GYM PROGRESS TRACKER DATABASE USING ORACLE APEX

DECEMBER 16, 2024

PL/SQL PROJECT REPORT
Authored by:
Joash Daligcon
Lance Mirano

Table of Contents

Executive Summary

Brief Overview of the Project

The Gym Progress Tracker Project manages aspects of gym operations such as member registration, attendance tracking, and payment transactions. It also monitors members' progress through attributes such as calorie intake, height, and weight for effective fitness management. Furthermore, it keeps track of trainer assignments and equipment usage, that can aid in determining training insights and optimizing the overall gym experience. The authors developed this database project with reference to OracleFlix Project Study in Oracle APEX.

Key Findings and Conclusions

This database project enables the successful creation of a relational database system for gym fitness centres. A project ERD was first created, and the database objects are then created including 6 tables, 3 functions, 6 views, 7 stored procedures, and 3 triggers. The implementation of constraints ensures data accuracy and integrity, while realistic sample data supports practical functionality. Features such as form validations, interactive reports, and visual graphs enhance usability and provide valuable insights. Furthermore, 4 users are created: Administrator, Member, Trainer, and Finance Staff.

This database project became an avenue for the authors to increase their proficiency in Database Programming, as well as their skill in technical documentation of a database project.

The implemented gym progress tracker relational database is thoroughly described in this report, as well as detailed recommendations for future work.

1. Introduction

Purpose of the Project

The purpose of this project is to create a database system for gym centers that will primarily keep track of the fitness progress of its members. In addition, this project will help in managing member registration and payment, attendance, gym equipment and usage records, and trainer assignments. By monitoring and keeping track of all these data, this database project can give substantial information that will aid in the maintenance scheduling of the gym facility and equipment, and in the optimization of trainer-trainee allocation. Furthermore, it can also help in simplifying administrative tasks such as easy access to identifying membership status and doing payment transactions.

Upon implementation in gym or fitness centers, this project targets to streamline the gym operations, potentially increasing the gym goers – thus, increasing sales and profit. And most importantly, this project aims to further improve the members' fitness, health, and overall well-being.

Scope of the Report

This report describes the comprehensive actions done in completing the project, which is grouped into three (3) different sections including the following: database design, database programming and querying, and application development.

In the database design section, the report shows the Entity-Relationship Diagram (ERD) and the business rules to be followed for the database structure. The ERD follows the Oracle convention. In addition, this section includes the create table scripts along with the table constraints and some sample data for the tables.

In the database programming and querying section, the report includes the create scripts for the views, functions, stored procedures, and triggers along with their explanations and figures to show the behavior of each created object.

In the application development section, the report focuses on showcasing the created navigational structure, interactive grids and forms, validations, reports and charts, and security implementation of the front-end application.

2. Database Design

2.1 Designing Tables

Before creating the tables, an entity-relationship diagram is designed first to provide a high-level understanding of the relationship between the entities. Afterwards, the tables are created from the designed model and the table constraints are created to establish each relationship and business rules. Finally, data are inserted into the tables which will complete the tables of the project.

2.1.1 Entity-Relationship Diagram (ERD)

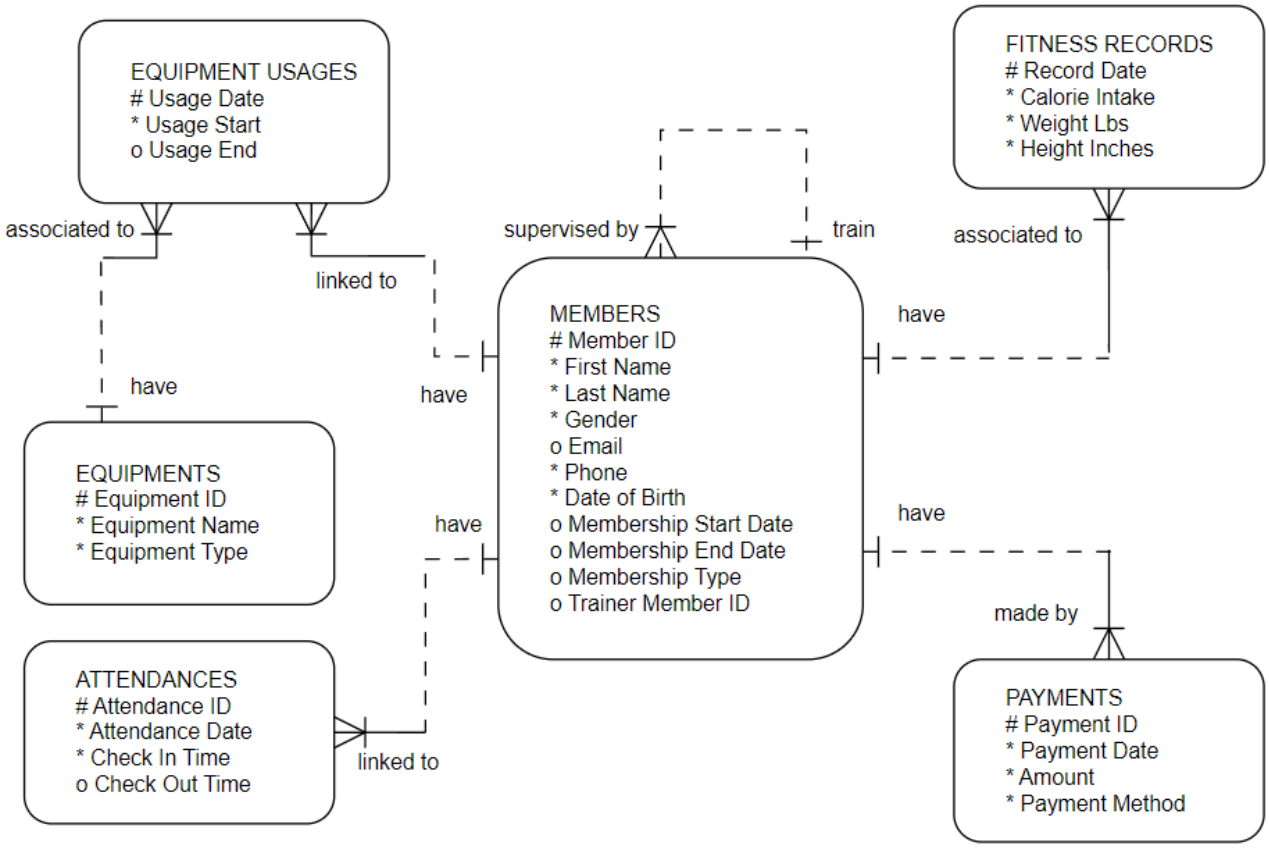


Figure 1. Gym Progress Tracker ERD (Created using draw.io).

This ERD displays the relationships and cardinalities between the entities following the Oracle drawing conventions [1]. The entities are represented by soft boxes, the unique identifiers are marked with a hash sign (#), the mandatory attributes are marked with an asterisk (*), and the optional attributes are marked with a circle (o). Solid lines are used for relationships that are required and dashed lines are used for those that are optional. The lines are terminated by either single toe or a crow's foot to show their cardinalities. Overall, this ERD is designed to satisfy the business requirements listed in the following section and will be used in creating the database tables.

2.1.2 Business Rules

1. Each member may be supervised by a member (trainer)
2. Each member (trainer) may train one or many members
3. Each attendance record must be linked to one and only one member
4. Each member may have one or many attendance records
5. Each equipment usage must be linked to one and only one member
6. Each member may have one or many equipment usages
7. Each equipment usage must be associated to one and only one equipment
8. Each equipment may have one or many equipment usages
9. Each fitness record must be associated to one and only one member
10. Each member may have one or many fitness records
11. Each payment record must be made by one and only one member
12. Each member may have one or many payment records

2.1.3 Creating Tables (DDL Scripts)

This section outlines the creation of database tables of the Gym Progress Tracker database. All of the six (6) tables include a Primary Key to ensure unique identification of records. For instance, the Members table uses MemberID. The scripts below show the table definition along with its attributes.

```
-- Create the Members table
CREATE TABLE Members (
    MemberID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    FirstName VARCHAR2(50)  NOT NULL,
    LastName VARCHAR2(50)   NOT NULL,
    Gender CHAR(1)          NOT NULL,
    Email VARCHAR2(50),
    Phone VARCHAR2(20)      NOT NULL,
    DateOfBirth DATE        NOT NULL,
    MembershipStart DATE,
    MembershipEnd DATE,
    MembershipType VARCHAR2(10),
    TrainerMbrID NUMBER
);
-- Create the Payments table
CREATE TABLE Payments (
    PaymentID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    PaymentDate DATE            NOT NULL,
    Amount NUMBER(10, 2)        NOT NULL,
    PaymentMethod VARCHAR2(50)  NOT NULL,
    MemberID NUMBER             NOT NULL
);
```

```
-- Create the Attendances table
CREATE TABLE Attendances (
    AttendanceID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    AttendanceDate DATE     NOT NULL,
    CheckInTime TIMESTAMP   NOT NULL,
    CheckOutTime TIMESTAMP,
    MemberID NUMBER         NOT NULL
);
-- Create the Equipments table
CREATE TABLE Equipments (
    EquipmentID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    EquipmentName VARCHAR2(50) NOT NULL,
    EquipmentType VARCHAR2(50) NOT NULL
);
-- Create the EquipmentUsages table
CREATE TABLE EquipmentUsages (
    UsageDate DATE          NOT NULL,
    MemberID NUMBER         NOT NULL,
    EquipmentID NUMBER      NOT NULL,
    UsageStart TIMESTAMP    NOT NULL,
    UsageEnd TIMESTAMP,
    -- Member can use each equipment once per day:
    --   MemberID + UsageDate + EquipmentID must be unique
    PRIMARY KEY (UsageDate, MemberID, EquipmentID)
);
-- Create the FitnessRecords table
CREATE TABLE FitnessRecords (
    RecordDate DATE             NOT NULL,
    MemberID NUMBER             NOT NULL,
    CalorieIntake NUMBER(10, 2) NOT NULL,
    WeightLbs NUMBER(10, 2)     NOT NULL,
    HeightInches NUMBER(10, 2)  NOT NULL,
    PRIMARY KEY (RecordDate, MemberID)
);
```

Not Null constraints are also applied to essential fields, such as FirstName and LastName in the Members table and PaymentDate in the Payments table, to ensure critical data is always provided. Unique constraints are used where appropriate, such as in the EquipmentUsages table, which enforces unique usage records for each combination of UsageDate, MemberID, and EquipmentID.

Additionally, foreign key relationships are established to link tables and maintain data integrity. For example, the MemberID in the Payments and Attendances tables references the Members table, creating a clear relationship between members and their respective payments or attendance records. This will be discussed in further in the next section.

2.2 Defining Constraints

This project uses constraints to enforce data integrity and ensure valid entries. Foreign key constraints establish relationships between tables, such as linking TrainerMbrID in the Members table to MemberID and connecting MemberID in tables like Payments, Attendances, EquipmentUsages, and FitnessRecords back to the Members table. Check constraints are used to allow only specific values, such as restricting Gender in the Members table to 'M' or 'F', and limiting MembershipType to 'Standard' or 'Premium'. Similarly, in the Payments table, valid amounts are limited to $50 or $100, and PaymentMethod must be 'Cash', 'Debit Card', or 'Credit Card'. Naming conventions such as fk* and chk* are also used. These constraints ensure consistent and accurate data entry across the database.

```
-- For the Members table
ALTER TABLE Members
ADD CONSTRAINT fkMembersMbrID FOREIGN KEY (TrainerMbrID) REFERENCES Members(MemberID);
ALTER TABLE Members
ADD CONSTRAINT chkMembersGender CHECK (Gender IN ('M', 'F'));
ALTER TABLE Members
ADD CONSTRAINT chkMembersMembershipType
     CHECK (MembershipType IN ('Standard', 'Premium'));

-- For the Payments table
ALTER TABLE Payments
ADD CONSTRAINT fkPaymentsMbrID FOREIGN KEY (MemberID) REFERENCES Members(MemberID);
ALTER TABLE payments
ADD CONSTRAINT chkPaymentsAmount CHECK (Amount IN (50, 100));
ALTER TABLE payments
ADD CONSTRAINT chkPaymentsPaymentMethod
     CHECK (PaymentMethod IN ('Cash', 'Debit Card', 'Credit Card'));

-- For the Attendances table
ALTER TABLE Attendances
ADD CONSTRAINT fkAttendancesMbrID FOREIGN KEY (MemberID) REFERENCES Members(MemberID);

-- For the EquipmentUsages table
ALTER TABLE EquipmentUsages
ADD CONSTRAINT fkEquipUsagesMembID FOREIGN KEY (MemberID)
     REFERENCES Members(MemberID);
ALTER TABLE EquipmentUsages
ADD CONSTRAINT fkEquipUsagesEquipID FOREIGN KEY (EquipmentID)
     REFERENCES Equipments(EquipmentID);

-- For the FitnessRecords table
ALTER TABLE FitnessRecords
ADD CONSTRAINT fkFitnessRecordsMbrID FOREIGN KEY (MemberID)
     REFERENCES Members(MemberID);
```

2.3 Inserting Sample Data

Realistic data was generated using Mockaroo [2], a tool specifically designed for creating mock datasets. This ensures that all entries accurately represent the functionality and purpose of the application. Each table contains data relevant to its use case to support real-world scenario of a gym centre. A total of six (6) CSV files are used to load the data in Oracle Application Express (APEX).

	A	B	C	D	E	F	G	H	I	J	K
1	MemberID	FirstName	LastName	Gender	Email	Phone	DateOfBirth	MembershipStart	MembershipEnd	MembershipType	TrainerMbrID
2	1	John	Doe	M	john.doe@example.com	1.23E+09	1990-05-15	2023-08-04	2024-05-09	Standard	
3	2	Jane	Smith	F	jane.smith@example.com	9.88E+09	1985-08-20	2024-03-29	2024-05-11	Standard	
4	3	Rahul	Kumar	M	rahul.kumar@example.com	5.55E+09	1995-11-10	2024-01-05	2024-05-02	Premium	1
5	4	Emily	Davis	F	emily.davis@example.com	4.45E+09	1988-04-25	2024-04-04	2024-05-03	Premium	2
6	5	Juan	Dela Cruz	M	juan.delacruz@example.com	6.67E+09	1992-09-30	2024-04-05	2024-05-05	Premium	1
7	6	Yuma	Lawerence	M	ylawerence0@miibeian.gov.cn	1.59E+09	1998-08-17	2024-04-09	2024-05-08	Premium	1
8	10	Demetra	Commander	F	dcommander1@answers.com	2.38E+09	1999-02-14	2024-04-09	2024-05-08	Premium	1
9	12	Kelly	Panner	M	kpanner2@dell.com	9.16E+09	2004-09-30	2024-04-10	2024-05-09	Premium	1
10	13	Filmer	Cromly	M	fcromly3@symantec.com	2.17E+09	1995-08-24	2024-04-12	2024-05-11	Premium	2

	A	B	C	D	E
1	PaymentID	PaymentDate	Amount	PaymentMethod	MemberID
2	1	2023-08-04	100	Credit Card	1
3	3	2024-01-05	50	Debit Card	3
4	12	2024-04-03	100	Credit Card	2
5	13	2024-04-04	50	Cash	4
6	14	2024-04-05	50	Debit Card	5
7	15	2024-04-07	50	Credit Card	6
8	16	2024-04-09	100	Cash	10
9	17	2024-04-10	100	Cash	12
10	18	2024-04-10	50	Cash	1
11	21	2024-04-12	100	Credit Card	13
12	22	2024-04-12	50	Credit Card	2

	A	B	C	D	E
1	Attendanc	AttendanceDate	CheckInTime	CheckOutTime	MemberID
2	19	2023-08-04	2023-08-04 7:30	2023-08-04 9:30	1
3	20	2023-08-05	2023-08-05 7:30	2023-08-05 9:30	1
4	21	2023-08-06	2023-08-06 7:30	2023-08-06 9:30	1
5	22	2023-08-07	2023-08-07 7:30	2023-08-07 9:30	1
6	23	2023-08-08	2023-08-08 7:30	2023-08-08 9:30	1
7	24	2023-08-09	2023-08-09 7:30	2023-08-09 9:30	1
8	25	2023-08-10	2023-08-10 7:30	2023-08-10 9:30	1
9	26	2023-08-11	2023-08-11 7:30	2023-08-11 9:30	1
10	27	2023-08-12	2023-08-12 7:30	2023-08-12 9:30	1
11	28	2023-08-13	2023-08-13 7:30	2023-08-13 9:30	1

	A	B	C	D	E
1	UsageDate	MemberID	Equipmen	UsageStart	UsageEnd
2	2023-08-04	1	3	2023-08-04 7:45	2023-08-04 8:30
3	2023-08-05	1	3	2023-08-04 7:45	2023-08-04 8:30
4	2023-08-06	1	3	2023-08-04 7:45	2023-08-04 8:30
5	2023-08-07	1	3	2023-08-04 7:45	2023-08-04 8:30
6	2023-08-08	1	3	2023-08-04 7:45	2023-08-04 8:30
7	2023-08-09	1	3	2023-08-04 7:45	2023-08-04 8:30
8	2023-08-10	1	3	2023-08-04 7:45	2023-08-04 8:30
9	2023-08-11	1	3	2023-08-04 7:45	2023-08-04 8:30
10	2023-08-12	1	3	2023-08-04 7:45	2023-08-04 8:30
11	2023-08-13	1	3	2023-08-04 7:45	2023-08-04 8:30
12	2023-08-14	1	3	2023-08-04 7:45	2023-08-04 8:30
13	2023-08-15	1	3	2023-08-04 7:45	2023-08-04 8:30

	A	B	C	D	E
1	RecordDate	MemberID	CalorieIntake	WeightLbs	HeightInches
2	2023-08-04	1	2600	151	68
3	2023-08-05	1	2600	151	68
4	2023-08-06	1	2600	150.5	68
5	2023-08-07	1	2600	151	68
6	2023-08-08	1	2600	150.5	68
7	2023-08-09	1	2600	151	68
8	2023-08-10	1	2600	151	68
9	2023-08-11	1	2600	151.5	68
10	2023-08-12	1	2600	151	68
11	2023-08-13	1	2600	151.5	68

	A	B	C
1	EquipmentID	EquipmentName	EquipmentType
2	1	Dumbbells	Strength
3	2	Barbell	Strength
4	3	Treadmill	Cardio
5	4	Exercise Bike	Cardio
6	5	Elliptical Machine	Cardio

Figure 2. CSV Files for each of the entities.

