral Raise", "4"},
                    {"Bicep Curl", "5"}, {"Tricep Dip", "5"},
                    {"Plank", "6"}, {"Russian Twist", "6"},
                    {"Treadmill", "7"}, {"Cycling", "7"}, {"Jump Rope", "7"},
                    {"Mountain Climbers", "6"}, {"Burpees", "7"}
            };
```

```java
    for (int i = 0; i < exercises.length; i++) {
        stmt.execute(String.format("INSERT INTO Exercise (ExerciseID, name, muscle_group_id) VALUES (%d, '%s', %s);", i + 1,
    }

    // Insert Users
    String[] names = {"Vihaan", "John", "Alice", "Mark", "Emma", "David", "Sophia", "Daniel", "Chloe", "Ethan", "Liam", "Ava
    for (int i = 0; i < names.length; i++) {
        String email = names[i].toLowerCase() + "@example.com";
        String phone = "555-01" + String.format("%03d", i + 1);
        String goal = switch (i % 4) {
            case 0 -> "Muscle Building";
            case 1 -> "Weight Loss";
            case 2 -> "Endurance";
            default -> "General Fitness";
        };
        String plan = (i % 2 == 0) ? "Premium" : "Basic";
        stmt.execute(String.format(
                "INSERT INTO Users (UserID, Name, Email, PhoneNumber, workout_goal, membership_plan) " +
                        "VALUES (%d, '%s', '%s', '%s', '%s', '%s');",
                i + 1, names[i], email, phone, goal, plan));
    }

    // Workout Day names
    String[] workoutDays = {"Push Day", "Pull Day", "Leg Day", "Cardio", "Full Body"};
    int workoutID = 1;
    int workoutExerciseID = 1;
```

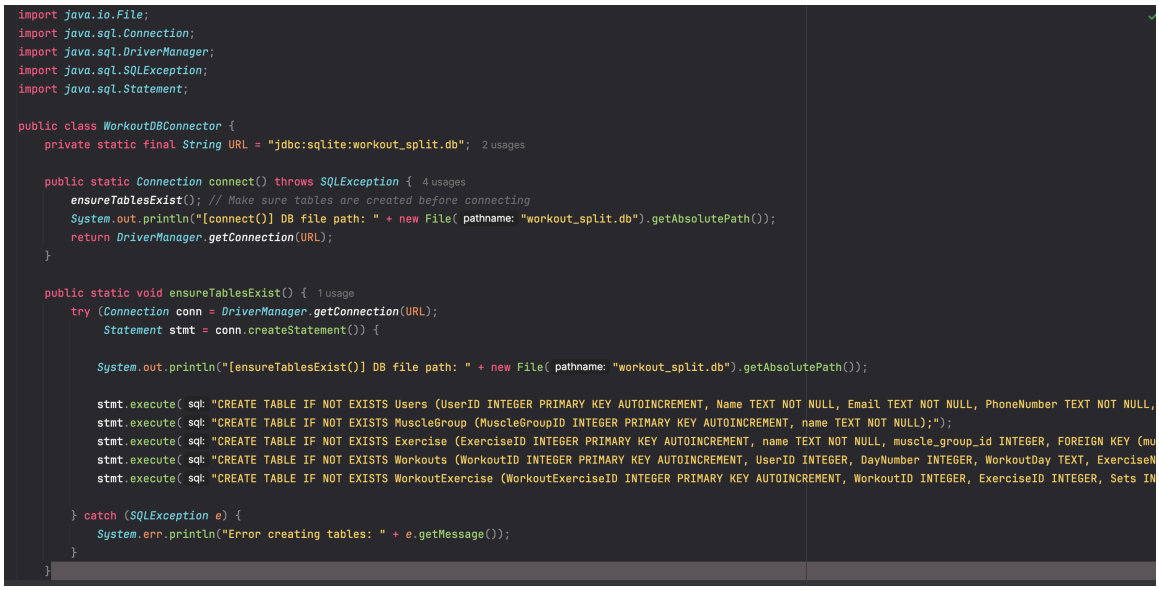
```
        // Assign 5-day plans per user
        for (int userID = 1; userID <= 15; userID++) {
            for (int day = 1; day <= 5; day++) {
                String workoutDay = workoutDays[day - 1];
                // Choose 2-3 random exercises
                int numberOfExercises = 2 + random.nextInt( bound: 2);
                int[] chosenExercises = random.ints( randomNumberOrigin: 1,  randomNumberBound: 21).distinct().limit(numberOfExercise

