Logo

Description automatically generated

University of Bahrain

College of Information Technology

Department of Computer Science

ITCS 496: PHYSICAL IMPLEMENTATION OF DATABASES

Section 01

**Gym Management Database System**

**Group Members:**

1. Maha Mohammed Ali: 202002565
2. Musherah Moqbel Ali :202002276
3. Noor Jaafar Ali Hasan: 202007922
4. Maryam Emaduddin: 202002021
5. Marram Hussain: 20207755

**Introduction**

In the dynamic sweep of fitness and wellness, the efficient management of gym operations is essential for the success and sustainability of fitness centers. MyGYM, located in Bahrain, currently relies on a manual system to record member information, manage fitness classes, and handle other critical aspects of its business operations. Recognizing the need for modernization and improved efficiency, the Gym Management System (GMS) has been conceived and developed to address the challenges faced by MyGYM in its day-to-day operations.

The Gym Management System is a broad solution designed to automate and streamline the administrative and managerial tasks inherent to a fitness center. By leveraging technology, MyGYM aims to enhance member experiences, optimize class scheduling, improve attendance tracking, streamline billing processes, and efficiently manage its dedicated staff.

The development and implementation of the Gym Management System are guided by a series of assumptions that lay the preparation for the system's operation. As we dig into the report, we will present an Entity-Relationship (ER) diagram, a mapping from the ER diagram to the relational schema, screen captures of SQL commands, data insertion, and visual representations of forms and reports. These components collectively showcase the integrated approach taken to address the unique needs of MyGYM.

The Gym Management System not only marks a technological crossover for MyGYM but also signifies a commitment to providing a seamless and enhanced experience for both staff and members. Through the automation of routine tasks and the provision of insightful analytics, the system is at ease to become a crucial tool in optimizing operations, fostering member engagement, and driving the sustained growth of MyGYM in the competitive fitness industry.

**Assumptions**

**User Roles:**

* The system assumes the existence of different user roles, such as
  + manager
  + staff
  + members.
* Access to certain features and functionalities is restricted based on the user's role and permissions.

**Member Registration:**

* The users can not register in a class unless he/she are members.

**Membership Plans:**

* The system assumes that membership plans are predefined and do not change frequently during the membership period (fixed).
  + Monthly Gold membership fees will be 40 BD
  + Yearly Gold membership fees will be 4000 BD
  + Monthly Silver membership fees will be 25 BD
  + Yearly Silver membership fees will be 3000 BD
  + Monthly Bronze membership fees will be 30 BD
  + Yearly Bronze membership fees will be 3500 BD

**Class Capacity Enforcement:**

* If the class capacity is full, then the member cannot register in this class.

**Monthly Report Timeframe:**

* The monthly report procedure considers a calendar month (from the 1st to the last day of the month) for generating insights into instructor performance, member enrollments, and profits.

**Instructor availability:**

* The members cannot register for a class unless the fitness class instructors are available during the scheduled class times.

**Attendance Recording:**

* The staff responsible for recording attendance are diligent in accurately marking member attendance for classes and gym visits.

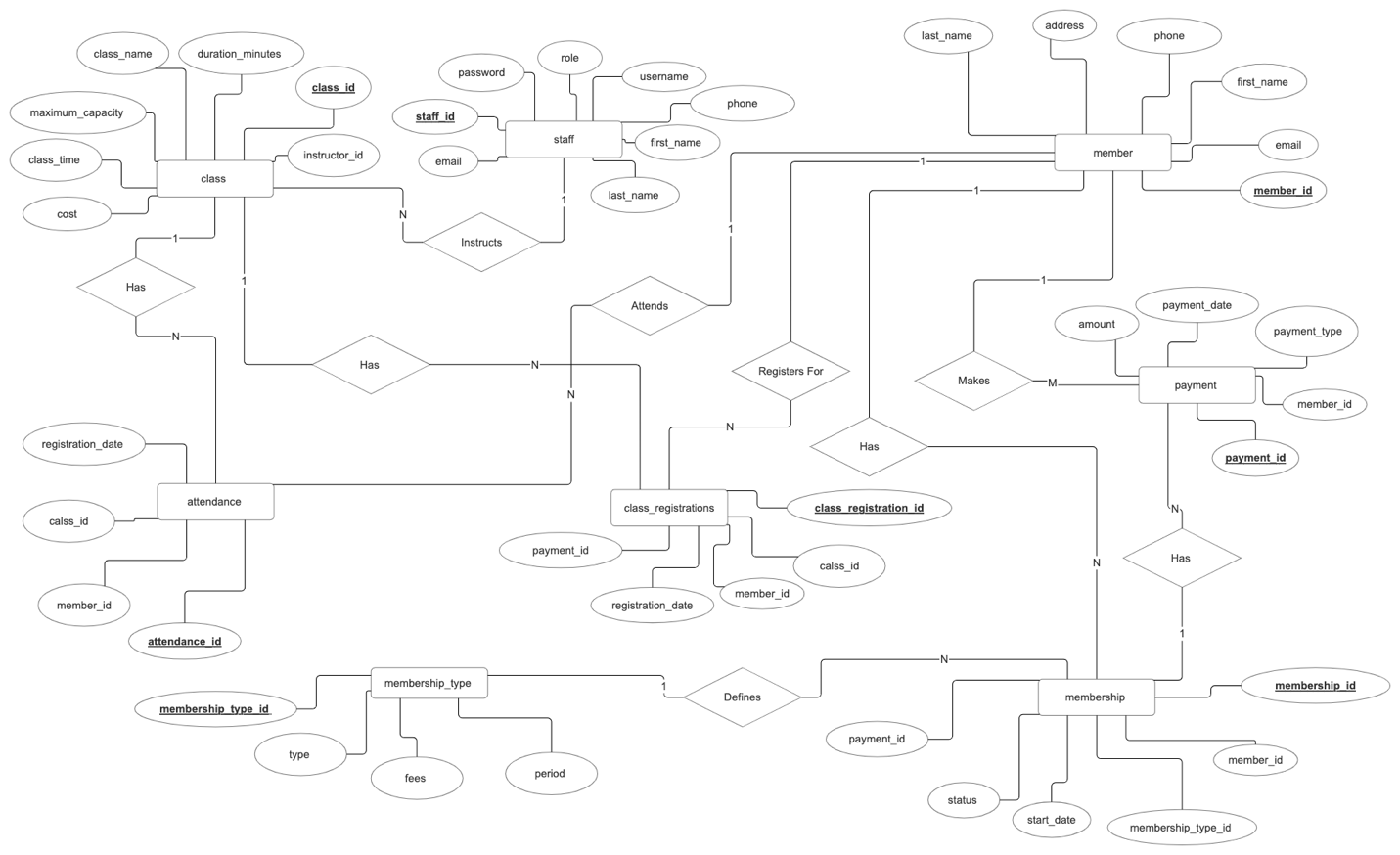
**Class Scheduling:**

* The class schedules are predefined and communicated to members in advance.
* Any changes to the class schedule are communicated promptly to the members.

**Billing and Payments:**

* Members are informed about upcoming payments and renewal dates in a timely manner.