After all the CSVs are loaded to Oracle Apex, the tables are queried to ensure that the data are stored accurately, which is briefly shown in the Figure 3 below.

MEMBERID	FIRSTNAME	LASTNAME	GENDER	EMAIL	PHONE	DATEOFBIRTH	MEMBERSHIPSTART	MEMBERSHIPEND	MEMBERSHIPTYPE	TRAINERMBRID
1	John	Doe	M	john.doe@example.com	3301171703	15-May-1990	04-Aug-2023	03-Sep-2023	Standard	-
2	Jane	Smith	F	jane.smith@example.com	9876543210	20-Aug-1985	29-Mar-2024	11-May-2024	Standard	-
3	Rahul	Kumar	M	rahul.kumar@example.com	5551234567	10-Nov-1995	05-Jan-2024	02-May-2024	Premium	1
4	Emily	Davis	F	emily.davis@example.com	4449876543	25-Apr-1988	04-Apr-2024	03-May-2024	Premium	2
5	Juan	Dela Cruz	M	juan.delacruz@example.com	6667890123	30-Sep-1992	05-Apr-2024	05-May-2024	Premium	1

PAYMENTID	PAYMENTDATE	AMOUNT	PAYMENTMETHOD	MEMBERID
1	04-Aug-2023	50	Credit Card	1
2	03-Dec-2024	50	Cash	7
3	05-Jan-2024	50	Debit Card	3
12	03-Apr-2024	50	Credit Card	2
13	04-Apr-2024	50	Cash	4

ATTENDANCEID	ATTENDANCEDATE	CHECKINTIME	CHECKOUTTIME	MEMBERID
477	13-Dec-2024	13-DEC-24 12.00.00.000000 AM	12-DEC-24 12.01.00.000000 AM	144
395	12-Dec-2024	12-DEC-24 12.00.00.000000 AM	08-DEC-24 12.00.00.000000 AM	70
417	08-Dec-2024	08-DEC-24 06.31.28.949238 PM	08-DEC-24 06.47.16.859230 PM	221
481	12-Dec-2024	12-DEC-24 11.33.46.098133 PM	12-DEC-24 11.34.06.663142 PM	264
493	13-Dec-2024	13-DEC-24 12.15.50.646602 AM	13-DEC-24 12.16.32.834815 AM	265

EQUIPMENTID	EQUIPMENTNAME	EQUIPMENTTYPE
1	Dumbbells	Strength
2	Barbell	Strength
3	Treadmill	Cardio
4	Exercise Bike	Cardio
5	Elliptical Machine	Cardio

USAGEDATE	MEMBERID	EQUIPMENTID	USAGESTART	USAGEEND
09-Dec-2024	190	1	09-DEC-24 05.37.58.000000 AM	09-DEC-24 05.59.48.000000 AM
12-Dec-2024	150	2	12-DEC-24 12.46.41.000000 PM	12-DEC-24 12.46.59.000000 PM
12-Dec-2024	150	2	12-DEC-24 02.21.12.000000 PM	12-DEC-24 02.21.30.000000 PM
12-Dec-2024	150	22	12-DEC-24 02.21.44.000000 PM	12-DEC-24 02.22.10.000000 PM
13-Dec-2024	265	2	13-DEC-24 12.18.27.000000 AM	13-DEC-24 12.18.48.000000 AM

RECORDDATE	MEMBERID	CALORIEINTAKE	WEIGHTLBS	HEIGHTINCHES
04-Aug-2023	1	2600	151	68
05-Aug-2023	1	2600	151	68
06-Aug-2023	1	2600	150.5	68
07-Aug-2023	1	2600	151	68
08-Aug-2023	1	2600	150.5	68
09-Aug-2023	1	2600	151	68

Figure 3. Table Data after Loading the CSV to Oracle APEX.

3. Database Programming and Querying

A total of six (6) views were created to simplify data access and present key information from multiple tables in a meaningful way. Additionally, seven (7) stored procedures and three (3) functions were written to automate common operations to ensure efficiency and reducing manual effort. To maintain data integrity and enforce business rules, three (3) triggers were implemented. The next sections will discuss each database object one by one.

3.1 Creating Views

All the views in this project are created with the Database Object I documentation [3] used as reference. Coding conventions and best practices such as commenting and adding the prefix "vw" is followed. Of all the 6 views, a total of three (3) views will be used for the Application front-end in Oracle APEX, which are Views #4 to 6. This is because View #4 is basically a combination of Views #1, 2, and 3.

```
-- View #1: vwMemberInitialWeight (Member View)
CREATE OR REPLACE VIEW vwMemberInitialWeight
AS
    SELECT  fr.MemberID,
            fr.RecordDate,
            fr.WeightLbs
    FROM FitnessRecords fr
    INNER JOIN (
        SELECT MemberID, MIN(RecordDate) AS InitialRecordDate
        FROM FitnessRecords
        GROUP BY MemberID) initialfr
    ON fr.MemberID = initialfr.MemberID AND fr.RecordDate = initialfr.InitialRecordDate;
/
```

View #1: This view displays the initial weight of members that are with existing fitness record. Figure 4 below shows a query of the vwMemberInitialWeight view.

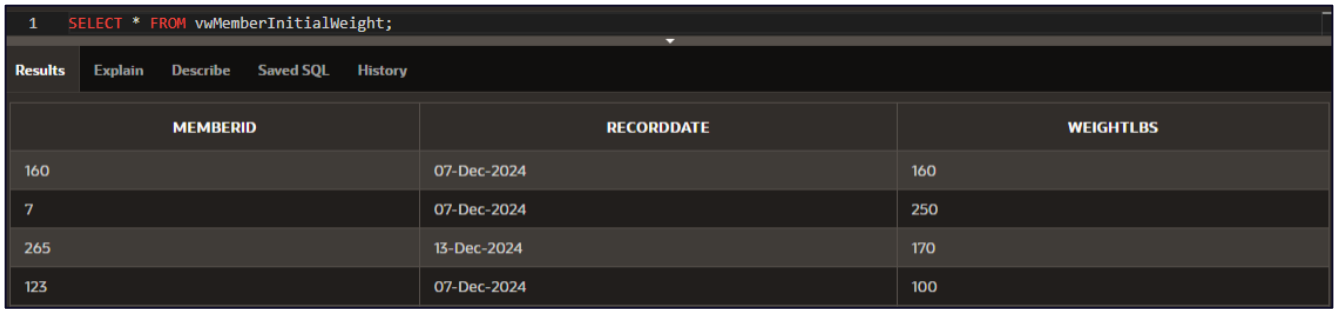


Figure 4. Sample Query of vwMemberInitialWeight from Oracle APEX.

```
-- View #2: vwMemberLatestWeight (Member View)
CREATE OR REPLACE VIEW vwMemberLatestWeight
AS
    SELECT  fr.MemberID,
            fr.RecordDate,
            fr.WeightLbs
    FROM FitnessRecords fr
    INNER JOIN (
        SELECT MemberID, MAX(RecordDate) AS InitialRecordDate
        FROM FitnessRecords
        GROUP BY MemberID) latestfr
    ON fr.MemberID = latestfr.MemberID AND fr.RecordDate = latestfr.InitialRecordDate;
/
```

View #2: This view displays the latest weight of members that are with existing fitness record. Figure 5 below shows a query of the vwMemberLatestWeight view.

```
1  SELECT * FROM vwMemberLatestWeight;
```

Results | Explain | Describe | Saved SQL | History

MEMBERID	RECORDDATE	WEIGHTLBS
160	08-Dec-2024	100
265	13-Dec-2024	170
123	07-Dec-2024	100
159	07-Dec-2024	140

Figure 5. Sample Query of vwMemberLatestWeight from Oracle APEX.

```
-- View #3: vwMemberInitialVsLatest (Member View) - (View #1+2)
CREATE OR REPLACE VIEW vwMemberInitialVsLatest
AS
    SELECT i.MemberID,
           i.RecordDate AS InitialRecordDate,
           i.WeightLbs AS InitialWeightLbs,
           l.RecordDate AS LatestRecordDate,
           l.WeightLbs AS LatestWeightLbs
    FROM vwMemberInitialWeight i
    INNER JOIN vwMemberLatestWeight l
    ON i.MemberID = l.MemberID;
/
```

View #3: This view displays the initial and latest weight of members with existing fitness record. It is basically a combination of view #1 and #2. The Figure 6 below dipslays a sample query of the vwMemberInitialVsLatest view.

```
1  SELECT * FROM vwMemberInitialVsLatest;
```

Results | Explain | Describe | Saved SQL | History

MEMBERID	INITIALRECORDDATE	INITIALWEIGHTLBS	LATESTRECORDDATE	LATESTWEIGHTLBS
160	07-Dec-2024	160	08-Dec-2024	100
265	13-Dec-2024	170	13-Dec-2024	170
123	07-Dec-2024	100	07-Dec-2024	100
159	07-Dec-2024	140	07-Dec-2024	140
135	07-Dec-2024	130	07-Dec-2024	110

Figure 6. Sample Query of vwMemberInitialVsLatest from Oracle APEX.

```
-- View #4: vwMemberLatestStatus (Member View)
CREATE OR REPLACE VIEW vwMemberLatestStatus
AS
    SELECT m.MemberID, m.FirstName, m.LastName, m.Gender, m.DateOfBirth,
           m.MembershipStart, m.MembershipEnd,
           vs.InitialRecordDate, vs.InitialWeightLbs,
           vs.LatestRecordDate, vs.LatestWeightLbs
    FROM Members m
    LEFT JOIN vwMemberInitialVsLatest vs
    ON m.MemberID = vs.MemberID;
/
```

View #4: This view displays the data of all the members alongside their initial and latest weight. Figure 7 below shows query of the vwMemberLatestStatus view.

```
1  SELECT * FROM vwMemberLatestStatus;
```

Results | Explain | Describe | Saved SQL | History

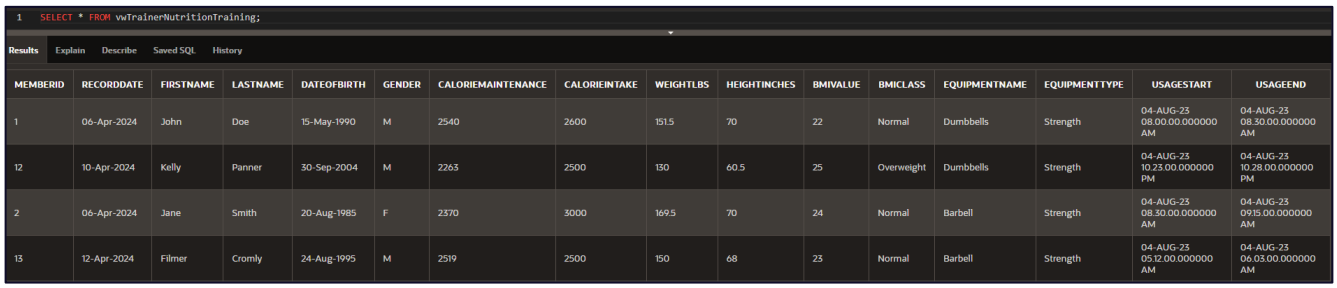
MEMBERID	FIRSTNAME	LASTNAME	GENDER	DATEOFBIRTH	MEMBERSHIPSTART	MEMBERSHIPEND	INITIALRECORDDATE	INITIALWEIGHTLBS	LATESTRECORDDATE	LATESTWEIGHTLBS
160	Jonathan	Jimeno	M	07-Dec-2024	07-Dec-2024	06-Jan-2025	07-Dec-2024	160	08-Dec-2024	100
265	Milissent	Mayers	M	01-Dec-2008	13-Dec-2024	12-Jan-2025	13-Dec-2024	170	13-Dec-2024	170
123	Shino	Mahiru	F	02-Nov-2024	07-Dec-2024	06-Jan-2025	07-Dec-2024	100	07-Dec-2024	100
159	yanaaaa	meeii	F	07-Dec-2024	08-Dec-2024	07-Jan-2025	07-Dec-2024	140	07-Dec-2024	140

Figure 7. Sample Query of vwMemberLatestStatus from Oracle APEX.

```
-- View #5: vwTrainerNutritionTraining (Trainer View)
CREATE OR REPLACE VIEW vwTrainerNutritionTraining
AS
    SELECT  m.MemberID, fr.RecordDate,
            m.FirstName, m.LastName, m.DateOfBirth, m.Gender,
            GetCalorieMaintenance(fr.RecordDate, m.MemberID) AS CalorieMaintenance,
            fr.CalorieIntake, fr.WeightLbs, fr.HeightInches,
            GetBMIValue(fr.HeightInches, fr.WeightLbs) AS BMIValue,
            GetBMIClass(GetBMIValue(fr.HeightInches, fr.WeightLbs)) AS BMIClass,
            e.EquipmentName, e.EquipmentType, u.UsageStart, u.UsageEnd
    FROM Members m
    JOIN FitnessRecords fr
        ON m.MemberID = fr.MemberID
    JOIN EquipmentUsages u
        ON m.MemberID = u.MemberID AND u.UsageDate = fr.RecordDate
```

```
    JOIN Equipments e
        ON u.EquipmentID = e.EquipmentID;
/
```

View #5: This view is for the gym trainers, and it displays the membership data of the members alongside their fitness records, equipment used, and usage records. The view also shows the calculated calorie maintenance needed by the member, their BMI value and classification. It will only show those members that have existing fitness record, equipment, and usage record. Figure 8 below shows a sample query of the vwTrainerNutritionTraining view.



```
1  SELECT * FROM vwTrainerNutritionTraining;
```

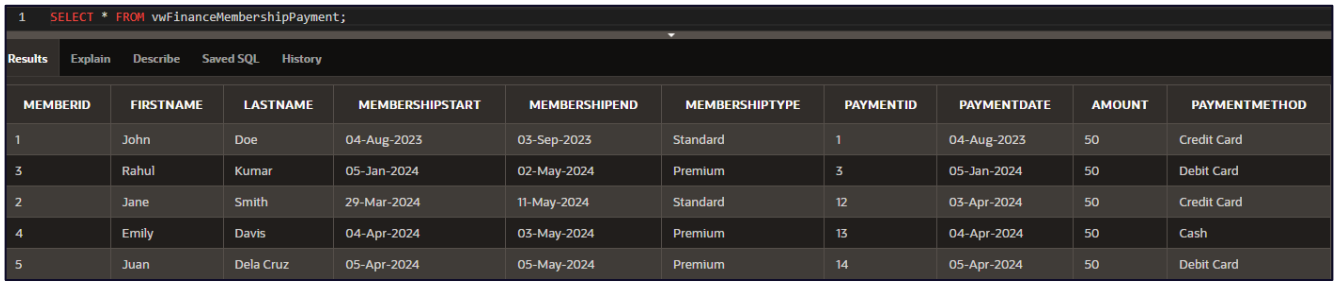
Results | Explain | Describe | Saved SQL | History

MEMBERID	RECORDDATE	FIRSTNAME	LASTNAME	DATEOFBIRTH	GENDER	CALORIEMAINTENANCE	CALORIEINTAKE	WEIGHTLBS	HEIGHTINCHES	BMIVALUE	BMICLASS	EQUIPMENTNAME	EQUIPMENTTYPE	USAGESTART	USAGEEND
1	06-Apr-2024	John	Doe	15-May-1990	M	2540	2600	151.5	70	22	Normal	Dumbbells	Strength	04-AUG-23 08.00.00.000000 AM	04-AUG-23 08.30.00.000000 AM
12	10-Apr-2024	Kelly	Panner	30-Sep-2004	M	2263	2500	130	60.5	25	Overweight	Dumbbells	Strength	04-AUG-23 10.23.00.000000 PM	04-AUG-23 10.28.00.000000 PM
2	06-Apr-2024	Jane	Smith	20-Aug-1985	F	2370	3000	169.5	70	24	Normal	Barbell	Strength	04-AUG-23 08.30.00.000000 AM	04-AUG-23 09.15.00.000000 AM
13	12-Apr-2024	Filmer	Cromly	24-Aug-1995	M	2519	2500	150	68	23	Normal	Barbell	Strength	04-AUG-23 05.12.00.000000 AM	04-AUG-23 06.03.00.000000 AM

Figure 8. Sample Query of vwTrainerNutritionTraining from Oracle APEX.

```
-- View #6: vwFinanceMembershipPayment (Finance View)
CREATE OR REPLACE VIEW vwFinanceMembershipPayment
AS
    SELECT m.MemberID, m.FirstName, m.LastName,
        m.MembershipStart, m.MembershipEnd, m.MembershipType,
        p.PaymentID, p.PaymentDate, p.Amount, p.PaymentMethod
    FROM Members m
    LEFT JOIN Payments p
    ON m.MemberID = p.MemberID;
/
```

View #6: This view is for the finance staff, and it displays the membership data of all the members alongside their payment records. It will show NULL for members with no existing payment record. Figure 9 below shows a sample query of the vwFinanceMembershipPayment view.



```
1  SELECT * FROM vwFinanceMembershipPayment;
```

Results | Explain | Describe | Saved SQL | History

MEMBERID	FIRSTNAME	LASTNAME	MEMBERSHIPSTART	MEMBERSHIPEND	MEMBERSHIPTYPE	PAYMENTID	PAYMENTDATE	AMOUNT	PAYMENTMETHOD
1	John	Doe	04-Aug-2023	03-Sep-2023	Standard	1	04-Aug-2023	50	Credit Card
3	Rahul	Kumar	05-Jan-2024	02-May-2024	Premium	3	05-Jan-2024	50	Debit Card
2	Jane	Smith	29-Mar-2024	11-May-2024	Standard	12	03-Apr-2024	50	Credit Card
4	Emily	Davis	04-Apr-2024	03-May-2024	Premium	13	04-Apr-2024	50	Cash
5	Juan	Dela Cruz	05-Apr-2024	05-May-2024	Premium	14	05-Apr-2024	50	Debit Card

Figure 9. Sample Query of vwFinanceMembershipPayment from Oracle APEX.