                for (int exIndex : chosenExercises) {
                    // Get exercise name
                    String exNameQuery = "SELECT name FROM Exercise WHERE ExerciseID = " + exIndex;
                    var rs = stmt.executeQuery(exNameQuery);
                    String exName = rs.next() ? rs.getString( columnLabel: "name") : "Exercise";

                    // Insert into Workouts
                    stmt.execute(String.format(
                            "INSERT INTO Workouts (WorkoutID, UserID, DayNumber, WorkoutDay, ExerciseName, Sets, Reps, Duration)
                                    "VALUES (%d, %d, %d, '%s', '%s', %d, %d, %d);",
                            workoutID, userID, day, workoutDay, exName,
                            3 + random.nextInt( bound: 2), 8 + random.nextInt( bound: 5), 20 + random.nextInt( bound: 41)));

                    // Insert into WorkoutExercise
                    stmt.execute(String.format(
                            "INSERT INTO WorkoutExercise (WorkoutExerciseID, WorkoutID, ExerciseID, Sets, Reps, Duration) " +
                                    "VALUES (%d, %d, %d, %d, %d, %d);",
                            workoutExerciseID++, workoutID, exIndex,
                            3 + random.nextInt( bound: 2), 8 + random.nextInt( bound: 5), 20 + random.nextInt( bound: 41)));

                    rs.close();
                    workoutID++;
                }
            }
        }
```

```
            System.out.println("Successfully populated 15 users with randomized 5-day workouts.");
            System.out.println("Inserting into DB at: " + new java.io.File( pathname: "workout_split.db").getAbsolutePath());

            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

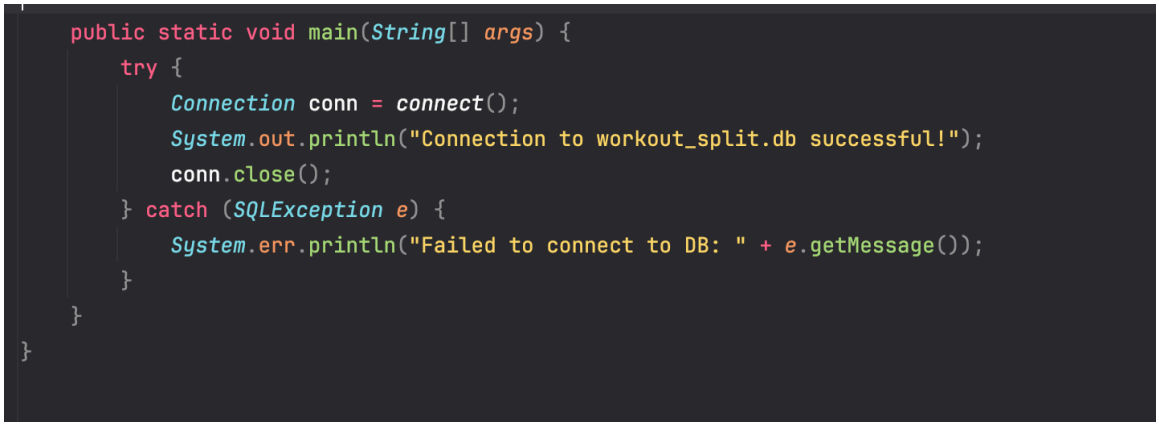
3. ClearWorkoutTables.java – Clears all existing workout data for reset.

```java
import java.sql.Connection;
import java.sql.Statement;

public class ClearWorkoutTables {
    public static void main(String[] args) {
        try (Connection conn = WorkoutDBConnector.connect();
             Statement stmt = conn.createStatement()) {

            // Temporarily disable foreign key checks
            stmt.execute( sql: "PRAGMA foreign_keys = OFF");

            // Delete all data from workout-related tables
            stmt.executeUpdate( sql: "DELETE FROM WorkoutExercise");
            stmt.executeUpdate( sql: "DELETE FROM Workouts");
            stmt.executeUpdate( sql: "DELETE FROM Exercise");
            stmt.executeUpdate( sql: "DELETE FROM MuscleGroup");
            stmt.executeUpdate( sql: "DELETE FROM Users");

            // Re-enable foreign key checks
            stmt.execute( sql: "PRAGMA foreign_keys = ON");

            System.out.println("All workout data cleared.");
        } catch (Exception e) {
            System.err.println("Error clearing tables: " + e.getMessage());
        }
    }
}
```