**ER diagram of the system.**



**ERD to relation schema mapping**

**A screenshot of a computer

Description automatically generated**

**SQL commands to create and insert data into the tables along with the screen shots**

CREATE TABLE member (

member\_id INT DEFAULT member\_seq.nextval PRIMARY KEY,

phone VARCHAR(20) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

address VARCHAR(200)

);

A screenshot of a video player

Description automatically generated

CREATE TABLE staff (

staff\_id INT DEFAULT staff\_seq.nextval PRIMARY KEY,

phone VARCHAR(20) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

password VARCHAR(100) NOT NULL,

username VARCHAR(50) UNIQUE NOT NULL,

role VARCHAR(20) NOT NULL

);

A screenshot of a computer

Description automatically generated

CREATE TABLE class (class\_id INT DEFAULT class\_seq.nextval PRIMARY KEY,

class\_name VARCHAR(50) NOT NULL,

maximum\_capacity INT NOT NULL,

class\_time DATE NOT NULL,

duration\_minutes INT NOT NULL,

cost NUMBER NOT NULL,

instructor\_id INT NOT NULL CONSTRAINT class\_staff\_id\_fk REFERENCES staff(staff\_id));

A screenshot of a computer

Description automatically generated

CREATE TABLE membership\_type (

membership\_type\_id INT DEFAULT membership\_type\_seq.nextval PRIMARY KEY,

fees DECIMAL(8) NOT NULL,

period VARCHAR(20) NOT NULL CHECK (period IN ('monthly', 'yearly')),

type VARCHAR(255) NOT NULL CONSTRAINT chk\_membership\_type CHECK (type IN ('Gold', 'Silver', 'Bronze')),

CONSTRAINT chk\_fee CHECK (

(type = 'Gold' AND period = 'monthly' AND fees = 40) OR

(type = 'Gold' AND period = 'yearly' AND fees = 4000) OR

(type = 'Silver' AND period = 'monthly' AND fees = 25) OR

(type = 'Silver' AND period = 'yearly' AND fees = 3000) OR

(type = 'Bronze' AND period = 'monthly' AND fees = 30) OR

(type = 'Bronze' AND period = 'yearly' AND fees = 3500)

)

);

A screenshot of a computer

Description automatically generated

CREATE TABLE membership (

membership\_id INT DEFAULT membership\_seq.nextval PRIMARY KEY,

member\_id INT NOT NULL,

membership\_type\_id INT,

start\_date DATE NOT NULL,

status VARCHAR(50) NOT NULL CONSTRAINT CHK\_MEMBERSHIP\_status CHECK (status IN (' Active', ' Expired', ' Suspended')),

payment\_id INT NOT NULL CONSTRAINT payment\_id\_fk REFERENCES payment(payment\_id),

FOREIGN KEY (member\_id) REFERENCES member(member\_id),

FOREIGN KEY (membership\_type\_id) REFERENCES membership\_type(membership\_type\_id)

);

A screenshot of a computer

Description automatically generated

CREATE TABLE class\_registration (

class\_registration\_id INT DEFAULT class\_registration\_seq.nextval PRIMARY KEY,

registration\_date DATE NOT NULL,

member\_id INT NOT NULL CONSTRAINT class\_reg\_memberid\_fk REFERENCES member(member\_id),

class\_id INT NOT NULL CONSTRAINT classid\_fk REFERENCES class(class\_id),

payment\_id INT NOT NULL CONSTRAINT paymentid\_fk REFERENCES payment(payment\_id)

);

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

CREATE TABLE attendance (

attendance\_id INT DEFAULT member\_attendance\_seq.nextval PRIMARY KEY,

registration\_date DATE NOT NULL,

member\_id INT NOT NULL CONSTRAINT class\_reg\_member\_id\_fk REFERENCES member(member\_id),

class\_id INT NOT NULL CONSTRAINT class\_id\_fk REFERENCES class(class\_id)

);

A screenshot of a computer

Description automatically generated

CREATE TABLE payment (

payment\_id INT DEFAULT payment\_seq.nextval PRIMARY KEY,

payment\_type VARCHAR(50) NOT NULL,

payment\_date DATE NOT NULL,

amount DECIMAL(8) NOT NULL,

member\_id INT NOT NULL CONSTRAINT payment\_member\_id\_fk REFERENCES member(member\_id)

);