3.2 Creating Functions

All the functions in this project are created with the Database Object II documentation [4] and 9-1 Creating Functions from Oracle [5] used as reference. The functions listed here are the ones that are used from the views as displayed and tested in section 3.1 above.

```
-- Function #1: GetCalorieMaintenance
CREATE OR REPLACE FUNCTION GetCalorieMaintenance (
    p_RecordDate IN FitnessRecords.RecordDate%TYPE,
    p_MemberID IN Members.MemberID%TYPE
) RETURN DECIMAL AS
    -- Declare variables for Gender, DOB (Age), WeightLbs, HeightInches
    v_Gender Members.Gender%TYPE;
    v_DOB Members.DateOfBirth%TYPE;
    v_Age NUMBER;
    v_WeightLbs FitnessRecords.WeightLbs%TYPE;
    v_HeightInches FitnessRecords.HeightInches%TYPE;
    v_WeightKg FitnessRecords.WeightLbs%TYPE;
    v_HeightCm FitnessRecords.HeightInches%TYPE;
    v_BMR DECIMAL(10, 2);
    v_CalorieMaintenance DECIMAL(10, 2);
BEGIN
    -- Get Gender and Date of Birth (DOB)
    SELECT Gender, DateOfBirth
    INTO v_Gender, v_DOB
    FROM Members
    WHERE MemberID = p_MemberID;
    -- Calculate Age
    v_Age := TRUNC(MONTHS_BETWEEN(p_RecordDate, v_DOB) / 12);

    -- Get WeightLbs and HeightInches from FitnessRecords
    SELECT WeightLbs, HeightInches
    INTO v_WeightLbs, v_HeightInches
    FROM FitnessRecords
    WHERE RecordDate = p_RecordDate AND MemberID = p_MemberID;

    -- Convert Weight to Kilograms and Height to Centimeters
    v_WeightKg := v_WeightLbs / 2.204623;
    v_HeightCm := v_HeightInches * 2.54;

    -- Calculate Basal Metabolic Rate (BMR) based on gender
    IF v_Gender = 'M' THEN
        v_BMR := (10 * v_WeightKg) + (6.25 * v_HeightCm) - (5 * v_Age) + 5;
    ELSE
        v_BMR := (10 * v_WeightKg) + (6.25 * v_HeightCm) - (5 * v_Age) - 161;
    END IF;
```

```
    -- Calculate TDEE (Calorie Maintenance) based on BMR and activity level
    -- Total Daily Energy Expenditure is how much energy you burn each day
    -- which is equivalent to the amount of calories your body needs to maintain weight
    v_CalorieMaintenance := v_BMR * 1.55; -- Moderately active

    -- Return the calculated Calorie Maintenance value
    RETURN v_CalorieMaintenance;

EXCEPTION
    WHEN OTHERS THEN
        -- Handle any potential errors (e.g., if the SELECT queries return no rows)
        RETURN NULL;
END GetCalorieMaintenance;
/
```

Function #1: This function is responsible for calculating the calorie maintenance of a member given a particular record date. It gets the member's gender, weight, and height. Then it calculates the age given the record date, and it also calculates the Basal Metabolic Rate (BMR) based on the gender. Afterwards, the BMR is multiplied to a factor for moderately active individuals, and it returns the result as calorie maintenance, which is also known as the Total Daily Energy Expenditure (TDEE) [6].

```
-- Function #2: GetBMIValue
CREATE OR REPLACE FUNCTION GetBMIValue (
    p_HeightInches IN FitnessRecords.HeightInches%TYPE,
    p_WeightLbs IN FitnessRecords.WeightLbs%TYPE
) RETURN DECIMAL AS
    v_BMI DECIMAL(10, 2);
BEGIN
    -- Calculate BMI using the formula (Weight / Height^2) * 703
    v_BMI := (p_WeightLbs / POWER(p_HeightInches, 2)) * 703;

    -- Return the calculated BMI value
    RETURN v_BMI;
EXCEPTION
    WHEN OTHERS THEN
        -- Handle any potential errors
        RETURN NULL;
END GetBMIValue;
/
```

Function #2: This function is responsible for calculating the BMI value given the height in inches and weight in pounds. It uses the BMI formula for imperial units [7] and returns the calculated value.

```
-- Function #3: GetBMIClass
CREATE OR REPLACE FUNCTION GetBMIClass (
    p_BMI IN DECIMAL
) RETURN VARCHAR AS
    v_BMICategory VARCHAR(50);
BEGIN
    -- Initialize the category variable
    v_BMICategory := NULL;

    -- Determine BMI category
    IF p_BMI < 18.5 THEN
        v_BMICategory := 'Underweight';
    ELSIF p_BMI >= 18.5 AND p_BMI < 25 THEN
        v_BMICategory := 'Normal';
    ELSIF p_BMI >= 25 AND p_BMI < 30 THEN
        v_BMICategory := 'Overweight';
    ELSIF p_BMI >= 30 THEN
        v_BMICategory := 'Obese';
    END IF;

    -- Return the determined category
    RETURN v_BMICategory;
EXCEPTION
    WHEN OTHERS THEN
        -- Handle any potential errors
        RETURN NULL;
END GetBMIClass;
/
```

Function #3: This function is responsible for classifying the given BMI value to the weight categories. It uses the standard BMI Chart [7], and the Obesity classes are merged into just one category for this function, and finally the corresponding BMI category is returned.

3.3 Creating Stored Procedures

All the stored procedures are created with the Database Object II documentation [4] and 8-1 Creating Procedures from Oracle [8] used as reference. Coding conventions and best practices such as commenting and adding the prefix “sp” is followed and error handling with the use of Try-Catch is implemented for proper transaction management. Also, all the seven (7) stored procedures will be used for the Application front-end in Oracle APEX.

```
-- Stored Procedure #1: spAddMember
CREATE OR REPLACE PROCEDURE spAddMember (
    p_FirstName       Members.FirstName%TYPE,
    p_LastName        Members.LastName%TYPE,
    p_Gender          Members.Gender%TYPE,
    p_Email           Members.Email%TYPE DEFAULT NULL,
    p_Phone           Members.Phone%TYPE,
    p_DateOfBirth     Members.DateOfBirth%TYPE,
    p_TrainerMbrID    Members.TrainerMbrID%TYPE DEFAULT NULL
) IS
    v_tmpMemberID     Members.MemberID%TYPE;
BEGIN
    -- Check if mandatory fields are provided
    IF p_FirstName IS NULL OR p_LastName IS NULL OR p_Gender IS NULL
       OR p_Phone IS NULL OR p_DateOfBirth IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Mandatory fields cannot be null.');
    END IF;
    -- Insert member into Members table
    INSERT INTO Members (
        FirstName, LastName, Gender, Email, Phone, DateOfBirth, TrainerMbrID
    ) VALUES (
        p_FirstName, p_LastName, p_Gender, p_Email, p_Phone, p_DateOfBirth, p_TrainerMbrID
    );
    -- Print success message
    SELECT MAX(MemberID) INTO v_tmpMemberID FROM Members;
    DBMS_OUTPUT.PUT_LINE('Member successfully added. Your Member ID is: ' || v_tmpMemberID);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
END spAddMember;
/
```

Stored Procedure #1: This stored procedure is responsible for adding a new member to the database. It will check first if the required attributes are given. If so, it will insert the member data to the members table and it will print a success message along with the new Member ID. Otherwise, it will throw an error message that the mandatory fields cannot be null. Also, if something went wrong during the execution, the stored procedure would catch it and will print error occurred.

Figure 10. Execution Query of spAddMember from Oracle APEX.

```
-- Stored Procedure #2: spAddPaymentUpdateMembership
CREATE OR REPLACE PROCEDURE spAddPaymentUpdateMembership (
    p_MemberID       IN Members.MemberID%TYPE,
    p_PaymentAmount  IN Payments.Amount%TYPE,
    p_PaymentMethod  IN Payments.PaymentMethod%TYPE
) AS
    v_PaymentDate      Payments.PaymentDate%TYPE := SYSDATE; -- Assume current date for payment
    v_MembershipStart  Members.MembershipStart%TYPE;
    v_MembershipEnd    Members.MembershipEnd%TYPE;
    v_MembershipType   Members.MembershipType%TYPE;
    v_OldMembershipStart  Members.MembershipStart%TYPE;
BEGIN
    -- Check if mandatory fields are provided
    IF p_MemberID IS NULL OR p_PaymentAmount IS NULL OR p_PaymentMethod IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'MemberID and Payment Info cannot be null.');
    END IF;
     -- Check if the member exists
    BEGIN
        SELECT MembershipStart
        INTO v_OldMembershipStart
        FROM Members
        WHERE MemberID = p_MemberID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20002, 'Member ID does not exist.');
    END;
    -- Add payment record
    INSERT INTO Payments (
        PaymentDate, Amount, PaymentMethod, MemberID
    ) VALUES (
        v_PaymentDate, p_PaymentAmount, p_PaymentMethod, p_MemberID
    );
    -- Calculate MembershipEnd date
    -- (one day before the end of the month following MembershipStart date)
    v_MembershipStart := NVL(v_OldMembershipStart, v_PaymentDate);
    v_MembershipEnd := ADD_MONTHS(v_MembershipStart, 1) - 1;
    -- Check if membership is standard or premium
    IF p_PaymentAmount = 50 THEN
        v_MembershipType := 'Standard';
    ELSIF p_PaymentAmount = 100 THEN
        v_MembershipType := 'Premium';
    ELSE
        RAISE_APPLICATION_ERROR(-20003,
            'Invalid Payment Amount. Only 50 (Standard) or 100 (Premium) are allowed.');
    END IF;
```

```
    -- Update MembershipStart, MembershipEnd, and MembershipType columns in Members table
    UPDATE Members
    SET MembershipStart = v_MembershipStart,
        MembershipEnd   = v_MembershipEnd,
        MembershipType  = v_MembershipType
    WHERE MemberID = p_MemberID;

    -- Print success message
    DBMS_OUTPUT.PUT_LINE('Payment record added and membership details updated.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
        RAISE;
END spAddPaymentUpdateMembership;
/
```

Stored Procedure #2: The above stored procedure is responsible for adding a payment record to the database for the given Member ID. It is also responsible for updating the membership information of that member according to which he paid for. It will check first if the mandatory fields are provided and will also check if the provided Member ID exists. If so, it will add they payment record and it will update the Members table. The Membership Start will be the Payment Date, and the Membership End will be after 1 month less 1 day. The Membership Type will be updated to "Standard" if payment is 50, and "Premium" if payment is 100. The stored procedure will throw an error if the mandatory fields are not provided, if the member ID does not exist, and if something went wrong during the execution.

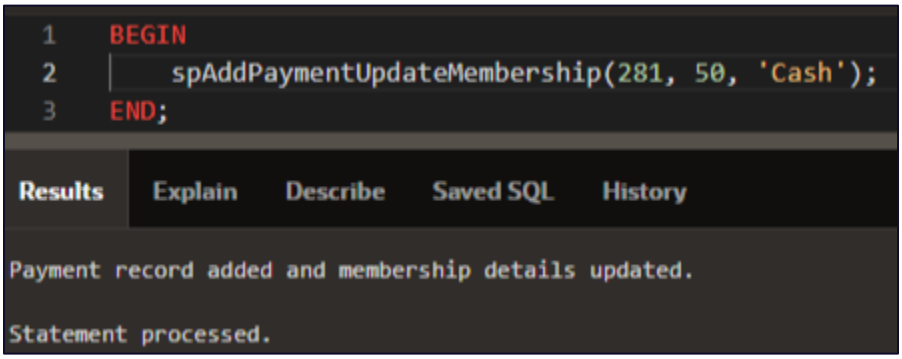


Figure 11. Sample Execution of spAddPaymentUpdateMembership from Oracle APEX.

```
-- Stored Procedure #3: spMemberCheckIn
CREATE OR REPLACE PROCEDURE spMemberCheckIn (
    p_MemberID IN Members.MemberID%TYPE
)
AS
    v_AttendanceDate Attendances.AttendanceDate%TYPE := SYSDATE;
    v_CheckInTime    Attendances.CheckInTime%TYPE := SYSTIMESTAMP;
    v_MembershipStart Members.MembershipStart%TYPE;
    v_MembershipEnd   Members.MembershipEnd%TYPE;
BEGIN
    -- Check if MemberID is provided
    IF p_MemberID IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'MemberID cannot be null.');
    END IF;

    -- Check if the member exists
    BEGIN
        SELECT MembershipStart, MembershipEnd
        INTO v_MembershipStart, v_MembershipEnd
        FROM Members
        WHERE MemberID = p_MemberID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20002, 'Member ID does not exist.');
    END;

    -- Check if the attendance date falls within the member's active membership period
    IF v_AttendanceDate < v_MembershipStart OR v_AttendanceDate > v_MembershipEnd THEN
        RAISE_APPLICATION_ERROR(-20003, 'Attendance date is not within the member''s active
membership period.');
    END IF;

    -- Convert Timezone to MST
    v_CheckInTime := FROM_TZ(v_CheckInTime, 'UTC') AT TIME ZONE 'US/Mountain';

    -- Insert attendance record into the Attendances table
    INSERT INTO Attendances (AttendanceDate, CheckInTime, MemberID)
    VALUES (v_AttendanceDate, v_CheckInTime, p_MemberID);

    -- Success message
    DBMS_OUTPUT.PUT_LINE('Member successfully checked in.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
        RAISE;
END;
/
```

Stored Procedure #3: The stored procedure above is responsible for member attendance check-ins while checking their active membership period. It first checks if the Member ID is provided, and it also checks if that Member ID exists. Afterwards, it checks that member's active membership period is valid. If all checks are passed, the current date and time is converted to MST time zone using the FROM_TZ oracle function [9][10] and then inserted into the Attendances table as a check-in record, and a success message will be printed. Otherwise, it will throw an error message.