4. WorkoutDBConnector.java – Manages database connection and ensures tables exist.



```java
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class WorkoutDBConnector {
    private static final String URL = "jdbc:sqlite:workout_split.db";  2 usages

    public static Connection connect() throws SQLException {  4 usages
        ensureTablesExist(); // Make sure tables are created before connecting
        System.out.println("[connect()] DB file path: " + new File( pathname: "workout_split.db").getAbsolutePath());
        return DriverManager.getConnection(URL);
    }

    public static void ensureTablesExist() {  1 usage
        try (Connection conn = DriverManager.getConnection(URL);
             Statement stmt = conn.createStatement()) {

            System.out.println("[ensureTablesExist()] DB file path: " + new File( pathname: "workout_split.db").getAbsolutePath());

            stmt.execute( sql: "CREATE TABLE IF NOT EXISTS Users (UserID INTEGER PRIMARY KEY AUTOINCREMENT, Name TEXT NOT NULL, Email TEXT NOT NULL, PhoneNumber TEXT NOT NULL,
            stmt.execute( sql: "CREATE TABLE IF NOT EXISTS MuscleGroup (MuscleGroupID INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL);");
            stmt.execute( sql: "CREATE TABLE IF NOT EXISTS Exercise (ExerciseID INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, muscle_group_id INTEGER, FOREIGN KEY (mu
            stmt.execute( sql: "CREATE TABLE IF NOT EXISTS Workouts (WorkoutID INTEGER PRIMARY KEY AUTOINCREMENT, UserID INTEGER, DayNumber INTEGER, WorkoutDay TEXT, ExerciseN
            stmt.execute( sql: "CREATE TABLE IF NOT EXISTS WorkoutExercise (WorkoutExerciseID INTEGER PRIMARY KEY AUTOINCREMENT, WorkoutID INTEGER, ExerciseID INTEGER, Sets IN

        } catch (SQLException e) {
            System.err.println("Error creating tables: " + e.getMessage());
        }
    }
```



```java
    public static void main(String[] args) {
        try {
            Connection conn = connect();
            System.out.println("Connection to workout_split.db successful!");
            conn.close();
        } catch (SQLException e) {
            System.err.println("Failed to connect to DB: " + e.getMessage());
        }
    }
}
```