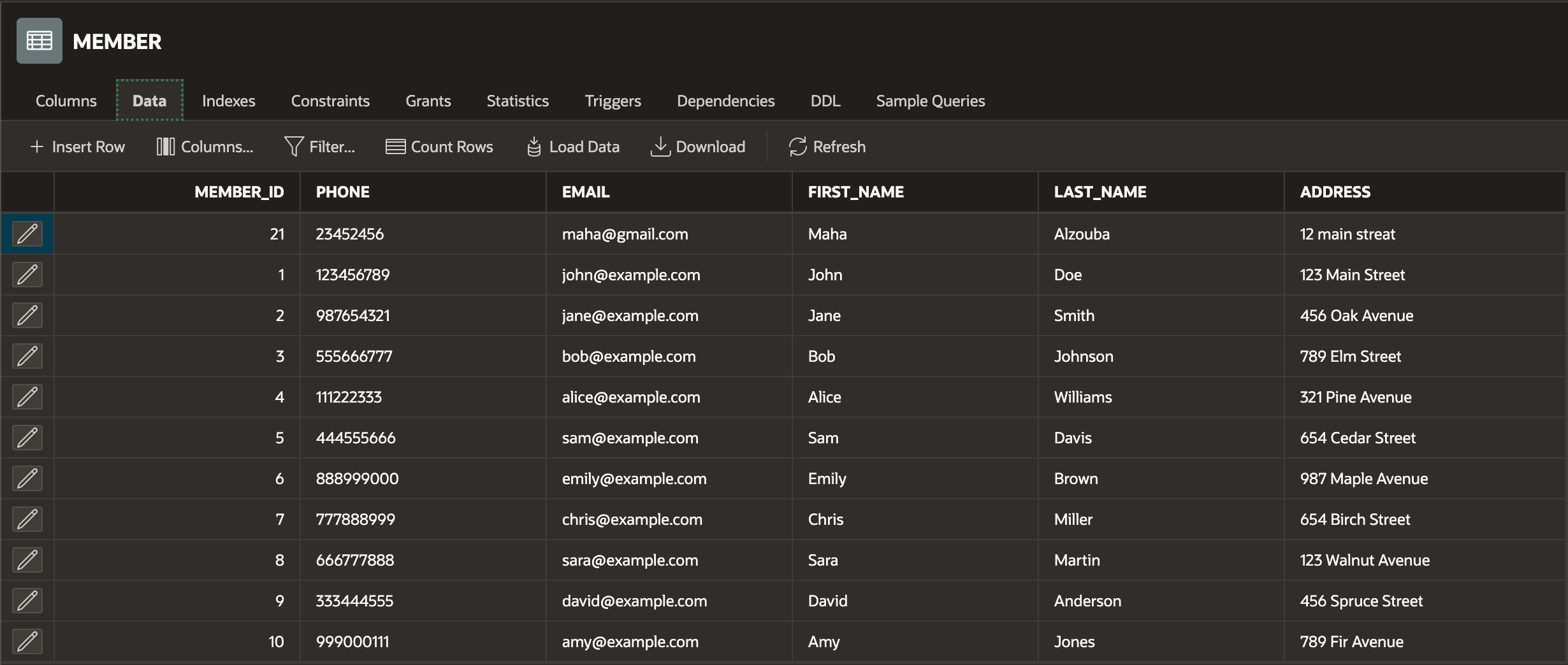
A screenshot of a computer

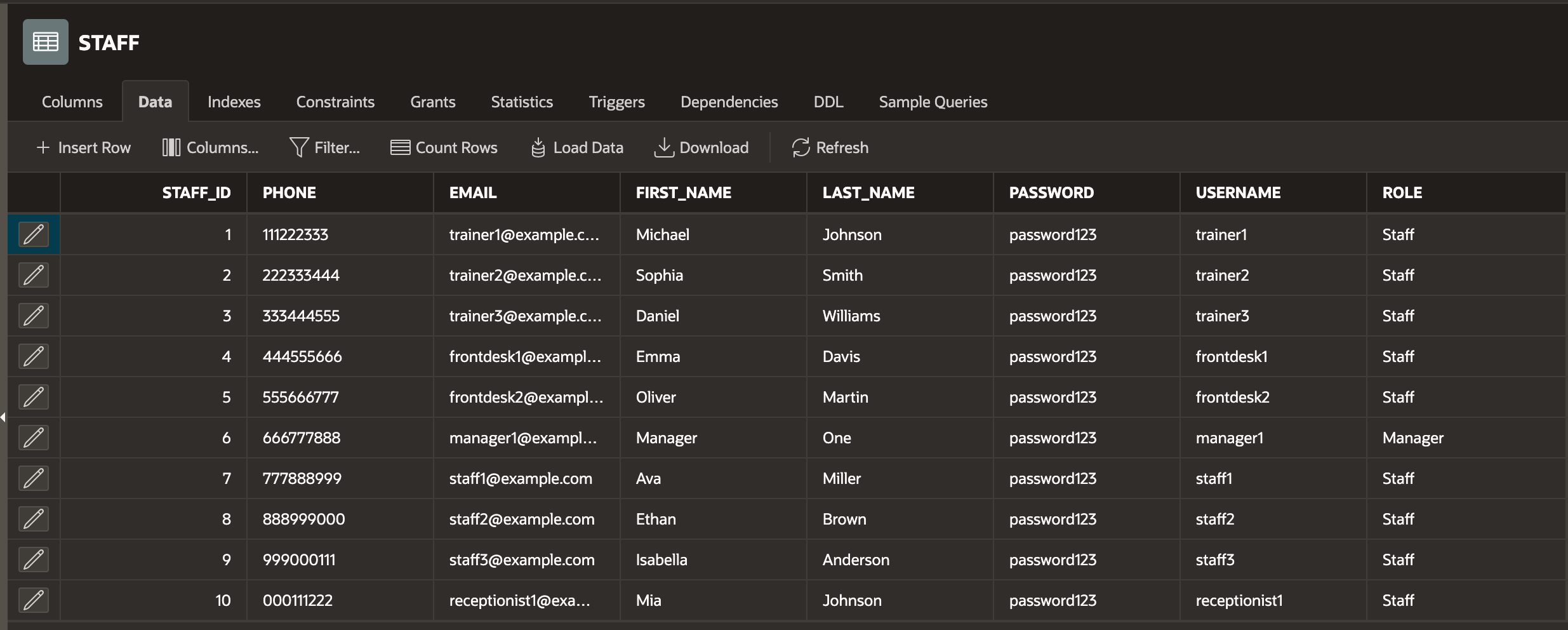
Description automatically generated

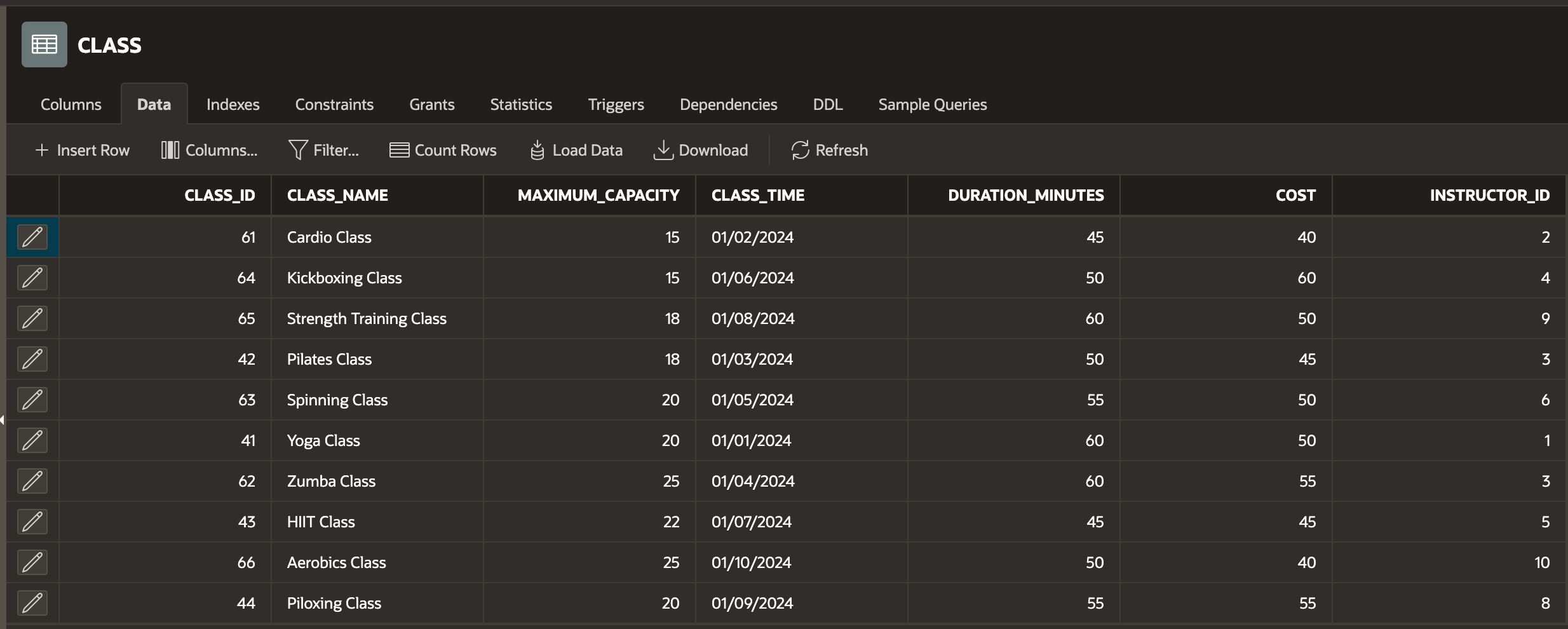
A screenshot of a computer

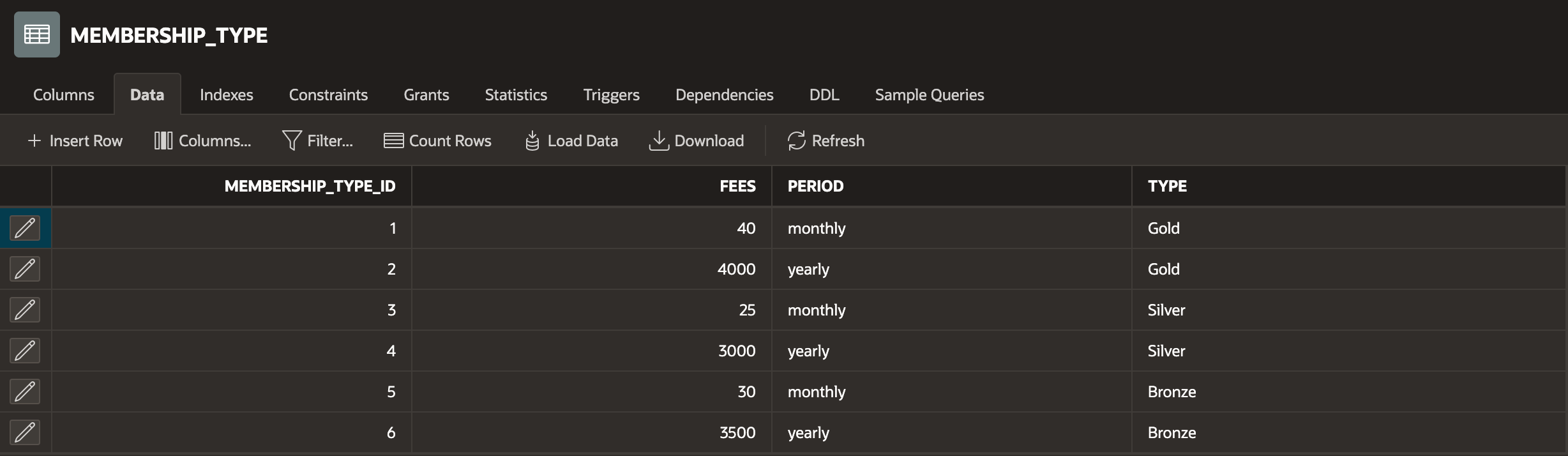
Description automatically generated

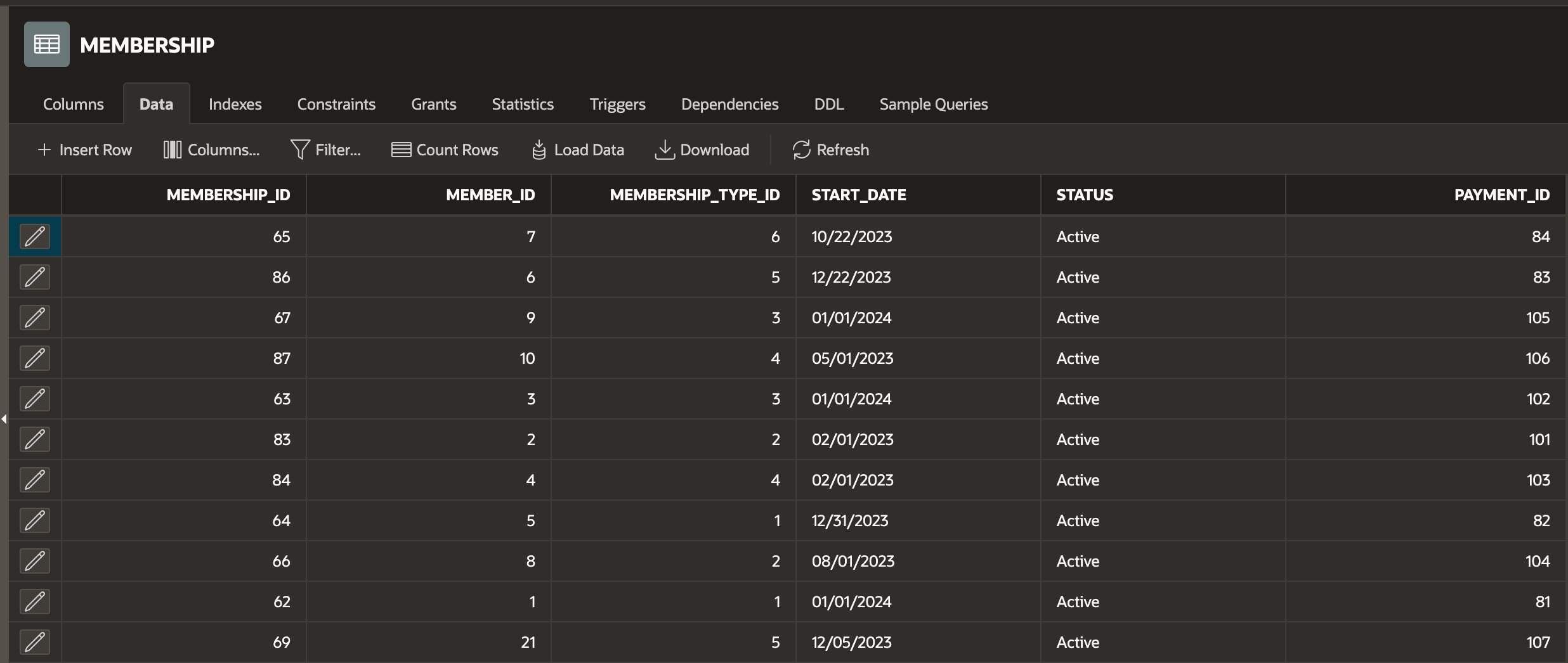
**Data Samples**

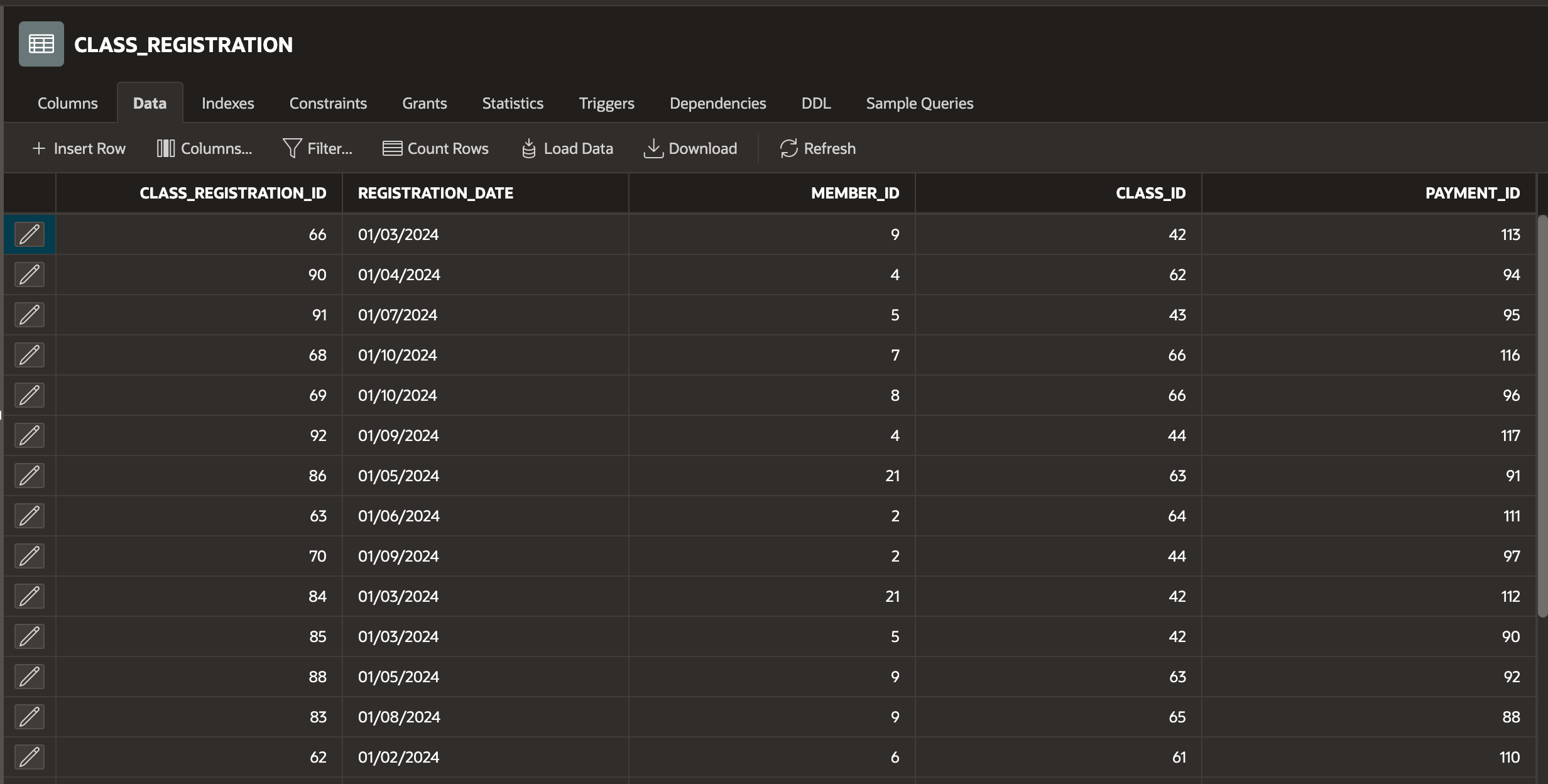
****

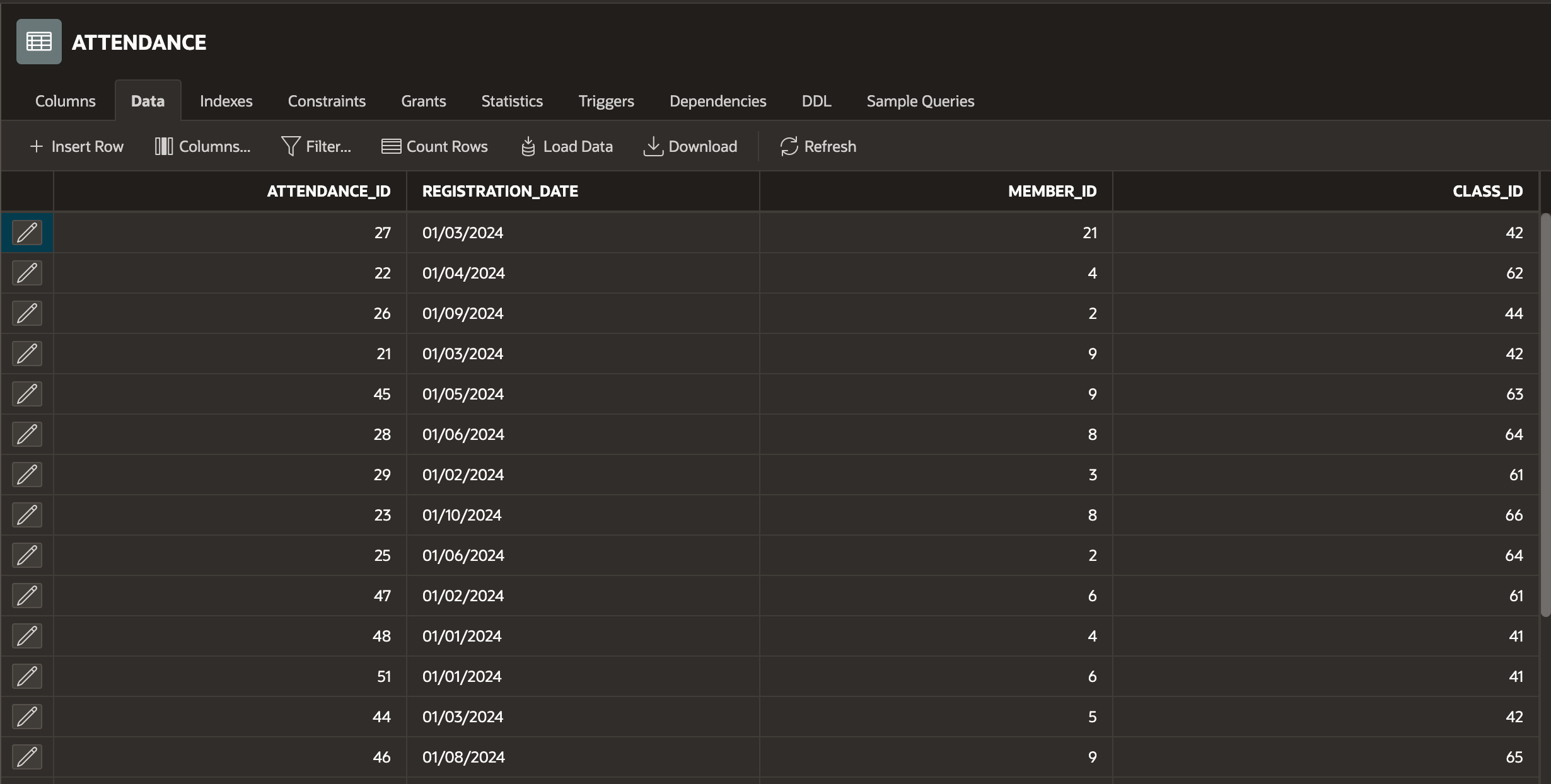
****

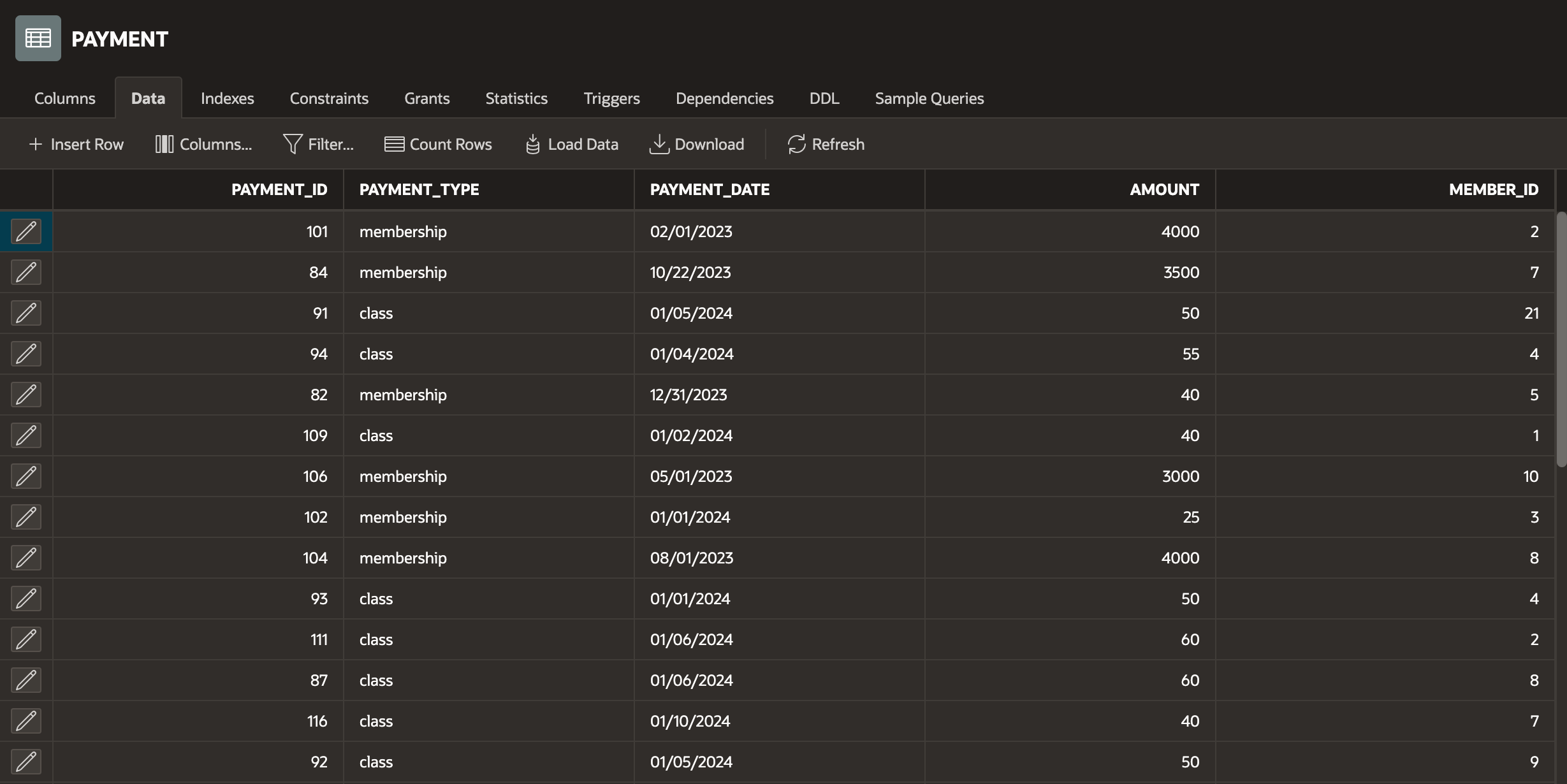