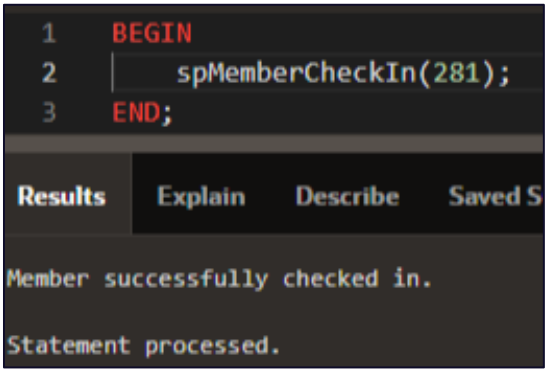


Figure 12. Sample Execution of spMemberCheckIn from Oracle APEX.

```
-- Stored Procedure #4: spMemberCheckOut
CREATE OR REPLACE PROCEDURE spMemberCheckOut (
    p_MemberID IN Members.MemberID%TYPE
)
AS
    v_AttendanceDate Attendances.AttendanceDate%TYPE := SYSDATE; -- Current date without time
    v_CheckOutTime   Attendances.CheckInTime%TYPE := SYSTIMESTAMP; -- Current timestamp
    v_AttendanceID   Attendances.AttendanceID%TYPE;
    v_MembershipStart Members.MembershipStart%TYPE;
    v_MembershipEnd   Members.MembershipEnd%TYPE;
BEGIN
    -- Check if p_MemberID is provided
    IF p_MemberID IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'MemberID cannot be null.');
    END IF;

    -- Check if the member exists and get MembershipStart and MembershipEnd dates
    BEGIN
        SELECT MembershipStart, MembershipEnd
        INTO v_MembershipStart, v_MembershipEnd
        FROM Members
        WHERE MemberID = p_MemberID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20002, 'Member ID does not exist.');
    END;
```

```
    -- Check if the attendance date falls within the member's active membership period
    IF v_AttendanceDate < v_MembershipStart OR v_AttendanceDate > v_MembershipEnd THEN
        RAISE_APPLICATION_ERROR(-20003,
            'Attendance date is not within the member''s active membership period.');
    END IF;
    -- Check if there is an active attendance record (NULL CheckOutTime)
    BEGIN
        SELECT AttendanceID
        INTO v_AttendanceID
        FROM Attendances
        WHERE MemberID = p_MemberID
          AND TO_TIMESTAMP(AttendanceDate) = TO_TIMESTAMP(v_AttendanceDate)
          AND CheckOutTime IS NULL
        ORDER BY AttendanceDate, CheckInTime DESC
        FETCH FIRST ROW ONLY; -- Fetch only the latest attendance record
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20004,
                'Member already clocked out or no active check-in found.');
    END;
    -- Convert Timezone to MST
    v_CheckOutTime := FROM_TZ(v_CheckOutTime, 'UTC') AT TIME ZONE 'US/Mountain';
    -- Update the latest attendance record with the checkout time
    UPDATE Attendances
    SET CheckOutTime = v_CheckOutTime
    WHERE AttendanceID = v_AttendanceID;
    -- Print success message
    DBMS_OUTPUT.PUT_LINE('Member successfully checked out.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
        RAISE;
END;
/
```

Stored Procedure #4: The above stored procedure is responsible for member attendance check-outs while checking their active membership period. It first checks if the Member ID is provided, and it also checks if that Member ID exists. Afterwards, it checks that member's active membership period is valid. It will then check if the member's latest check-in record has no corresponding check-out record, and it will get the Attendance ID of that record. If all checks are passed, the current time is converted to MST time zone and then inserted into that Attendance ID record as a check-out, and a success message will be printed. Otherwise, the stored procedure will throw an error message accordingly.

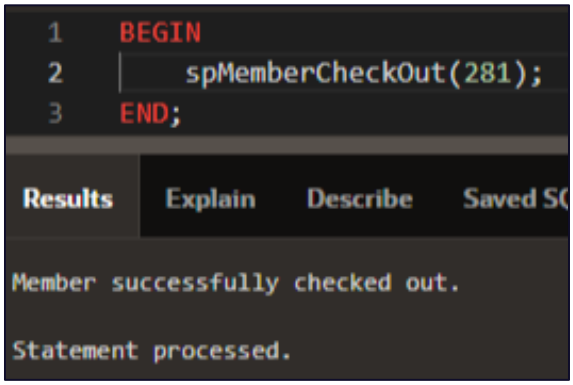


Figure 13. Sample Execution of spMemberCheckOut from Oracle APEX.

```
-- Stored Procedure #5: spAddFitnessRecord
CREATE OR REPLACE PROCEDURE spAddFitnessRecord (
    p_MemberID      IN Members.MemberID%TYPE,
    p_CalorieIntake IN FitnessRecords.CalorieIntake%TYPE,
    p_WeightLbs      IN FitnessRecords.WeightLbs%TYPE,
    p_HeightInches   IN FitnessRecords.HeightInches%TYPE
)
AS
    v_RecordDate FitnessRecords.RecordDate%TYPE := SYSDATE;
    v_MembershipStart Members.MembershipStart%TYPE;
    v_MembershipEnd   Members.MembershipEnd%TYPE;
BEGIN
    -- Check if mandatory fields are provided
    IF p_MemberID IS NULL OR p_CalorieIntake IS NULL OR
       p_WeightLbs IS NULL OR p_HeightInches IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Mandatory fields cannot be null.');
    END IF;
    -- Check if the member exists and get MembershipStart and MembershipEnd dates
    BEGIN
        SELECT MembershipStart, MembershipEnd
        INTO v_MembershipStart, v_MembershipEnd
        FROM Members
        WHERE MemberID = p_MemberID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20002, 'Member ID does not exist.');
    END;
    -- Check if the record date falls within the member's active membership period
    IF v_RecordDate < v_MembershipStart OR v_RecordDate > v_MembershipEnd THEN
        RAISE_APPLICATION_ERROR(-20003,
             'Record date is not within the member''s active membership period.');
    END IF;
```

```
    -- Insert fitness record data
    INSERT INTO FitnessRecords (RecordDate, MemberID, WeightLbs, HeightInches, CalorieIntake)
    VALUES (v_RecordDate, p_MemberID, p_WeightLbs, p_HeightInches, p_CalorieIntake);
    -- Print success message
    DBMS_OUTPUT.PUT_LINE('Fitness record successfully added. BMI is: ' ||
          GetBMIClass(GetBMIValue(p_HeightInches, p_WeightLbs)));
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
        RAISE;
END;
/
```

Stored Procedure #5: The above stored procedure is responsible for adding a fitness record of the members while checking their active membership period. It first checks if the mandatory fields are provided, and it also checks if the given Member ID exists. Afterwards, it checks that member's active membership period is valid. If all checks are passed, the current date is inserted into the Fitness Records table along with the provided weight, height, and calorie intake. Then a success message will be printed with the calculated BMI classification. Otherwise, the stored procedure will throw an error message accordingly.

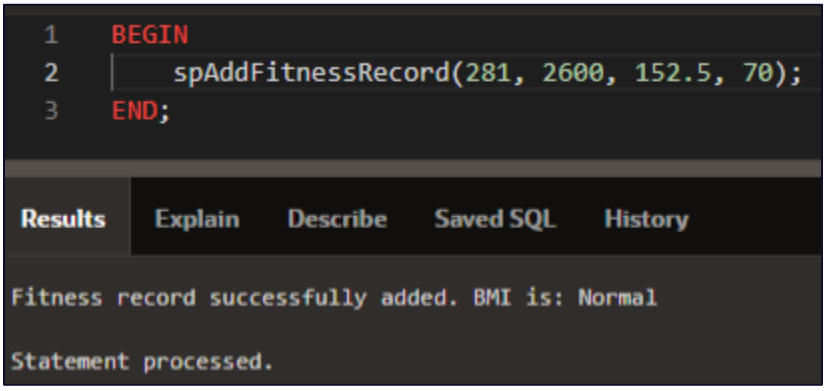


Figure 14. Sample Execution of spAddFitnessRecord from Oracle APEX.

```
-- Stored Procedure #6: spAddEquipmentUsageStart
CREATE OR REPLACE PROCEDURE spAddEquipmentUsageStart (
    p_MemberID    IN Members.MemberID%TYPE,
    p_EquipmentID IN Equipments.EquipmentID%TYPE
)
AS
    v_UsageDate EquipmentUsages.UsageDate%TYPE := SYSDATE;
    v_UsageStart EquipmentUsages.UsageEnd%TYPE := SYSDATE;
    v_MembershipStart Members.MembershipStart%TYPE;
    v_MembershipEnd   Members.MembershipEnd%TYPE;
    v_Exist NUMBER;
BEGIN
```

```
    -- Check if mandatory fields are provided
    IF p_MemberID IS NULL OR p_EquipmentID IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Mandatory fields cannot be null.');
    END IF;

    -- Check if the member exists and get MembershipStart and MembershipEnd dates
    BEGIN
        SELECT MembershipStart, MembershipEnd
        INTO v_MembershipStart, v_MembershipEnd
        FROM Members
        WHERE MemberID = p_MemberID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20002, 'Member ID does not exist.');
    END;

    -- Check if the equipment exists
    BEGIN
        SELECT 1
        INTO v_Exist
        FROM Equipments
        WHERE EquipmentID = p_EquipmentID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20003, 'Equipment ID does not exist.');
    END;
    -- Check if the usage date falls within the member's active membership period
    IF v_UsageDate < v_MembershipStart OR v_UsageDate > v_MembershipEnd THEN
        RAISE_APPLICATION_ERROR(-20004,
             'Usage date is not within the member''s active membership period.');
    END IF;
    -- Insert equipment usage start record
    INSERT INTO EquipmentUsages
        (MemberID, UsageDate, UsageStart, UsageEnd, EquipmentID)
    VALUES
        (p_MemberID, v_UsageDate, v_UsageStart, NULL, p_EquipmentID);
    -- Print success message (Oracle DBMS_OUTPUT usage)
    DBMS_OUTPUT.PUT_LINE('Equipment usage start time successfully added.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
        RAISE;
END;
/
```

Stored Procedure #6: The above stored procedure is responsible for adding an equipment usage start record while checking the member's active membership period. It first checks if the mandatory fields

are provided, and it also checks if the given Member ID exists. Afterwards, it checks if the given Equipment ID exists. Furthermore, it also checks if the member's active membership period is valid. If all checks are passed, the current date and time are inserted into the Equipment Usages table as Usage Start along with the provided Equipment ID. Finally, a success message will be printed. Otherwise, the stored procedure will throw an error message accordingly.

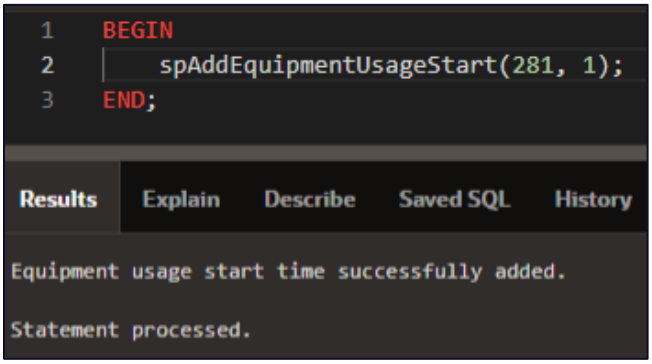


Figure 15. Sample Execution of spAddEquipmentUsageStart from Oracle APEX.

```
-- Stored Procedure #7: spAddEquipmentUsageEnd
CREATE OR REPLACE PROCEDURE spAddEquipmentUsageEnd (
    p_MemberID    IN Members.MemberID%TYPE,
    p_EquipmentID IN Equipments.EquipmentID%TYPE
)
AS
    v_UsageDate EquipmentUsages.UsageDate%TYPE := SYSDATE;
    v_UsageEnd EquipmentUsages.UsageEnd%TYPE := SYSDATE;
    v_MembershipStart Members.MembershipStart%TYPE;
    v_MembershipEnd   Members.MembershipEnd%TYPE;
    v_Exists NUMBER;
BEGIN

    -- Check if mandatory fields are provided
    IF p_MemberID IS NULL OR p_EquipmentID IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Mandatory fields cannot be null.');
    END IF;

    -- Check if the member exists
    BEGIN
        SELECT MembershipStart, MembershipEnd
        INTO v_MembershipStart, v_MembershipEnd
        FROM Members
        WHERE MemberID = p_MemberID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20002, 'Member ID does not exist.');
    END;
```

```
    -- Check if the usage date falls within the member's active membership period
    IF v_UsageDate < v_MembershipStart OR v_UsageDate > v_MembershipEnd THEN
        RAISE_APPLICATION_ERROR(-20004,
             'Usage date is not within the member''s active membership period.');
    END IF;

    -- Check if the equipment exists
    BEGIN
        SELECT 1
        INTO v_Exists
        FROM Equipments
        WHERE EquipmentID = p_EquipmentID
        FETCH FIRST 1 ROWS ONLY;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20004, 'Equipment ID does not exist.');
    END;

    -- Check if the equipment is being used by the member
    BEGIN
        SELECT 1
        INTO v_Exists
        FROM EquipmentUsages
        WHERE MemberID = p_MemberID
        AND EquipmentID = p_EquipmentID
        AND TO_TIMESTAMP(UsageDate) = TO_TIMESTAMP(v_UsageDate)
        FETCH FIRST 1 ROWS ONLY;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20005, 'Equipment ID not in use.');
    END;

    -- Check if the latest check-in has a NULL checkout
    BEGIN
        SELECT 1
        INTO v_Exists
        FROM EquipmentUsages
        WHERE MemberID = p_MemberID
        AND EquipmentID = p_EquipmentID
        AND TO_TIMESTAMP(UsageDate) = TO_TIMESTAMP(v_UsageDate)
        AND UsageEnd IS NULL
        ORDER BY UsageDate DESC, UsageStart DESC
        FETCH FIRST 1 ROWS ONLY;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE_APPLICATION_ERROR(-20006, 'End time already logged.');
    END;
```

```
    -- Update the end time for the equipment usage
    UPDATE EquipmentUsages
    SET UsageEnd = v_UsageEnd
    WHERE MemberID = p_MemberID
    AND EquipmentID = p_EquipmentID
    AND TO_TIMESTAMP(UsageDate) = TO_TIMESTAMP(v_UsageDate)
    AND UsageEnd IS NULL;

    -- Print success message
    DBMS_OUTPUT.PUT_LINE('Equipment usage end time successfully updated.');

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
        RAISE;
END;
/
```

Stored Procedure #7: This stored procedure is responsible for adding an equipment usage end record while checking the member's active membership period. It first checks if the mandatory fields are provided, and it also checks if the given Member ID exists. It then checks if the member's active membership period is valid. Afterwards, it checks if the given Equipment ID exists, and if that equipment is currently recorded to the member. Furthermore, it checks if the latest usage start record has no corresponding usage end record. If all checks are passed, the current time is inserted into the Equipment Usages table as Usage End. Finally, a success message will be printed. Otherwise, the stored procedure will throw an error message accordingly.

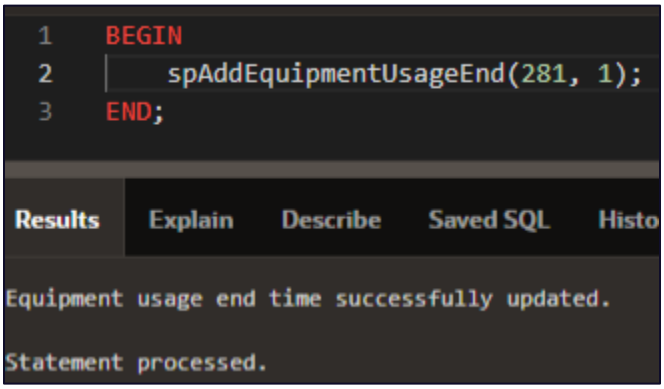


Figure 16. Sample Execution of spAddEquipmentUsageEnd from Oracle APEX.

3.4 Creating Triggers

All the triggers in this project are created with the Database Object II documentation [4] used as reference. Coding conventions and best practices such as commenting and adding the prefix "tg" is followed. All these triggers will be used for the Application front-end in Oracle APEX.

```
-- Trigger #1: tgDeleteAttendanceOutsideGymHours
CREATE OR REPLACE TRIGGER tgDeleteAttendanceOutsideGymHours
BEFORE INSERT ON Attendances
FOR EACH ROW
DECLARE
    -- Convert SYSDATE Timezone to MST
    v_CurrentTime TIMESTAMP := FROM_TZ(SYSDATE, 'UTC') AT TIME ZONE 'US/Mountain';

    v_GymOpenTime    CONSTANT Attendances.CheckInTime%TYPE :=
        TO_TIMESTAMP(TO_CHAR(v_CurrentTime, 'DD-MON-YYYY') || ' ' ||
            '04:30:00', 'DD-MON-YYYY HH24:MI:SS');
    v_GymCloseTime   CONSTANT Attendances.CheckInTime%TYPE :=
        TO_TIMESTAMP(TO_CHAR(v_CurrentTime, 'DD-MON-YYYY') || ' ' ||
            '23:30:00', 'DD-MON-YYYY HH24:MI:SS');
    v_GymLastCheckIn CONSTANT Attendances.CheckInTime%TYPE :=
        v_GymCloseTime - INTERVAL '1' HOUR;
BEGIN
    -- Check if the CheckInTime falls outside gym hours
    IF :NEW.CheckInTime < v_GymOpenTime OR :NEW.CheckInTime > v_GymLastCheckIn THEN
        RAISE_APPLICATION_ERROR(-20113, 'TG1: Check-in is outside gym hours.');
    END IF;
END;
/
```

Trigger #1: The functionality of this trigger is to ensure that the records inserted outside of the allowed gym check-in hours are deleted from the Attendances table. A check-in should only be done within the allowed window hours. The Gym Open Time is set to 4:30AM and the Gym Close Time is set to 11:30PM. The window hours of allowable check-ins are from the Gym Open Time until to 1 hour before the Gym Close Time. This trigger will fire after the insert statement to check the validity of the check-in. If invalid, an error message will be thrown, and the insert transaction will be rolled back.