5. Menu.java – Main JavaFX GUI with navigation, data views, and user interaction.

```java
import javafx.application.Application;
import javafx.beans.property.*;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;



import java.sql.*;

public class Menu extends Application {
    private Connection conn;  7 usages
    private Scene mainMenuScene, userViewScene, exerciseScene, allWorkoutScene;  5 usages
    private TableView<User> userTable = new TableView<>();  4 usages
    private TableView<Workout> workoutTable = new TableView<>();  3 usages
```

```java
public static void main(String[] args) { launch(args); }

@Override
public void start(Stage stage) {
    try {
        conn = WorkoutDBConnector.connect();
    } catch (SQLException e) {
        e.printStackTrace();
        return;
    }

    mainMenuScene = buildMainMenuScene(stage);
    userViewScene = buildUserViewScene(stage);
    exerciseScene = buildExerciseScene(stage);
    allWorkoutScene = buildAllWorkoutScene(stage);

    stage.setTitle("Workout_split DB");
    stage.setScene(mainMenuScene);
    stage.show();
}
```

```
private Scene buildMainMenuScene(Stage stage) {  1 usage
    VBox layout = new VBox( v: 15);
    layout.setPadding(new Insets( v: 30));
    layout.setAlignment(Pos.CENTER);
    layout.setStyle("-fx-background-color: #f5f5f5;");

    layout.getChildren().addAll(
            createNavButton( label: "User",  color: "#4285F4", () -> stage.setScene(userViewScene)),
            createNavButton( label: "Exercises",  color: "#34A853", () -> stage.setScene(exerciseScene)),
            createNavButton( label: "All Workouts",  color: "#FABB05", () -> stage.setScene(allWorkoutScene)),
            createNavButton( label: "Exit",  color: "#EA4335", stage::close)
    );

    return new Scene(layout,  v: 350,  v1: 400);
}

private Button createNavButton(String label, String color, Runnable action) {  4 usages
    Button btn = new Button(label);
    btn.setStyle("-fx-background-color: " + color + "; -fx-text-fill: white; -fx-font-size: 14px; -fx-font-weight: bold;");
    btn.setMinWidth(180);
    btn.setOnAction(e -> action.run());
    return btn;
}

private void showInfo(String message) {  no usages
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Info");
    alert.setContentText(message);
    alert.showAndWait();
}
```

```
private Scene buildUserViewScene(Stage stage) {  1 usage
    userTable.getColumns().setAll(
            createCol( title: "ID", u -> u.idProperty().asString()),
            createCol( title: "Name", User::nameProperty),
            createCol( title: "Goal", User::goalProperty),
            createCol( title: "Email", User::emailProperty),
            createCol( title: "Phone", User::phoneProperty),
            createCol( title: "Membership", User::membershipProperty)
    );
    userTable.setItems(loadUsers());

    userTable.setRowFactory(tv -> {
        TableRow<User> row = new TableRow<>();
        row.setOnMouseClicked(e -> {
            if (!row.isEmpty()) {
                workoutTable.setItems(loadWorkoutsForUser(row.getItem().getId()));
            }
        });
        return row;
    });

    workoutTable.getColumns().setAll(
            createCol( title: "Day", Workout::dayNumberProperty),
            createCol( title: "Workout Day", Workout::workoutDayProperty),
            createCol( title: "Exercise", Workout::exerciseNameProperty),
            createCol( title: "Sets", w -> w.setsProperty().asString()),
            createCol( title: "Reps", w -> w.repsProperty().asString()),
            createCol( title: "Duration", w -> w.durationProperty().asString())
    );
```

```
    Button backBtn = new Button( s: "Back");
    backBtn.setOnAction(e -> stage.setScene(mainMenuScene));

    VBox layout = new VBox( v: 10, backBtn, new Label( s: "Users"), userTable, new Label( s: "Workout Plan"), workoutTable);
    layout.setPadding(new Insets( v: 20));
    return new Scene(layout,  v: 950,  v1: 600);
}

private Scene buildExerciseScene(Stage stage) {  1 usage
    TextArea output = new TextArea();
    output.setEditable(false);
    StringBuilder sb = new StringBuilder();

    try (Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(
                 sql: "SELECT e.ExerciseID, e.name, m.name AS muscle_group " +
                         "FROM Exercise e JOIN MuscleGroup m ON e.muscle_group_id = m.MuscleGroupID"
         )) {
        while (rs.next()) {
            sb.append("ID: ").append(rs.getInt( columnLabel: "ExerciseID"))
                    .append(" | Name: ").append(rs.getString( columnLabel: "name"))
                    .append(" | Muscle Group: ").append(rs.getString( columnLabel: "muscle_group")).append("\n");
        }
    } catch (SQLException e) {
        sb.append("Error loading exercises.");
    }
```

```
    output.setText(sb.toString());

    Button backBtn = new Button( s: "Back");
    backBtn.setOnAction(e -> stage.setScene(mainMenuScene));

    VBox layout = new VBox( v: 10, backBtn, new Label( s: "All Exercises"), output);
    layout.setPadding(new Insets( v: 20));
    return new Scene(layout,  v: 600,  v1: 500);
}