****

****

****

****

****

****

**screen captures of query/procedure results**

CREATE OR REPLACE PROCEDURE MonthlyReport (in\_year NUMBER, in\_month NUMBER) IS

BEGIN

FOR instructor\_info IN (

SELECT s.staff\_id, s.first\_name, s.last\_name, s.phone, s.email, s.role,

COUNT(DISTINCT cr.member\_id) AS num\_members, SUM(c.cost) AS total\_profits

FROM class\_registration cr

JOIN class c ON cr.class\_id = c.class\_id

JOIN staff s ON c.instructor\_id = s.staff\_id

WHERE EXTRACT(YEAR FROM cr.registration\_date) = in\_year

AND EXTRACT(MONTH FROM cr.registration\_date) = in\_month

GROUP BY s.staff\_id, s.first\_name, s.last\_name, s.phone, s.email, s.role

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Instructor ID: ' || instructor\_info.staff\_id);

DBMS\_OUTPUT.PUT\_LINE('Name: ' || instructor\_info.first\_name || ' ' || instructor\_info.last\_name);

DBMS\_OUTPUT.PUT\_LINE('Phone: ' || instructor\_info.phone);

DBMS\_OUTPUT.PUT\_LINE('Email: ' || instructor\_info.email);

DBMS\_OUTPUT.PUT\_LINE('Role: ' || instructor\_info.role);