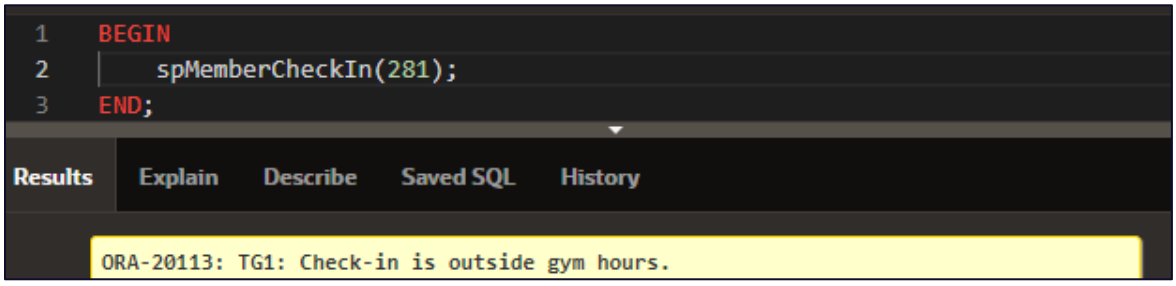


Figure 17. Sample Execution of tgDeleteAttendanceOutsideGymHours from Oracle APEX.

```
-- Trigger #2: tgPreventCheckinsWithoutCheckout
CREATE OR REPLACE TRIGGER tgPreventCheckinsWithoutCheckout
BEFORE INSERT ON Attendances
FOR EACH ROW
DECLARE
    v_count INTEGER;
BEGIN
    -- Check if the member has already checked in but not checked out
    SELECT COUNT(*)
    INTO v_count
    FROM Attendances
    WHERE MemberID = :NEW.MemberID
    AND TO_TIMESTAMP(AttendanceDate) = TO_TIMESTAMP(:NEW.AttendanceDate)
    AND CheckOutTime IS NULL;

    -- If such a record exists, raise an error
    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'TG2: Please check-out first.');
    END IF;

    -- If no violations, the new row will be inserted
END;
/
```

Trigger #2: The functionality of this trigger is to ensure that the members cannot check-in multiple times without checking-out first from the Attendances table. Instead of the insert statement, the trigger will check first if the records to be inserted violate the rule that the member has not checked-out yet. If the rule is violated, the trigger will throw an error message. Otherwise, the insert statement is performed.

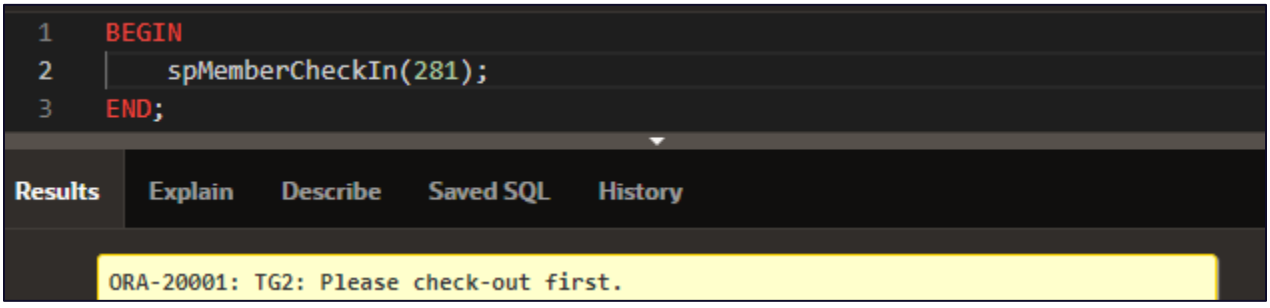


Figure 18. Sample Execution of tgPreventCheckinsWithoutCheckout from Oracle APEX.

```
-- Trigger #3: tgPreventIncorrectPayment
CREATE OR REPLACE TRIGGER tgPreventIncorrectPayment
BEFORE INSERT ON Payments
FOR EACH ROW
DECLARE
    v_TrainerMbrID Members.TrainerMbrID%TYPE;
BEGIN
    -- Retrieve TrainerMbrID for the MemberID being inserted
    SELECT TrainerMbrID
    INTO v_TrainerMbrID
    FROM Members
    WHERE MemberID = :NEW.MemberID;

    -- Validate the Amount based on TrainerMbrID
    IF :NEW.Amount NOT IN (50, 100) THEN
        RAISE_APPLICATION_ERROR(-20002, 'TG3: Must be $50 or $100.');
    ELSIF v_TrainerMbrID IS NULL AND :NEW.Amount <> 50 THEN
        RAISE_APPLICATION_ERROR(-20003, 'TG3: Amount should be 50.');
    ELSIF v_TrainerMbrID IS NOT NULL AND :NEW.Amount <> 100 THEN
        RAISE_APPLICATION_ERROR(-20004, 'TG3: Amount should be 100.');
    END IF;

    -- If validations pass, the row is inserted automatically.
END;
/
```

Trigger #3: This functionality of this trigger is to ensure that incorrect payment information cannot be inserted to the Payments table. To be considered correct payment, the rules are as follows. First, the payment amount should only be 50 or 100. Second, if the member has no trainer, the amount should be 50. Finally, if the member has a trainer, the amount to be paid should be 100. Instead of the insert statement, the trigger will check first if the records to be inserted violate any of the rules. If violated, the trigger will throw an error message accordingly. Otherwise, the insert statement is performed.

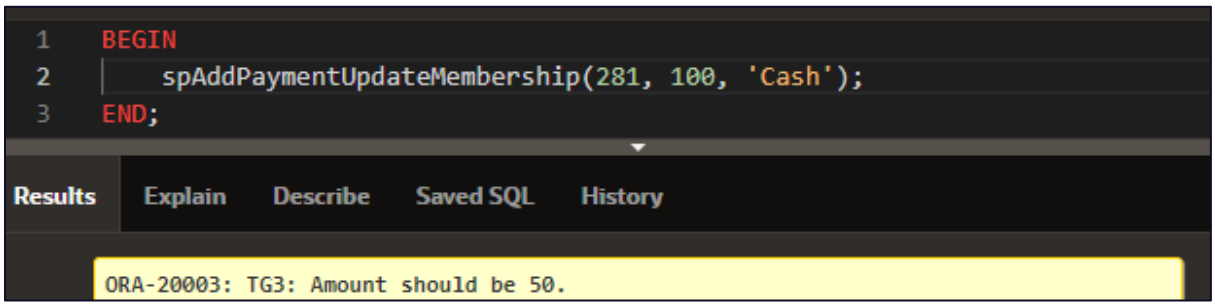


Figure 19. Sample Execution of tgPreventIncorrectPayment from Oracle APEX.

4. Application Development

This section focuses on showcasing the project's front-end using Oracle APEX. A well-structured navigation menu with links to interactive grids and reports for different tables was created, complete with visually appealing icons for ease of use. Interactive Grids and Forms were implemented to allow users to view, edit, and manage data efficiently. Reports were developed to display relevant insights, providing a clear overview of the database's contents. Form validations were added to ensure that all inputs meet the necessary criteria, maintaining data integrity. Security measures were also implemented to control user access and protect sensitive data.

4.1 Navigational Structure

This navigation of the project is divided into a number of groups to categorize each functionality of the application. There are seven (7) parent navigation sections which is denoted by ">" symbol, and each parent has a sub-navigation link. It structured to be visually organized along with relevant icons.

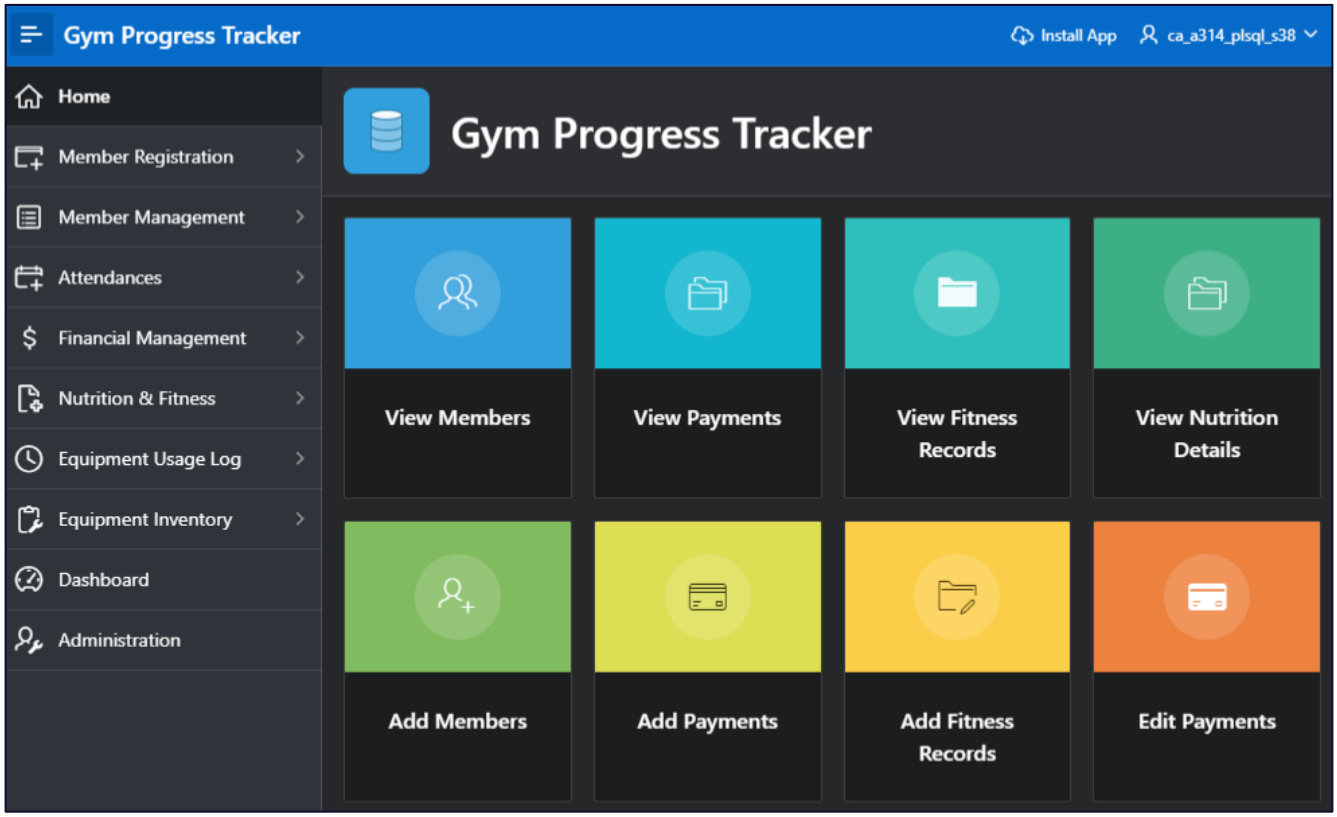


Figure 20. Navigational Structure of the Gym Progress Tracker Project in Oracle APEX.

4.2 Interactive Grids & Forms

Interactive Grids provide a dynamic way for users to view, edit, and manage multiple rows of data simultaneously in a spreadsheet-like interface. These grids allow users to filter, sort, and interact with large sets of data efficiently. They offer the flexibility of in-place editing, making them ideal for displaying summary information where users may need to make bulk updates across rows. In this project, Figures 21, 22, 23 display some sample interactive grids used in the front-end application using Oracle APEX.

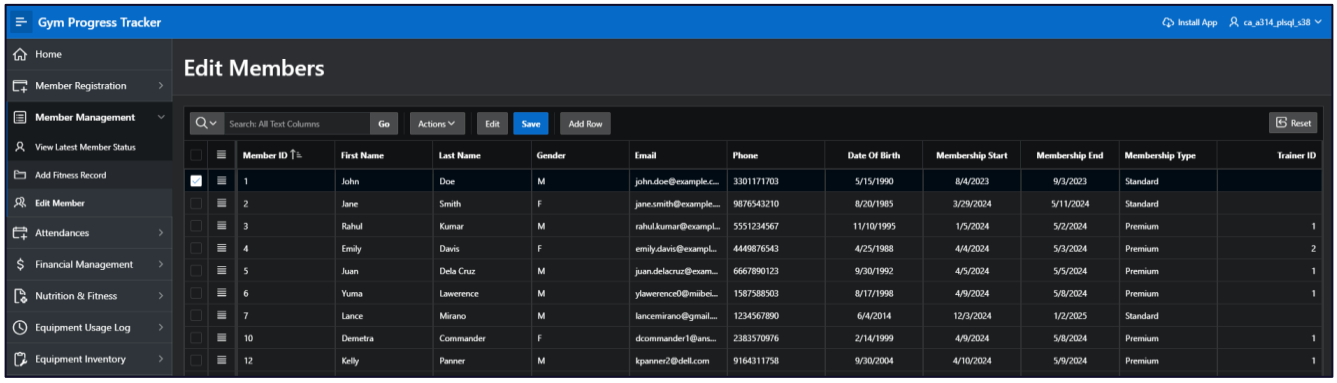


Figure 21. Members Interactive Grid of the Gym Progress Tracker Project.

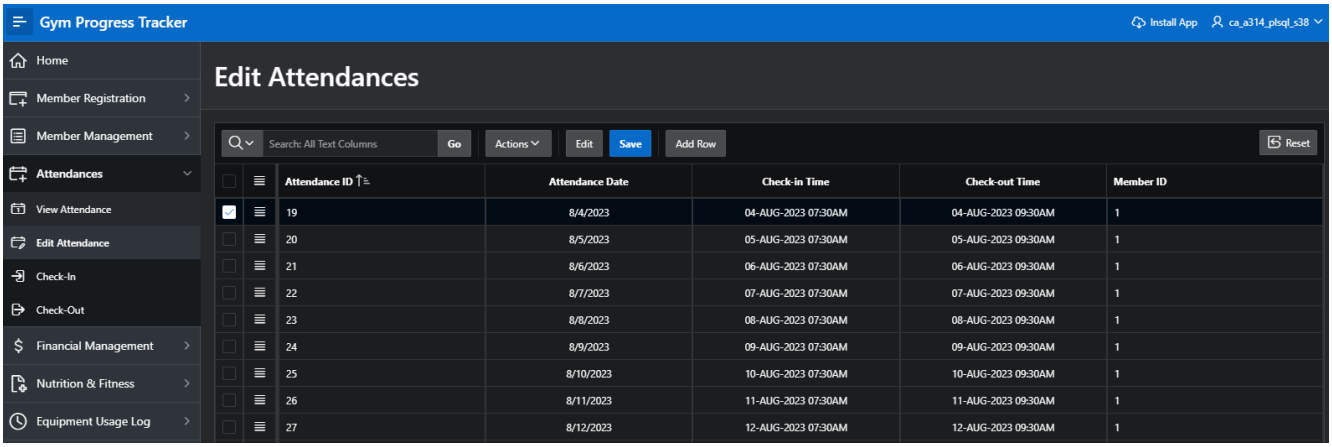


Figure 22. Attendances Interactive Grid of the Gym Progress Tracker Project.

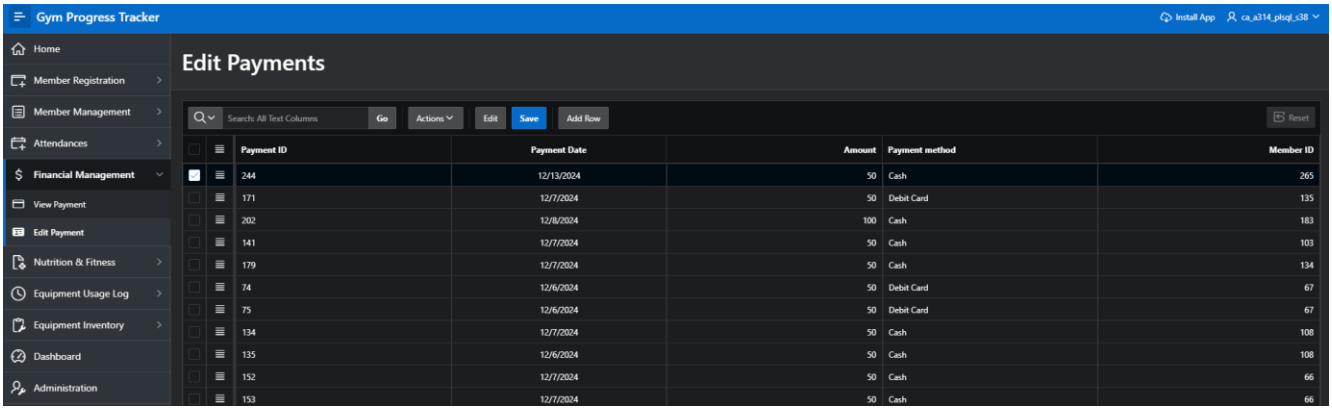


Figure 23. Payments Interactive Grid of the Gym Progress Tracker Project.

Next, forms are designed for detailed data entry or editing which are focused on a single record at a time. Forms are more structured than grids and allow users to input data into individual fields, providing a cleaner and more guided experience. They are ideal for adding new records or modifying existing ones where each field needs to be validated and processed separately. In this project, Figures 24 to 29 display the forms for all tables used in the front-end application using Oracle APEX.