private Scene buildAllWorkoutScene(Stage stage) {  1 usage
    TextArea output = new TextArea();
    output.setEditable(false);
    StringBuilder sb = new StringBuilder();

    try (Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(
                 sql: "SELECT w.UserID, u.Name, w.DayNumber, w.WorkoutDay, w.ExerciseName, w.Sets, w.Reps, w.Duration " +
                         "FROM Workouts w JOIN Users u ON w.UserID = u.UserID ORDER BY w.UserID, w.DayNumber")) {
        while (rs.next()) {
            sb.append("User: ").append(rs.getString( columnLabel: "Name"))
                    .append(" | Day ").append(rs.getInt( columnLabel: "DayNumber"))
                    .append(" | ").append(rs.getString( columnLabel: "WorkoutDay"))
                    .append(" | ").append(rs.getString( columnLabel: "ExerciseName"))
                    .append(" | Sets: ").append(rs.getInt( columnLabel: "Sets"))
                    .append(" | Reps: ").append(rs.getInt( columnLabel: "Reps"))
                    .append(" | Duration: ").append(rs.getInt( columnLabel: "Duration")).append(" mins\n");
        }
    } catch (SQLException e) {
        sb.append("Error loading workouts.");
    }

    output.setText(sb.toString());
```

```
    Button backBtn = new Button( s: "Back");
    backBtn.setOnAction(e -> stage.setScene(mainMenuScene));

    VBox layout = new VBox( v: 10, backBtn, new Label( s: "All Workouts"), output);
    layout.setPadding(new Insets( v: 20));
    return new Scene(layout,  v: 800,  v1: 600);
}

private <T> TableColumn<T, String> createCol(String title, javafx.util.Callback<T, ObservableValue<String>> mapper) {  12 usages
    TableColumn<T, String> col = new TableColumn<>(title);
    col.setCellValueFactory(cellData -> mapper.call(cellData.getValue()));
    return col;
}

private ObservableList<User> loadUsers() {  1 usage
    ObservableList<User> users = FXCollections.observableArrayList();
    try (Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery( sql: "SELECT * FROM Users")) {
        while (rs.next()) {
            System.out.println("Found user: " + rs.getString( columnLabel: "Name"));

            users.add(new User(
                    rs.getInt( columnLabel: "UserID"),
                    rs.getString( columnLabel: "Name"),
                    rs.getString( columnLabel: "workout_goal"),
                    rs.getString( columnLabel: "Email"),
                    rs.getString( columnLabel: "PhoneNumber"),
                    rs.getString( columnLabel: "membership_plan")
            ));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
    return users;
}

private ObservableList<Workout> loadWorkoutsForUser(int userId) {  1 usage
    ObservableList<Workout> list = FXCollections.observableArrayList();
    try {
        String sql = "SELECT w.DayNumber, e.Name AS ExerciseName, we.Sets, we.Reps, we.Duration " +
                "FROM Workouts w JOIN WorkoutExercise we ON w.WorkoutID = we.WorkoutID " +
                "JOIN Exercise e ON we.ExerciseID = e.ExerciseID WHERE w.UserID = ? ORDER BY w.DayNumber";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt( parameterIndex: 1, userId);
        ResultSet rs = pstmt.executeQuery();

        String[] types = {"Push Day", "Pull Day", "Leg Day", "Cardio", "Full Body"};
        int lastDay = -1;

        while (rs.next()) {
            int day = rs.getInt( columnLabel: "DayNumber");
            String type = types[(day - 1) % types.length];
            String dayStr = (day == lastDay) ? "" : String.valueOf(day);
            String typeStr = (day == lastDay) ? "" : type;

            list.add(new Workout(dayStr, typeStr, rs.getString( columnLabel: "ExerciseName"),
                    rs.getInt( columnLabel: "Sets"), rs.getInt( columnLabel: "Reps"), rs.getInt( columnLabel: "Duration")));
            lastDay = day;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return list;
}
```

```java
public static class User {  10 usages
    private final int id;  2 usages
    private final SimpleIntegerProperty idProp;  2 usages
    private final SimpleStringProperty name, goal, email, phone, membership;  2 usages

    public User(int id, String name, String goal, String email, String phone, String membership) {  1 usage
        this.id = id;
        this.idProp = new SimpleIntegerProperty(id);
        this.name = new SimpleStringProperty(name);
        this.goal = new SimpleStringProperty(goal);
        this.email = new SimpleStringProperty(email);
        this.phone = new SimpleStringProperty(phone);
        this.membership = new SimpleStringProperty(membership);
    }

    public int getId() { return id; }  1 usage
    public SimpleIntegerProperty idProperty() { return idProp; }  1 usage
    public SimpleStringProperty nameProperty() { return name; }  1 usage
    public SimpleStringProperty goalProperty() { return goal; }  1 usage
    public SimpleStringProperty emailProperty() { return email; }  1 usage
    public SimpleStringProperty phoneProperty() { return phone; }  1 usage
    public SimpleStringProperty membershipProperty() { return membership; }  1 usage
}
```