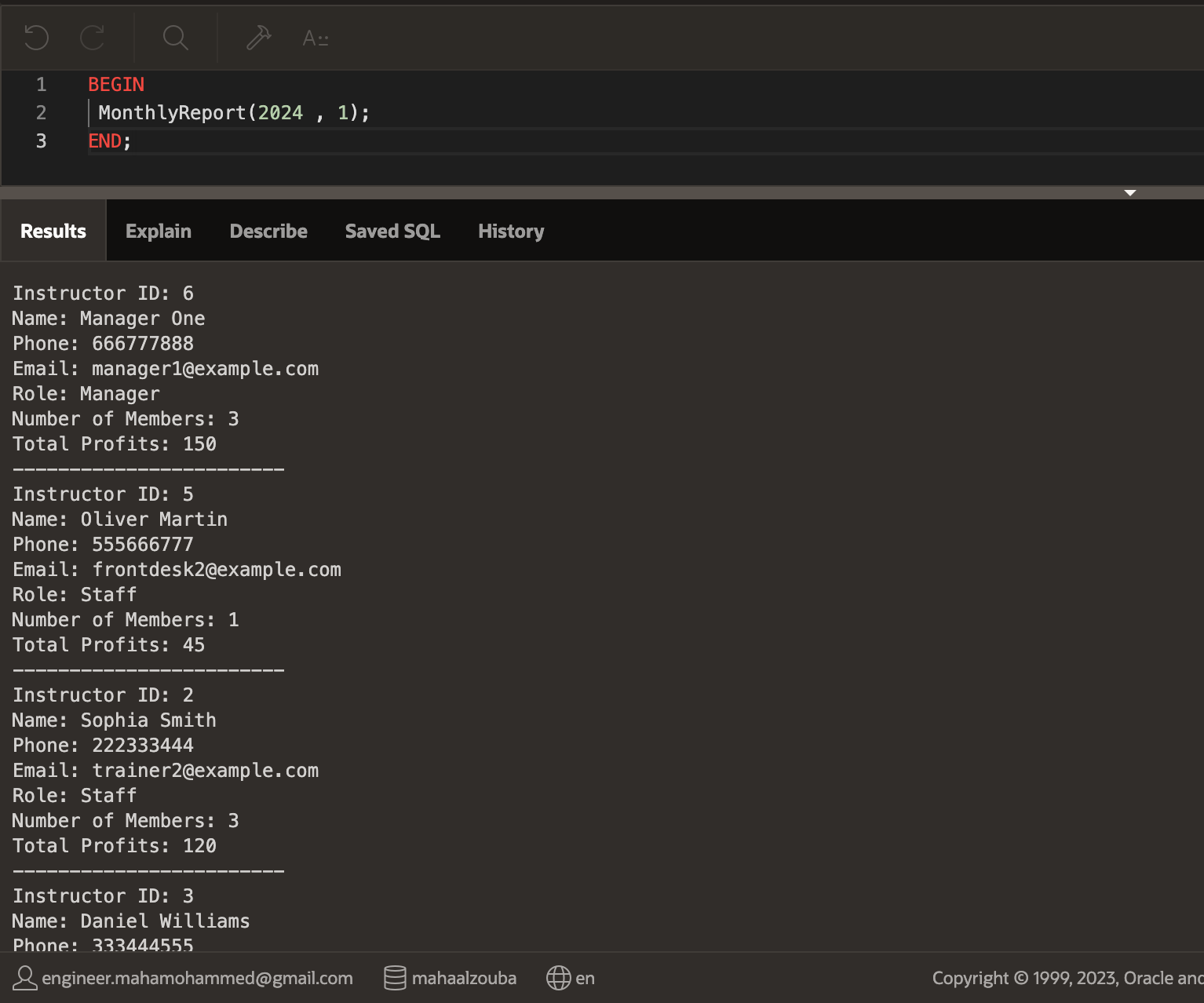
DBMS\_OUTPUT.PUT\_LINE('Number of Members: ' || instructor\_info.num\_members);

DBMS\_OUTPUT.PUT\_LINE('Total Profits: ' || instructor\_info.total\_profits);

DBMS\_OUTPUT.PUT\_LINE('------------------------');

END LOOP;

END MonthlyReport;

****

CREATE OR REPLACE PROCEDURE MemberServices (in\_member\_id INT) IS

BEGIN

FOR member\_info IN (

SELECT m.\*, cr.class\_registration\_id, cr.registration\_date, c.class\_name, c.class\_time, c.duration\_minutes, c.cost

FROM member m

JOIN class\_registration cr ON m.member\_id = cr.member\_id

JOIN class c ON cr.class\_id = c.class\_id

WHERE m.member\_id = in\_member\_id

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Member ID: ' || member\_info.member\_id);

DBMS\_OUTPUT.PUT\_LINE('Phone: ' || member\_info.phone);

DBMS\_OUTPUT.PUT\_LINE('Email: ' || member\_info.email);

DBMS\_OUTPUT.PUT\_LINE('Address: ' || member\_info.address);

DBMS\_OUTPUT.PUT\_LINE('Class Registration ID: ' || member\_info.class\_registration\_id);

DBMS\_OUTPUT.PUT\_LINE('Registration Date: ' || TO\_CHAR(member\_info.registration\_date, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Class Name: ' || member\_info.class\_name);

DBMS\_OUTPUT.PUT\_LINE('Class Time: ' || TO\_CHAR(member\_info.class\_time, 'YYYY-MM-DD HH24:MI:SS'));

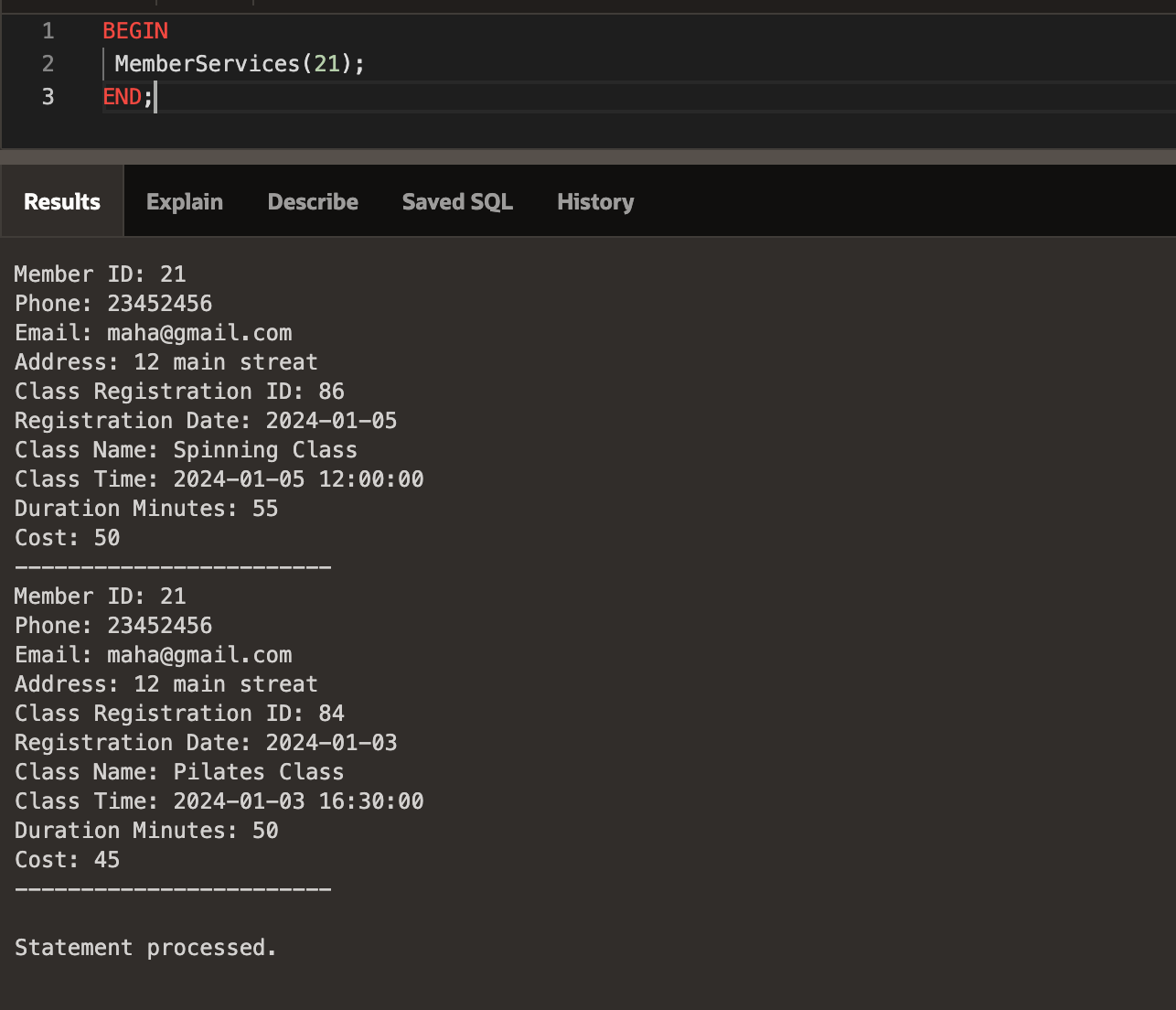
DBMS\_OUTPUT.PUT\_LINE('Duration Minutes: ' || member\_info.duration\_minutes);