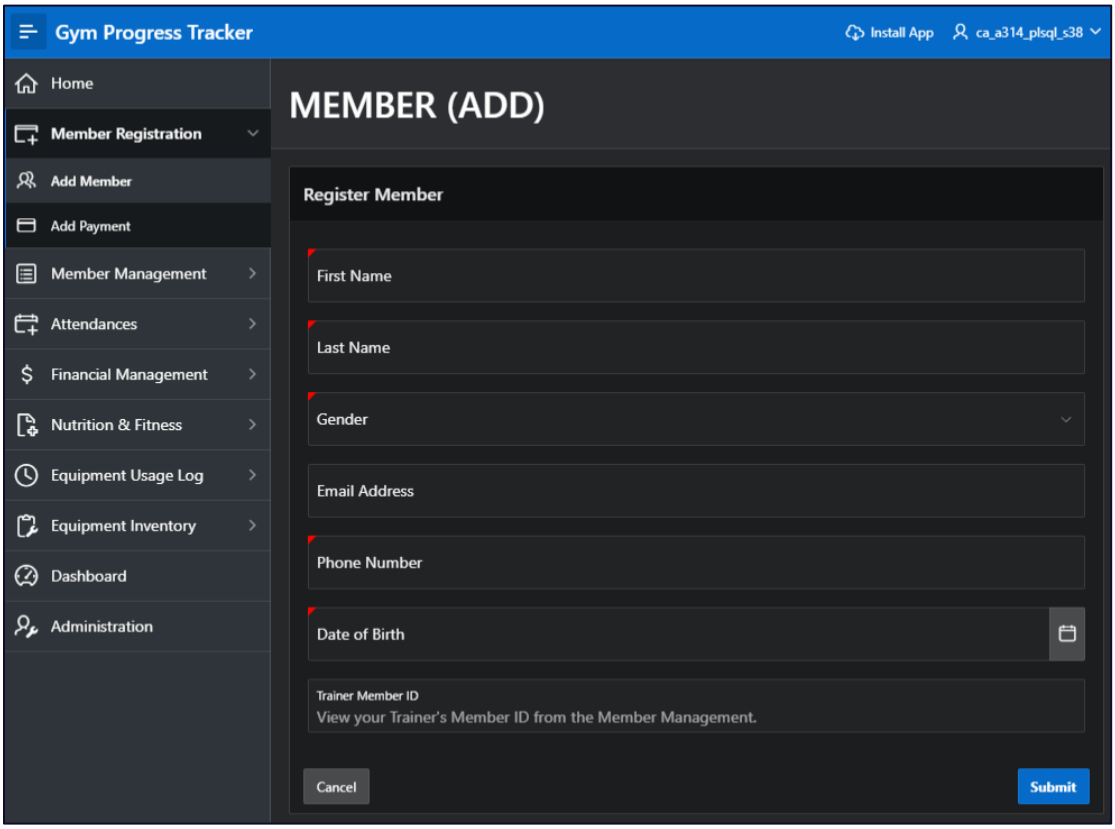


Figure 24. Add Member Form of the Gym Progress Tracker Project.

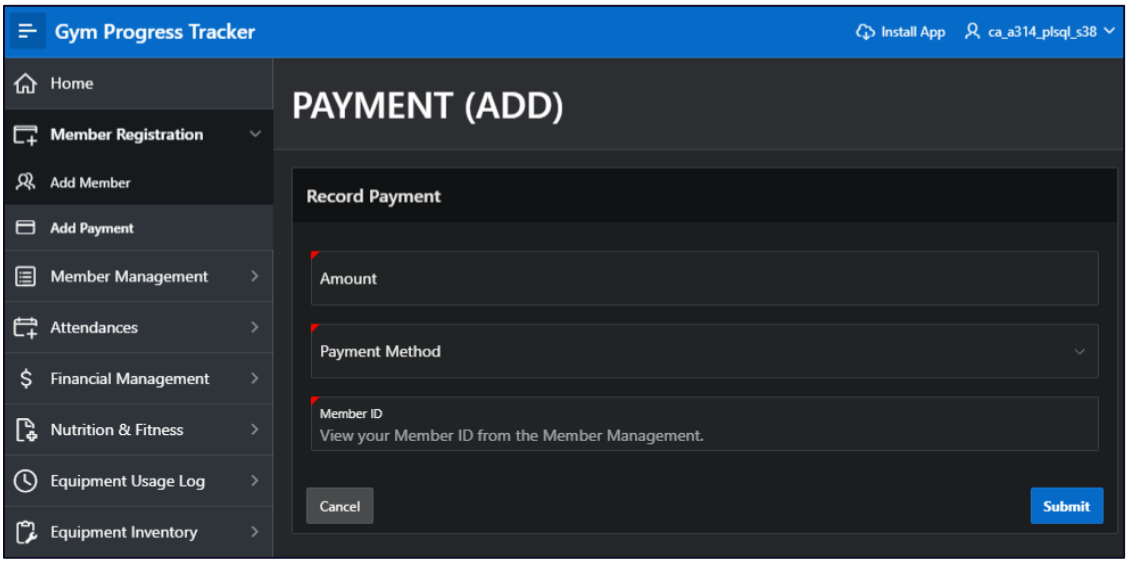


Figure 25. Add Payment Form of the Gym Progress Tracker Project.

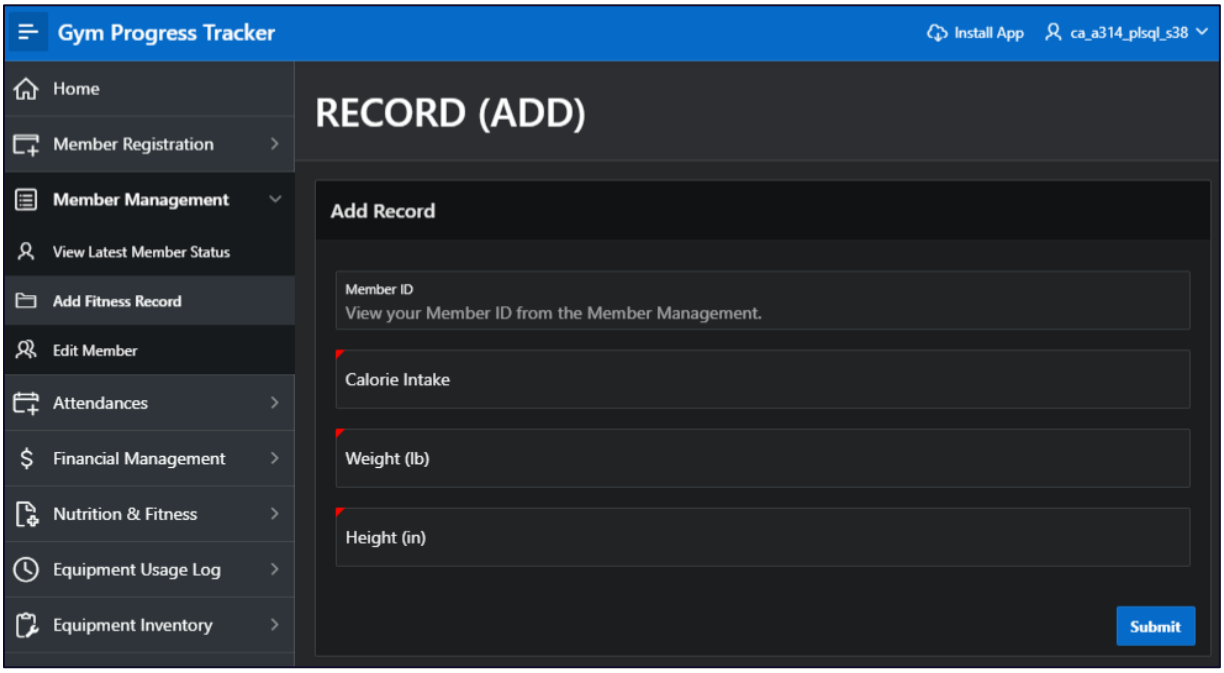


Figure 26. Add Fitness Record Form of the Gym Progress Tracker Project.

Also, modal dialog is used to a number of entities to display important information or request user input without navigating away from the current page. It overlays the main content, focusing the user's attention on the dialog box. Figures 27 to 29 uses modal dialog to the affected entities.

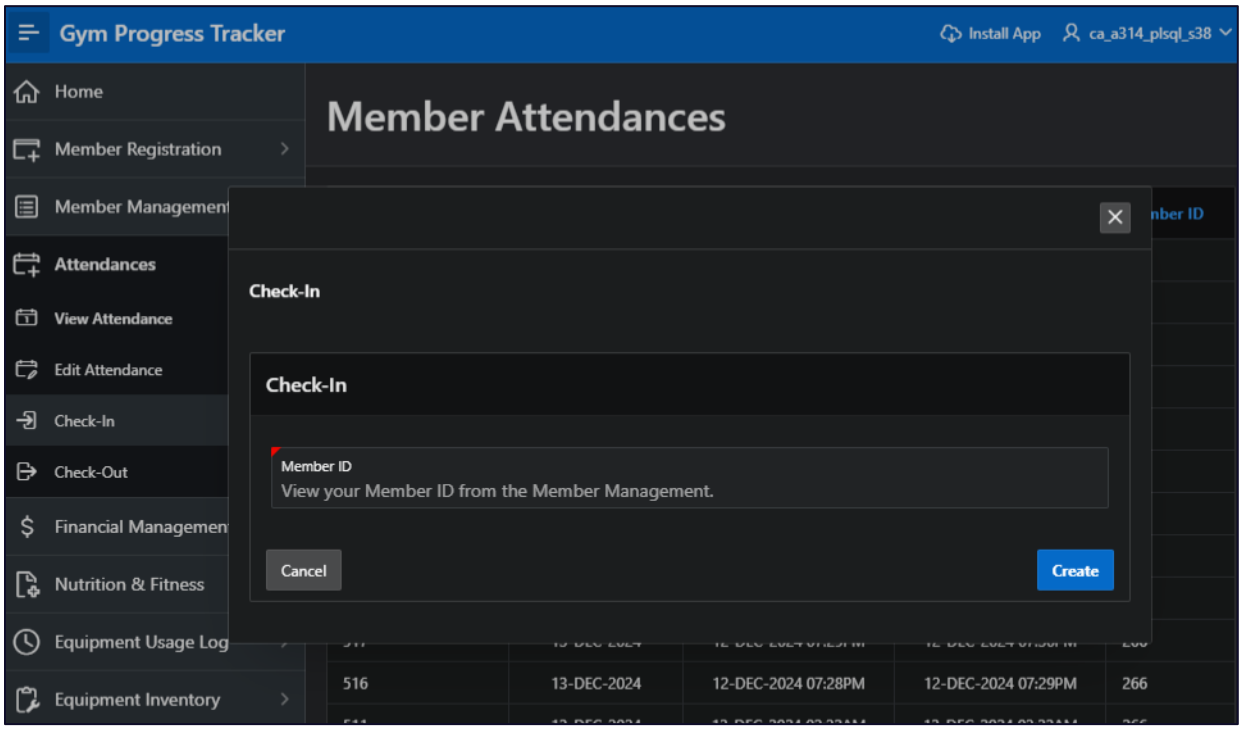


Figure 27. Check-in Form (Modal Dialog) of the Gym Progress Tracker Project.

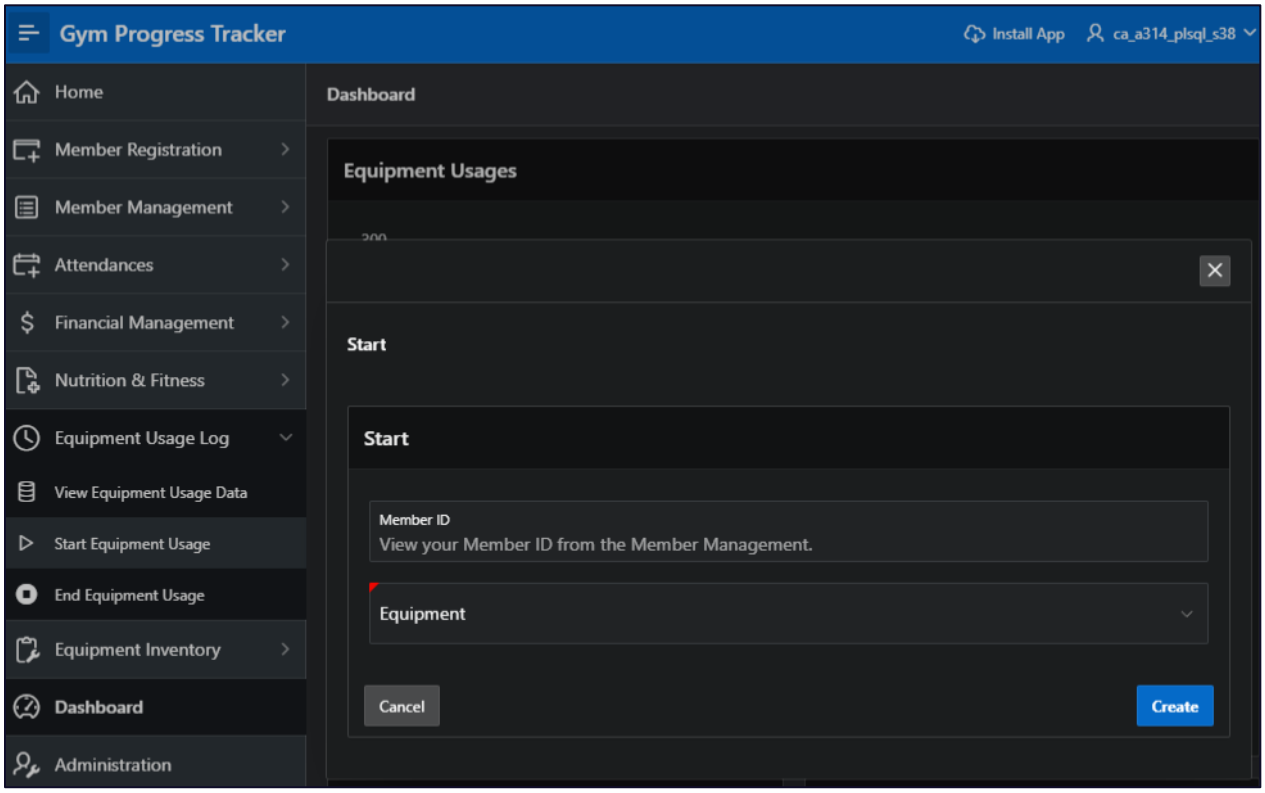


Figure 28. Start Equipment Usage Form (Modal Dialog) of the Gym Progress Tracker Project.

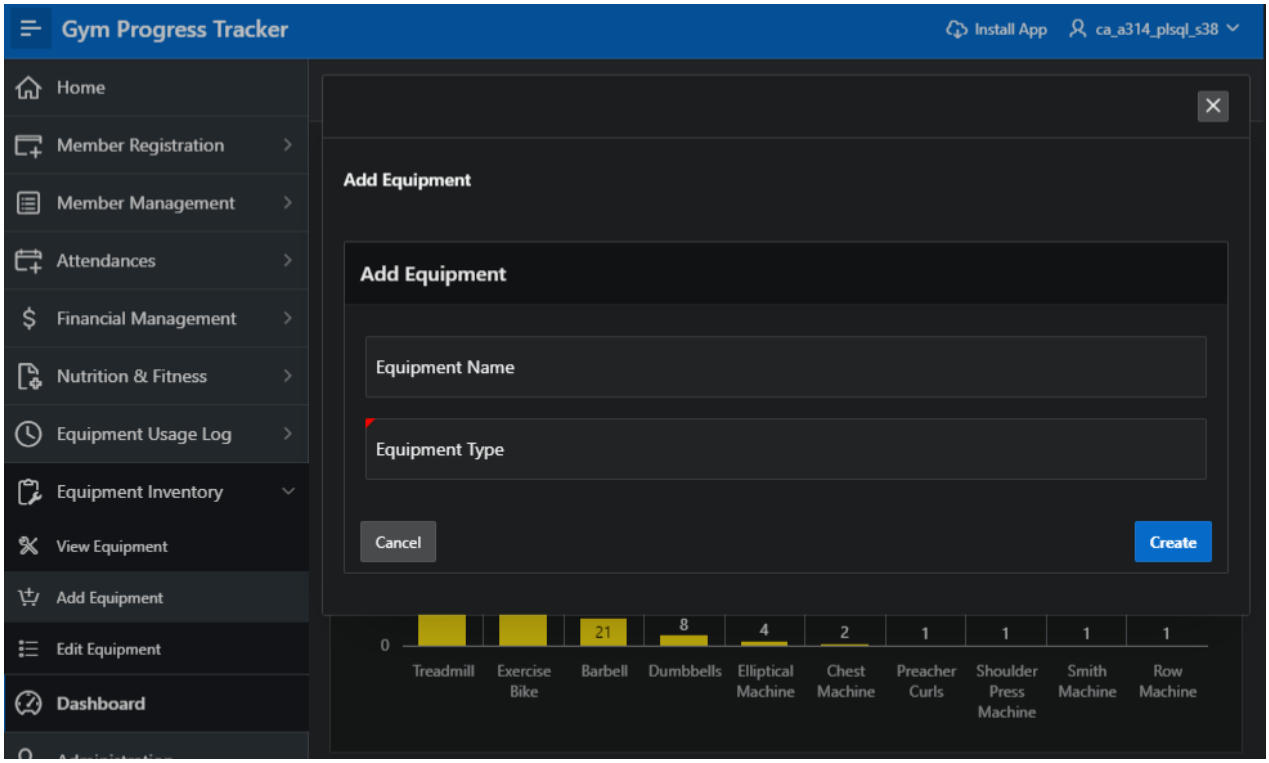


Figure 29. Add Equipment Form (Modal Dialog) of the Gym Progress Tracker Project.

4.3 Form Validations

This project uses form validation to ensure that the data entered by users meets specific criteria before being submitted to the database. It helps maintain data integrity by preventing invalid or incomplete entries. To be more specific, form validations were implemented to verify required fields, enforce email and phone number formats. These validations provide immediate feedback to users, improving the user experience and ensuring accurate data is stored in the database.