6. Workout.java – Code for storing workout details.

```java
public static class Workout {  7 usages
    private final SimpleStringProperty dayNumber;  2 usages
    private final SimpleStringProperty workoutDay;  2 usages
    private final SimpleStringProperty exerciseName;  2 usages
    private final SimpleIntegerProperty sets, reps, duration;  2 usages

    public Workout(String day, String type, String ex, int s, int r, int d) {  1 usage
        dayNumber = new SimpleStringProperty(day);
        workoutDay = new SimpleStringProperty(type);
        exerciseName = new SimpleStringProperty(ex);
        sets = new SimpleIntegerProperty(s);
        reps = new SimpleIntegerProperty(r);
        duration = new SimpleIntegerProperty(d);
    }

    public SimpleStringProperty dayNumberProperty() { return dayNumber; }  1 usage
    public SimpleStringProperty workoutDayProperty() { return workoutDay; }  1 usage
    public SimpleStringProperty exerciseNameProperty() { return exerciseName; }  1 usage
    public SimpleIntegerProperty setsProperty() { return sets; }  1 usage
    public SimpleIntegerProperty repsProperty() { return reps; }  1 usage
    public SimpleIntegerProperty durationProperty() { return duration; }  1 usage
}

@Override
public void stop() throws Exception {
    if (conn != null) conn.close();
}
```

```java
public class Workout {  no usages
    private int dayNumber;  2 usages
    private String workoutDay;  2 usages
    private String exerciseName;  2 usages
    private int sets;  2 usages
    private int reps;  2 usages
    private int duration;  2 usages

    // Constructor
    public Workout(int dayNumber, String workoutDay, String exerciseName, int sets, int reps, int duration) {  no usages
        this.dayNumber = dayNumber;
        this.workoutDay = workoutDay;
        this.exerciseName = exerciseName;
        this.sets = sets;
        this.reps = reps;
        this.duration = duration;
    }

    // Getters and setters (if needed)
    public int getDayNumber() {  no usages
        return dayNumber;
    }

    public String getWorkoutDay() {  no usages
        return workoutDay;
    }

    public String getExerciseName() {  no usages
        return exerciseName;
    }

    public int getSets() {  no usages
        return sets;
    }
```

```java
    public int getReps() {  no usages
        return reps;
    }

    public int getDuration() {  no usages
        return duration;
    }
}
```

User Guide

To run the Workout Split Manager:

1. Ensure Java and JavaFX are properly installed.
2. Compile and run the Menu.java class.
3. The main menu allows navigation to view users, exercises, and all workouts.
4. Click on a user to view their specific 5-day workout plan.
5. To regenerate data, run InsertingWorkoutData.java.
6. To reset the database, run ClearWorkoutTables.java.

Conclusion

The Workout Split Manager demonstrates a full-stack integration of database management and GUI interaction using Java. Through this project, the team explored the importance of structured database design, the challenges of randomized data generation, and the creation of an intuitive JavaFX interface. This application not only reflects the theoretical knowledge gained in class but also provides a glimpse into real-world software development for health and fitness systems.