DBMS\_OUTPUT.PUT\_LINE('Cost: ' || member\_info.cost);

DBMS\_OUTPUT.PUT\_LINE('------------------------');

END LOOP;

END MemberServices;

****

CREATE OR REPLACE PROCEDURE AddMember (

p\_phone VARCHAR2,

p\_email VARCHAR2,

p\_first\_name VARCHAR2,

p\_last\_name VARCHAR2,

p\_address VARCHAR2,

p\_membership\_type VARCHAR2,

p\_membership\_period VARCHAR2

) IS

v\_member\_id INT;

v\_membership\_type\_id INT;

v\_payment\_id\_membership INT;

v\_payment\_id\_payment INT;

v\_membership\_fee DECIMAL(8);

BEGIN

-- Step 1: Insert into member table

INSERT INTO member (member\_id, phone, email, first\_name, last\_name, address)

VALUES (member\_seq.nextval, p\_phone, p\_email, p\_first\_name, p\_last\_name, p\_address)

RETURNING member\_id INTO v\_member\_id;

-- Step 2: Get membership type ID and fee

SELECT membership\_type\_id, fees

INTO v\_membership\_type\_id, v\_membership\_fee

FROM membership\_type

WHERE type = p\_membership\_type AND period = p\_membership\_period;

-- Step 3: Insert into membership table without payment\_id

INSERT INTO membership (membership\_id, member\_id, membership\_type\_id, start\_date, status, payment\_id)

VALUES (membership\_seq.nextval, v\_member\_id, v\_membership\_type\_id, SYSDATE, ' Active', NULL)

RETURNING payment\_id INTO v\_payment\_id\_membership;

-- Step 4: Insert into payment table with payment\_id

INSERT INTO payment (payment\_id, payment\_type, payment\_date, amount, member\_id)

VALUES (payment\_seq.NEXTVAL, 'membership', SYSDATE, v\_membership\_fee, v\_member\_id)

RETURNING payment\_id INTO v\_payment\_id\_payment;

-- Step 5: Update payment\_id in membership table

UPDATE membership

SET payment\_id = v\_payment\_id\_payment

WHERE member\_id = v\_member\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Member added successfully!');

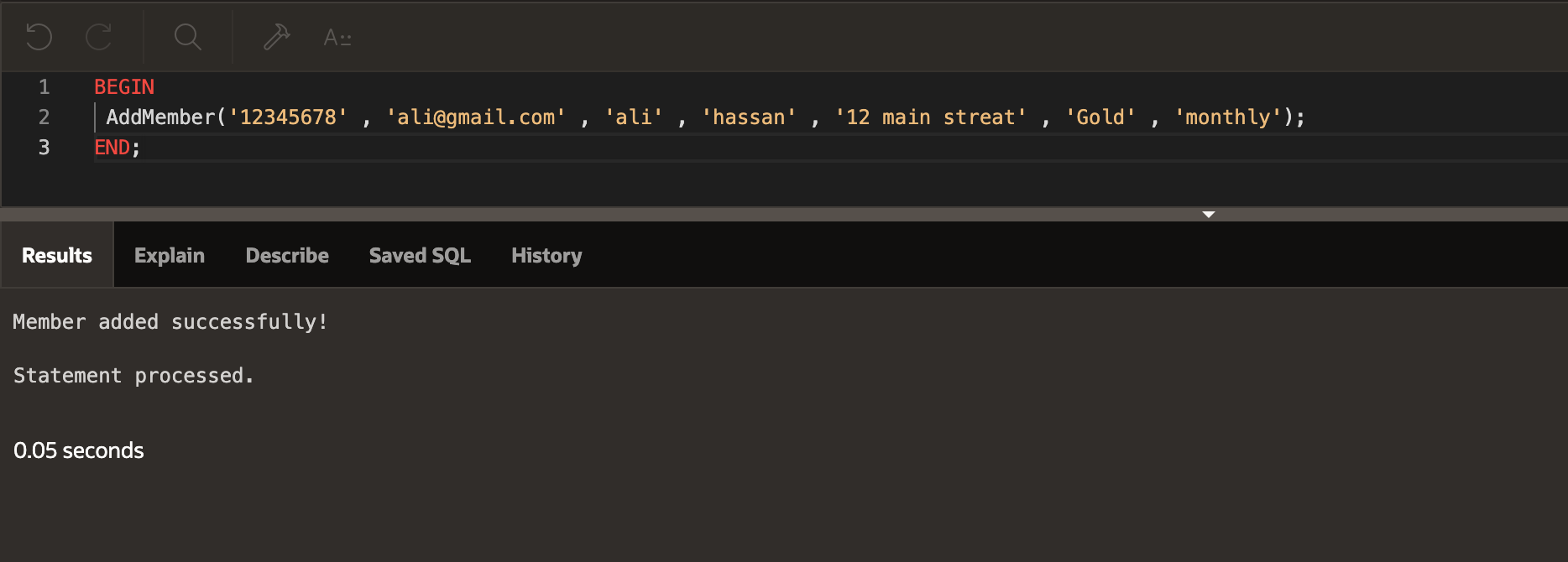
EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END AddMember;

****

CREATE OR REPLACE PROCEDURE ClassStatus (

in\_instructor\_id INT

) IS

v\_instructor\_name VARCHAR2(100);

v\_total\_members INT := 0;

v\_total\_profits DECIMAL(8, 2) := 0;

BEGIN

-- Get the instructor's name

SELECT first\_name || ' ' || last\_name INTO v\_instructor\_name

FROM staff

WHERE staff\_id = in\_instructor\_id;

-- Display the instructor's name

DBMS\_OUTPUT.PUT\_LINE('Instructor: ' || v\_instructor\_name);

-- Iterate through classes

FOR class\_info IN (

SELECT c.class\_id, c.class\_name, c.maximum\_capacity, c.class\_time, c.duration\_minutes, c.cost,

COUNT(cr.member\_id) AS enrolled\_members,

(COUNT(cr.member\_id) \* c.cost) AS total\_profit

FROM class c

LEFT JOIN class\_registration cr ON c.class\_id = cr.class\_id

WHERE c.instructor\_id = in\_instructor\_id

AND c.class\_time > SYSDATE

GROUP BY c.class\_id, c.class\_name, c.maximum\_capacity, c.class\_time, c.duration\_minutes, c.cost

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Class ID: ' || class\_info.class\_id);

DBMS\_OUTPUT.PUT\_LINE('Class Name: ' || class\_info.class\_name);

DBMS\_OUTPUT.PUT\_LINE('Maximum Capacity: ' || class\_info.maximum\_capacity);

DBMS\_OUTPUT.PUT\_LINE('Class Time: ' || TO\_CHAR(class\_info.class\_time, 'YYYY-MM-DD HH24:MI:SS'));

DBMS\_OUTPUT.PUT\_LINE('Duration Minutes: ' || class\_info.duration\_minutes);

DBMS\_OUTPUT.PUT\_LINE('Cost: ' || class\_info.cost);