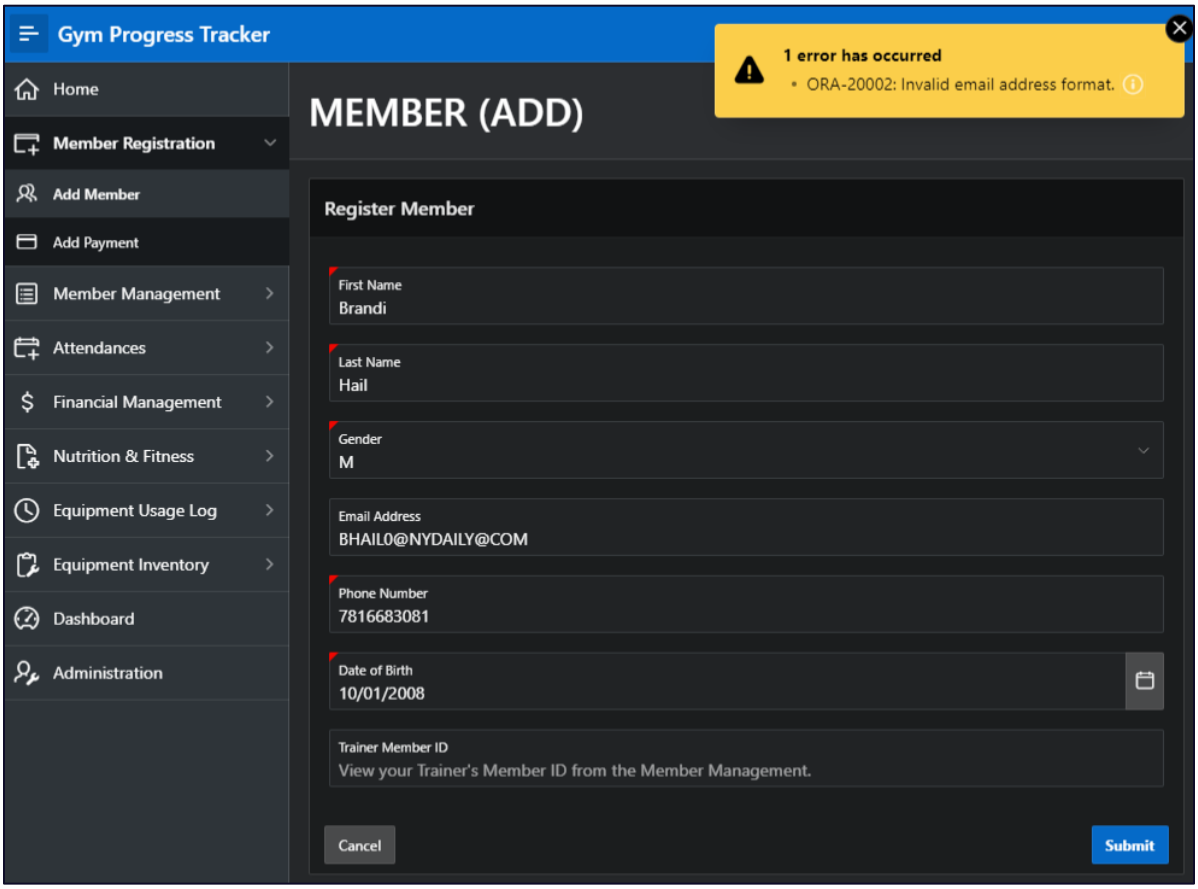


Figure 30. Form Validation for the Email Address Format (Invalid format: Double @ symbol).

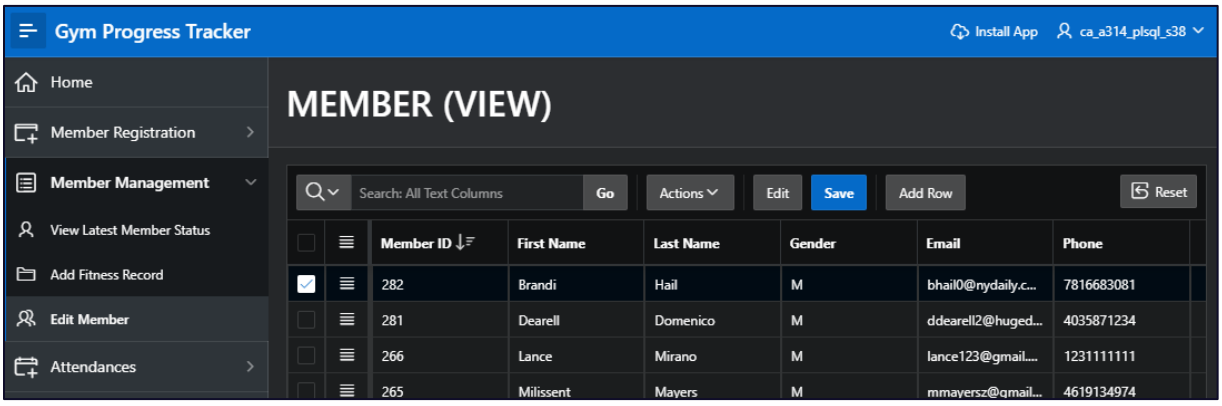


Figure 31. Form Validation for the Email Address Format (Automatically adjusted to lowercase).

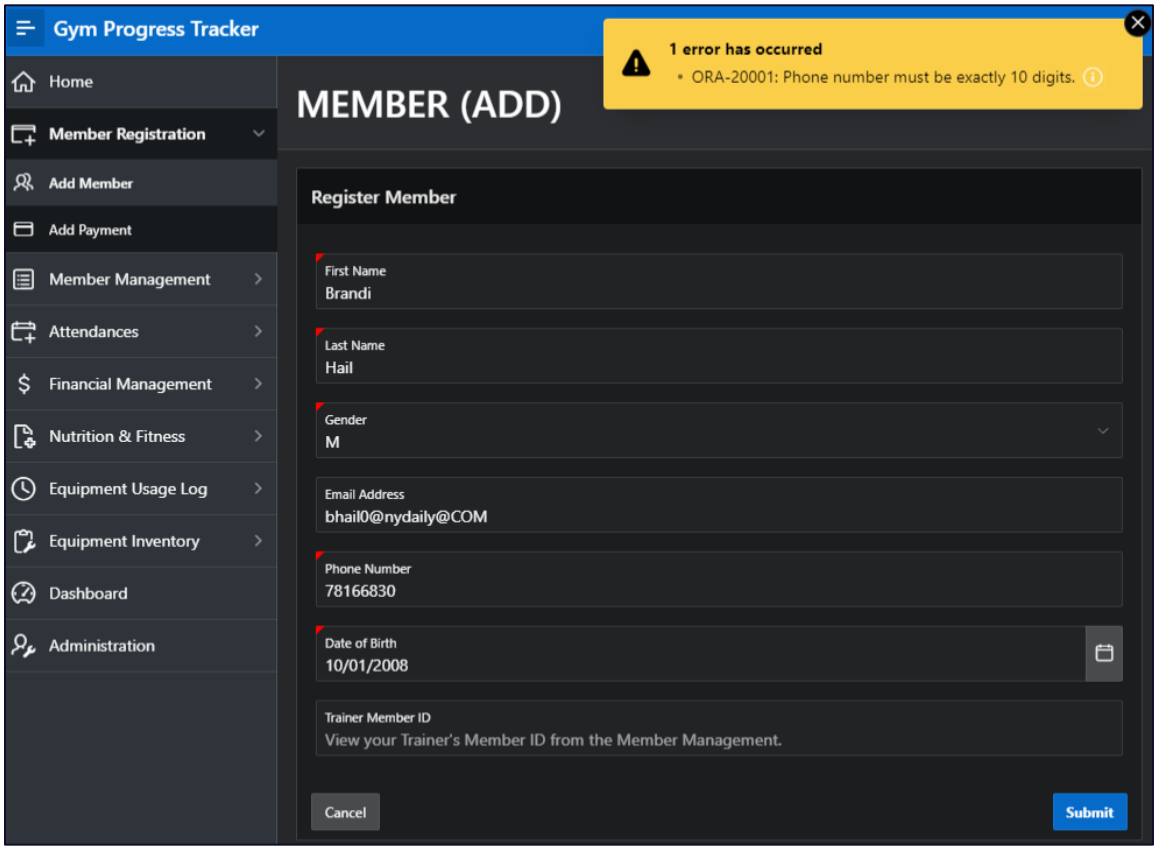


Figure 32. Form Validation for the Phone Number (Not 10 digits).

4.4 Reports & Charts Development

This section shows the report data, charts and graphs to display and give insights about the database project. A classic report was created for three of the tables to present detailed records in a structured format, which are shown in Figures 33 to 35. Interactive reports were also developed for other three tables, offering dynamic filtering, sorting, and searching capabilities in Figures 36 to 38.

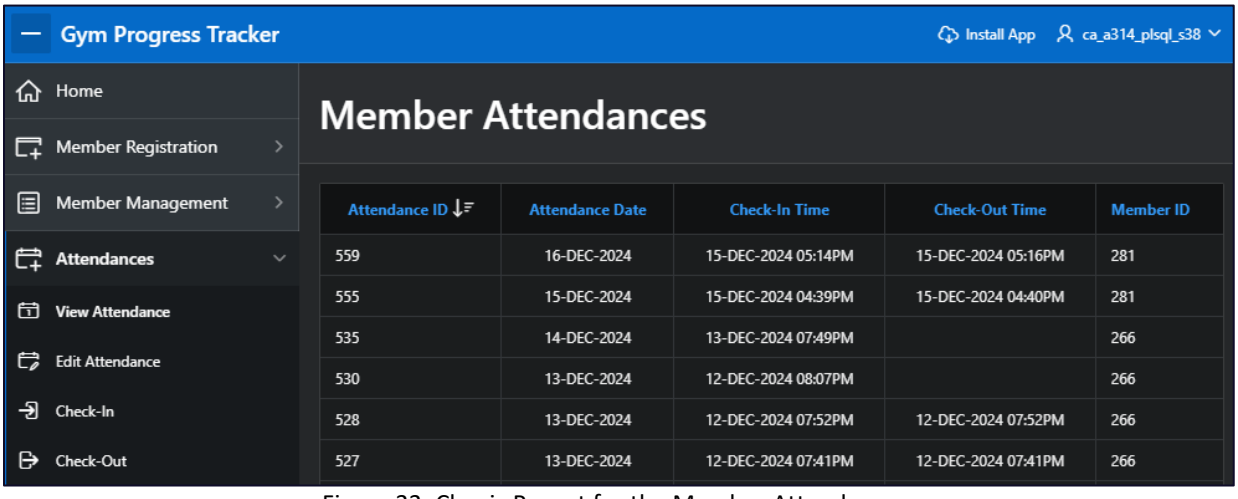


Figure 33. Classic Report for the Member Attendances.

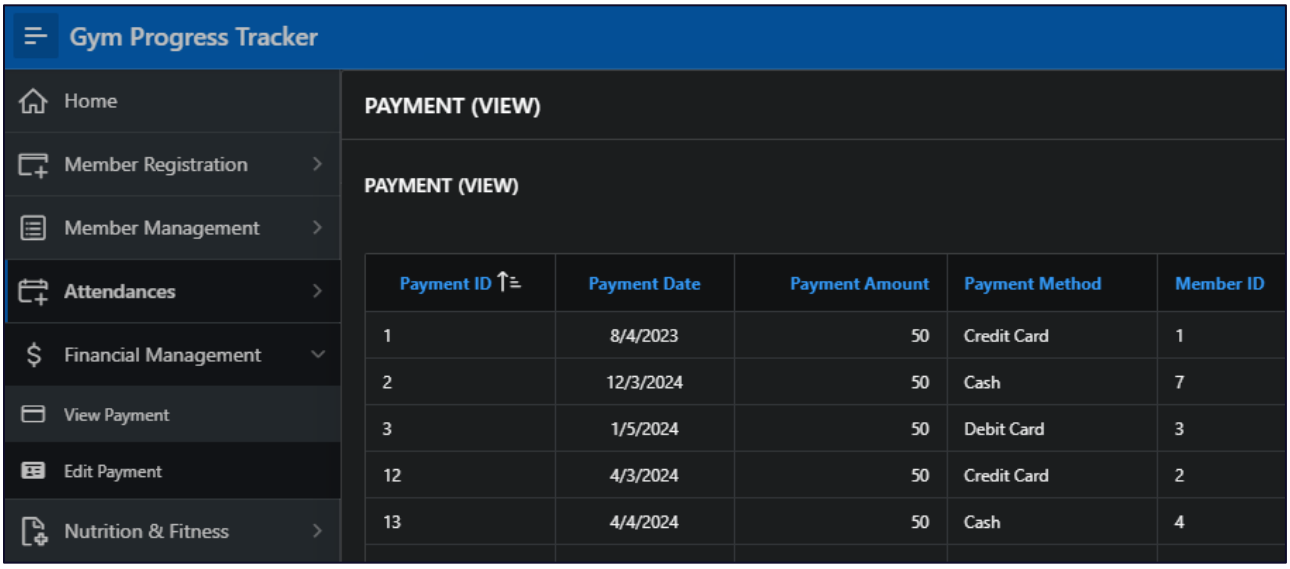


Figure 34. Classic Report for the Payment Records.

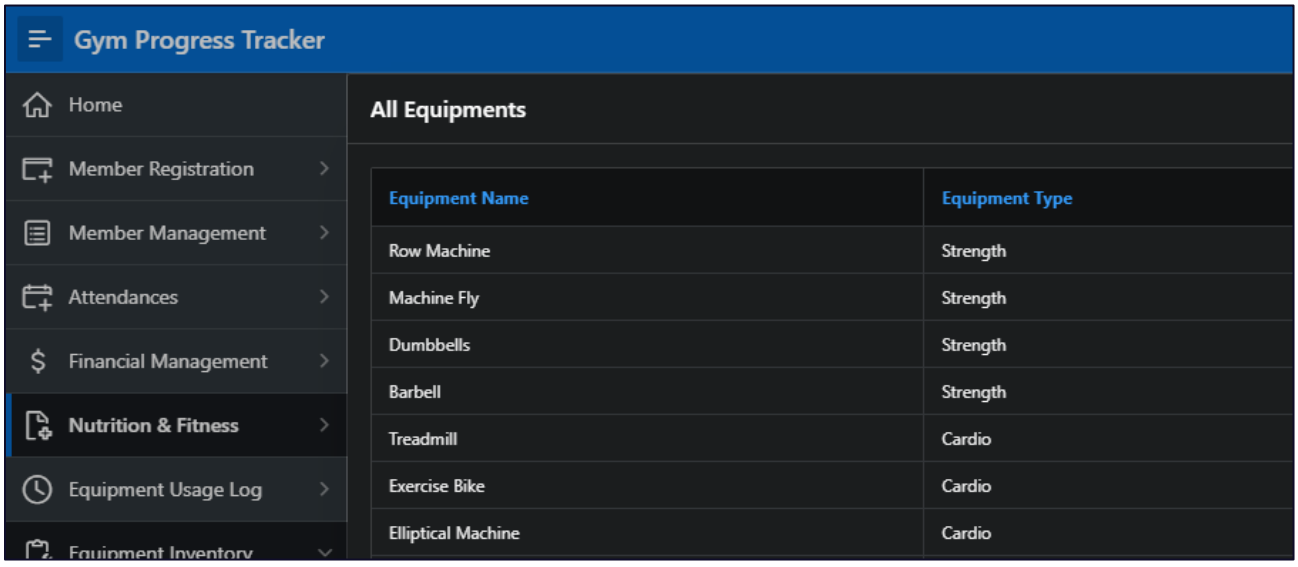


Figure 35. Classic Report for the Equipments.

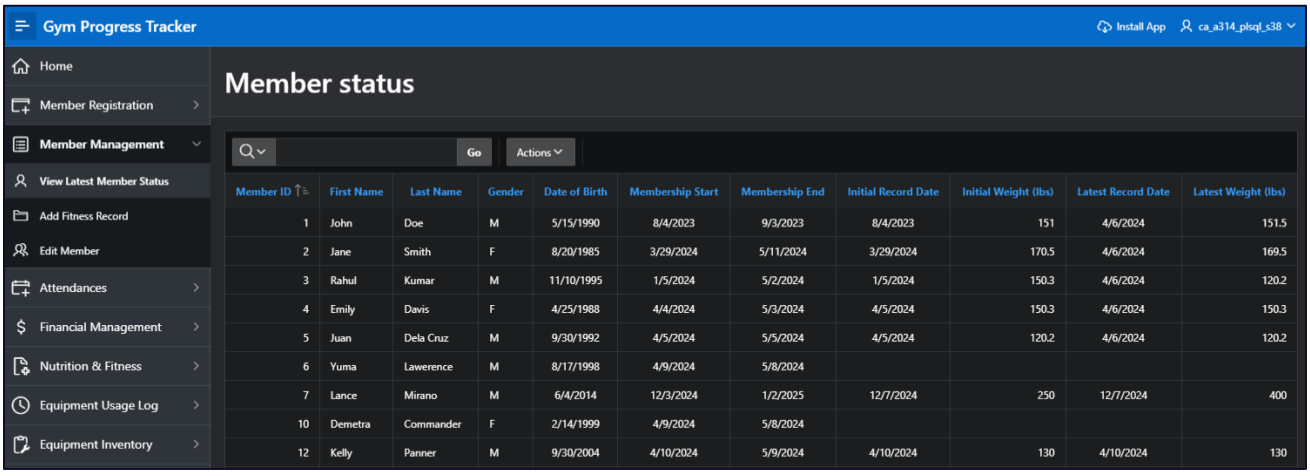


Figure 36. Interactive Report for the Members status.

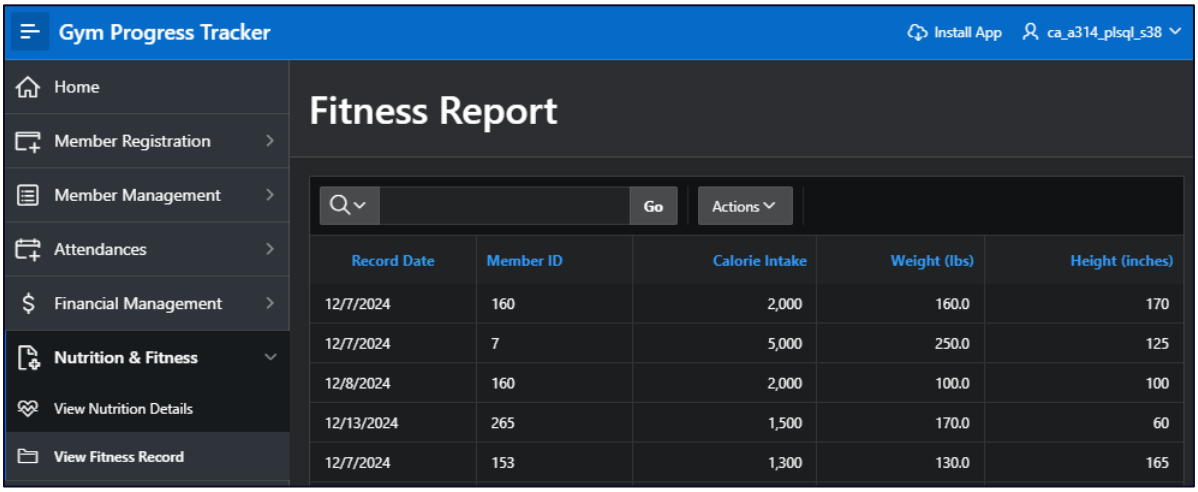


Figure 37. Interactive Report for the Fitness Record.

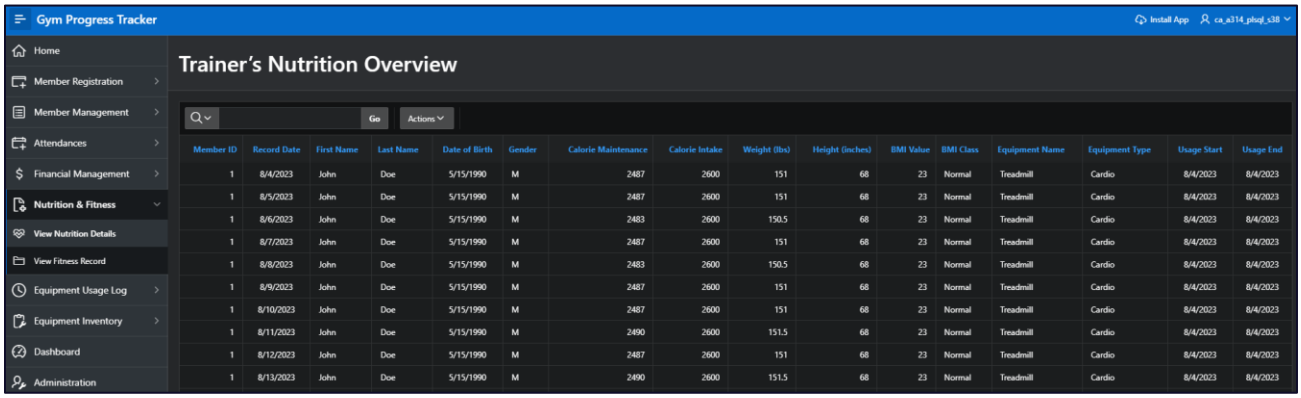


Figure 38. Interactive Report for the Trainer Nutrition Overview.

In addition to the above, a variety of graphs and charts were implemented to visualize data trends and patterns. These include a line graph to track a member's weight trends over time, bar graphs showcasing monthly and quarterly attendance, and equipment usage statistics, and pie charts representing membership types and BMI groups. Figures 39 to 42 shows the different graphs and charts of the Gym Progress Tracker project.

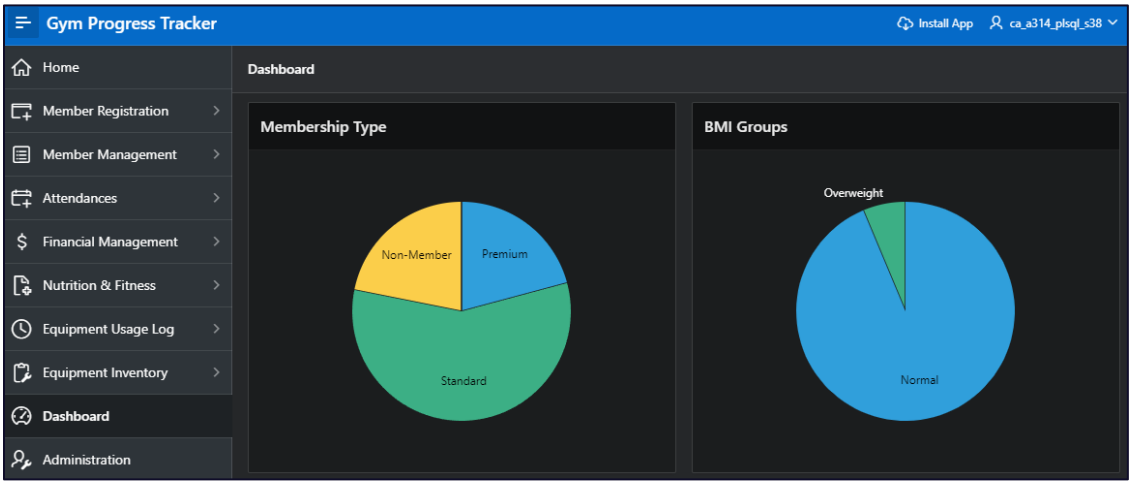


Figure 39. Pie Charts for the Membership Type and BMI Groups.

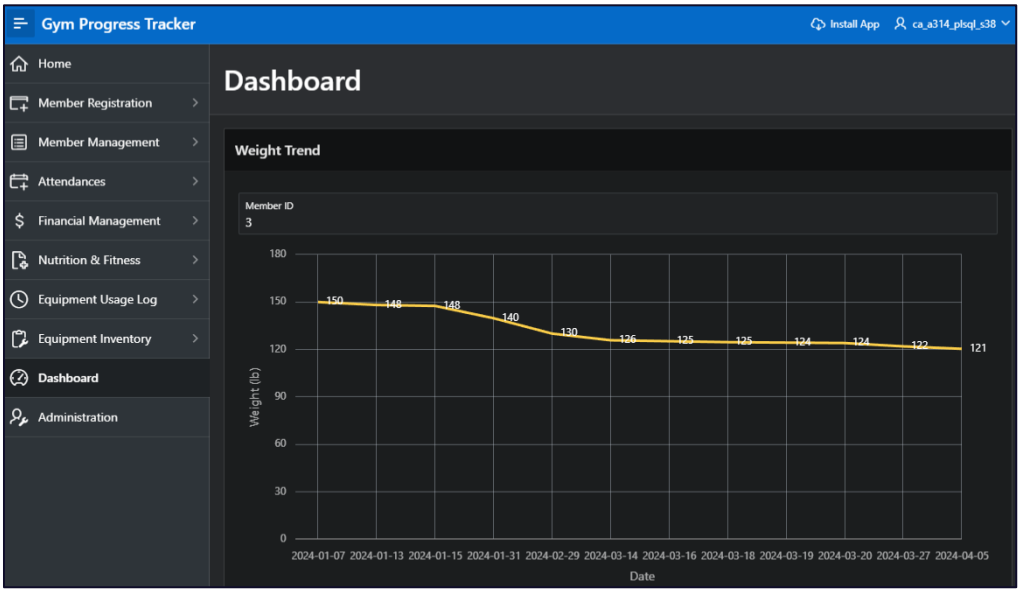


Figure 40. Line Graph for the Weight Trend of a given Member ID.

Figure 41. Bar Graph of the Monthly and Quarterly Attendance.

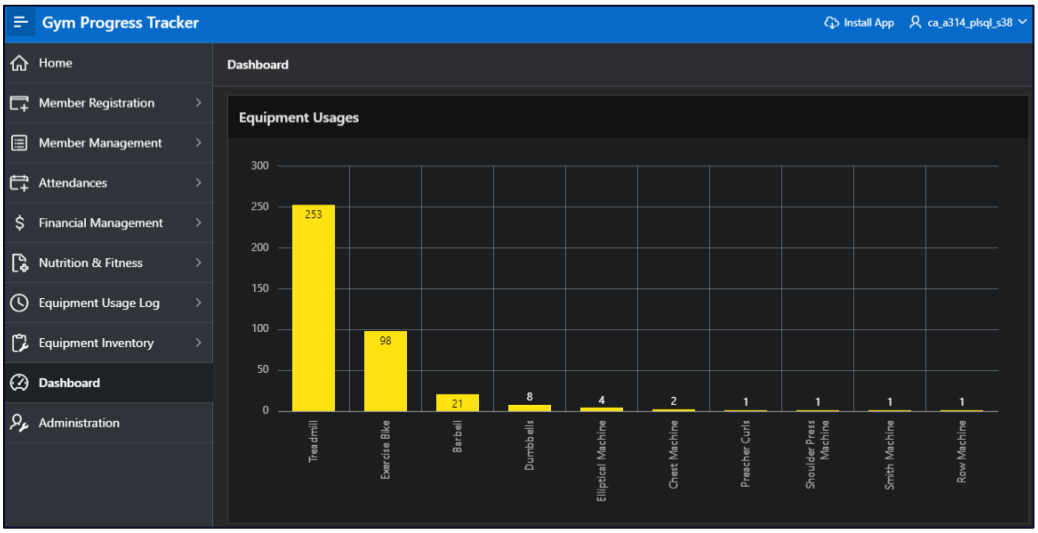


Figure 42. Bar Graph of the Equipment Usages Count per Equipment.

4.5 Security Implementation

The Security Implementation ensures that only authorized users can access the application and perform specific actions. A dedicated Staff table, which contains the Administrator, Finance, Member, and Trainer, was created to manage user roles and permissions as shown in the Figure 43 below. Additionally, an administration page was developed where the admin can assign or modify user roles, granting appropriate access rights.

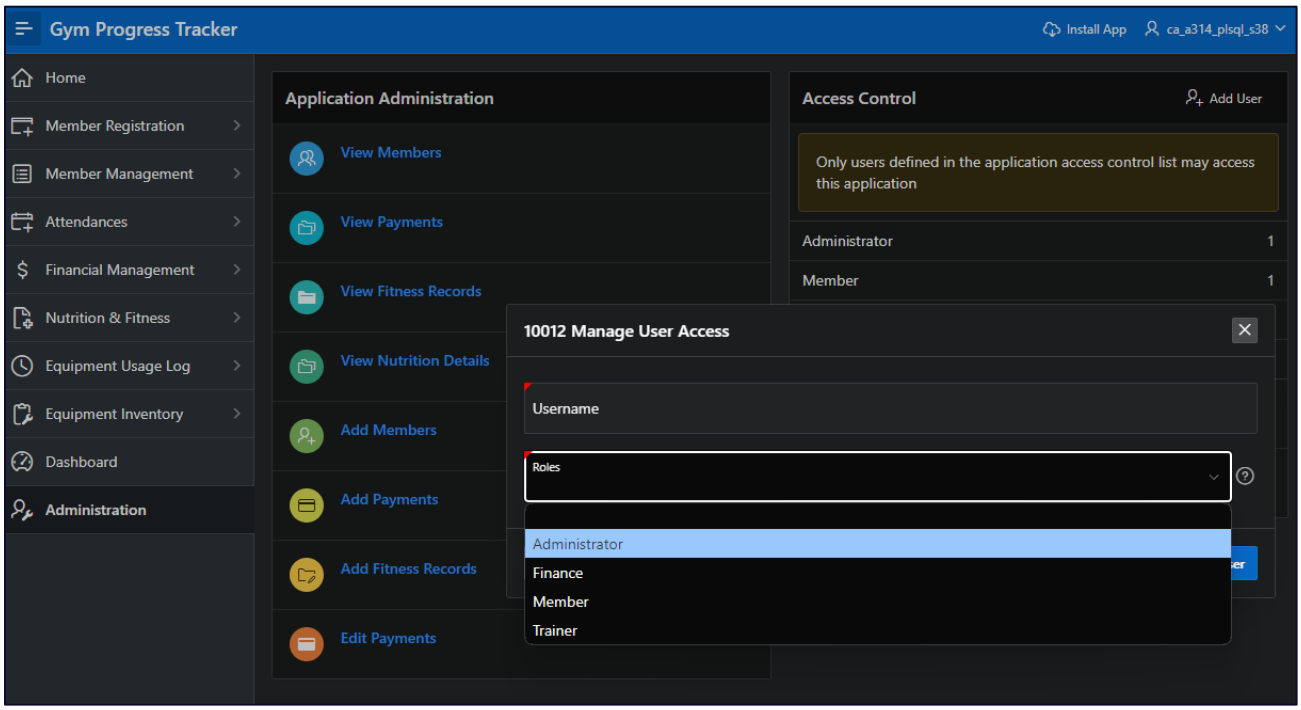


Figure 43. Manage User Access Modal Dialog in the Administration Section.

4.6 Exporting the Project

The project has been exported as a separate SQL file named gym_progress_apex.sql. This file contains all the application settings, configurations, and metadata required to recreate the Oracle APEX application. It can be used to deploy the project on another APEX workspace. Please see the attached file alongside with this report.

5. Conclusion

This project successfully implemented a relational database system for gym fitness centres. The authors analyzed the business requirements and translated them into a functional relational database system. First, the business rules were transformed into an Entity-Relationship Diagram (ERD). This model was then converted into a database structure using SQL Data Definition Language (DDL) statements. Database objects were created including six (6) tables, three (3) functions, six (6) views, seven (7) stored procedures, and three (3) triggers. Additionally, four (4) user roles—Admin, Member, Trainer, and Finance Staff—were established. Finally, the database records were displayed using a front-end interface developed with Oracle APEX.

The implementation phase was a significant learning experience for the authors especially in writing SQL scripts for creating tables, defining constraints, and establishing relationships between entities. Incorporating primary keys, foreign keys, and check constraints demonstrated the importance of maintaining data integrity and ensuring realistic, enforceable business rules within the database.

Developing stored procedures, functions, and triggers further enhanced the authors' understanding of database programming. These features not only automated repetitive tasks but also added efficiency and reliability to the database. Additionally, creating views and reports showcased the ability to present data in meaningful ways, while the use of Oracle APEX for the front-end development demonstrated the importance of integrating a user-friendly interface with the back-end system.

Overall, this project was a comprehensive exercise that strengthened technical skills in database management, problem-solving, and system design. It highlighted the importance of careful planning, attention to detail, and the application of theoretical knowledge to real-world scenarios. The experience gained from this project will undoubtedly serve as a foundation for tackling more complex database projects in the future.

Future enhancements could include integrating credit card payment processing for seamless transactions and adding automated notifications for membership renewals or expirations to improve member engagement. Strengthening security with multi-factor authentication (MFA) would ensure better protection of sensitive data. Additionally, a mobile application is recommended for ease-of-access and to further boost member engagement.

References

[1] Oracle Academy, “3-3 Speaking ERDish & Drawing Relationships,” Presented by Dr. Tufail, Donald School of Business, Science and Computing, Red Deer Polytechnic, Red Deer, Alberta, Canada, 2024.

[2] “Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel,” Mockaroo.com, 2013. https://mockaroo.com/

[3] M. Tufail, “Database Objects I,” Donald School of Business, Science and Computing, Red Deer Polytechnic, Red Deer, Alberta, Canada, 2024.

[4] M. Tufail, “Database Objects II,” Donald School of Business, Science and Computing, Red Deer Polytechnic, Red Deer, Alberta, Canada, 2024.

[5] Oracle Academy, “9-1 Creating Functions,” Presented by Dr. Tufail, Donald School of Business, Science and Computing, Red Deer Polytechnic, Red Deer, Alberta, Canada, 2024.

[6] “Maintenance Calorie Calculator,” Inch Calculator, Apr. 07, 2023. https://www.inchcalculator.com/maintenance-calorie-calculator

[7] “BMI Calculator,” Inch Calculator. https://www.inchcalculator.com/bmi-calculator

[8] Oracle Academy, “8-1 Creating Procedures,” Presented by Dr. Tufail, Donald School of Business, Science and Computing, Red Deer Polytechnic, Red Deer, Alberta, Canada, 2024.

[9] “FROM_TZ,” Oracle Help Center, 2019. https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/FROM_TZ.html

[10] “FROM_TZ() Function in Oracle,” Database.guide, Aug. 12, 2021. https://database.guide/from_tz-function-in-oracle