DBMS\_OUTPUT.PUT\_LINE('Enrolled Members: ' || class\_info.enrolled\_members);

DBMS\_OUTPUT.PUT\_LINE('Total Profit: ' || class\_info.total\_profit);

DBMS\_OUTPUT.PUT\_LINE('------------------------');

-- Update totals

v\_total\_members := v\_total\_members + class\_info.enrolled\_members;

v\_total\_profits := v\_total\_profits + class\_info.total\_profit;

END LOOP;

-- Display totals

DBMS\_OUTPUT.PUT\_LINE('Total Enrolled Members: ' || v\_total\_members);

DBMS\_OUTPUT.PUT\_LINE('Total Expected Profits: ' || v\_total\_profits);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

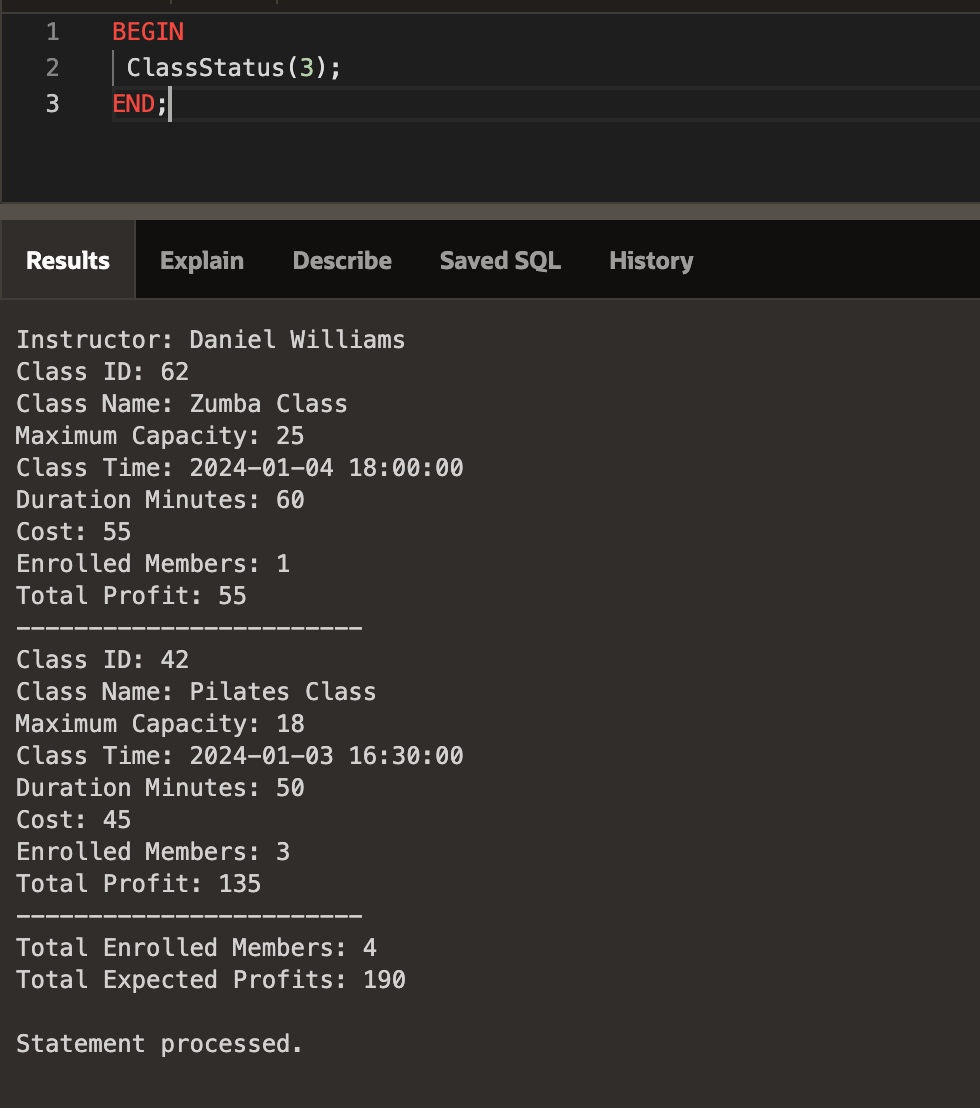
DBMS\_OUTPUT.PUT\_LINE('Instructor not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

END ClassStatus;

**Screen captures of different Forms and Reports with explanation.**

****

CREATE OR REPLACE PROCEDURE ClassRegistration(p\_class\_id INT, p\_member\_id INT) AS

v\_enrollment\_count INT;

v\_max\_capacity INT;

v\_payment\_id INT;

v\_membership\_start\_date DATE;

v\_membership\_end\_date DATE;

v\_membership\_type\_period VARCHAR2(20);

v\_class\_date DATE;

v\_class\_cost NUMBER;

v\_existing\_registration INT;

BEGIN

-- Check if the provided member\_id is valid

IF NOT Check\_member\_id(p\_member\_id) THEN

DBMS\_OUTPUT.PUT\_LINE('Invalid member ID. Registration failed.');

RETURN;

END IF;

-- Check if the provided class\_id is valid

IF NOT Check\_class\_id(p\_class\_id) THEN

DBMS\_OUTPUT.PUT\_LINE('Invalid class ID. Registration failed.');

RETURN;

END IF;

-- Check if the member is already registered for the class

SELECT COUNT(\*) INTO v\_existing\_registration

FROM class\_registration

WHERE class\_id = p\_class\_id AND member\_id = p\_member\_id;

IF v\_existing\_registration > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Member is already registered for the class. Registration not allowed.');

RETURN;

END IF;

BEGIN

-- Retrieve the current enrollment count for the class

SELECT COUNT(\*) INTO v\_enrollment\_count FROM class\_registration WHERE class\_id = p\_class\_id;

-- Retrieve the maximum capacity for the class

SELECT maximum\_capacity INTO v\_max\_capacity FROM class WHERE class\_id = p\_class\_id;

-- Retrieve the membership start date and type period for the member

SELECT start\_date, period INTO v\_membership\_start\_date, v\_membership\_type\_period

FROM membership\_type mt

JOIN membership m ON mt.membership\_type\_id = m.membership\_type\_id

WHERE m.member\_id = p\_member\_id AND m.status = ' Active';

-- Retrieve the class date and cost

SELECT class\_time, cost INTO v\_class\_date, v\_class\_cost FROM class WHERE class\_id = p\_class\_id;

-- Calculate the membership end date based on the type period

IF v\_membership\_type\_period = 'monthly' THEN

v\_membership\_end\_date := ADD\_MONTHS(v\_membership\_start\_date, 1);

ELSIF v\_membership\_type\_period = 'yearly' THEN

v\_membership\_end\_date := ADD\_MONTHS(v\_membership\_start\_date, 12);

END IF;

-- Check if there is still capacity for enrollment

IF v\_enrollment\_count < v\_max\_capacity THEN

-- Check if the class date is before the membership end date

IF v\_class\_date < v\_membership\_end\_date THEN

-- Generate a new payment\_id

v\_payment\_id := PAYMENT\_SEQ.NEXTVAL;

-- Insert the new registration into the payment table

INSERT INTO payment(payment\_id, payment\_type, payment\_date, amount, member\_id)

VALUES (v\_payment\_id, 'class', SYSDATE, v\_class\_cost, p\_member\_id);

-- Insert the new registration into the class\_registration table

INSERT INTO class\_registration(class\_registration\_id, member\_id, class\_id, registration\_date, payment\_id)

VALUES (class\_registration\_seq.NEXTVAL, p\_member\_id, p\_class\_id, SYSDATE, v\_payment\_id);

-- Display success message

DBMS\_OUTPUT.PUT\_LINE('Member successfully registered for the class.');

ELSE

-- Display error message if the class date is after the membership end date

DBMS\_OUTPUT.PUT\_LINE('Class date is after the member''s membership end date. Registration not allowed.');

END IF;

ELSE

-- Display error message if the class is already at full capacity

DBMS\_OUTPUT.PUT\_LINE('Class is already at full capacity. Registration not allowed.');

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No data found during registration.');

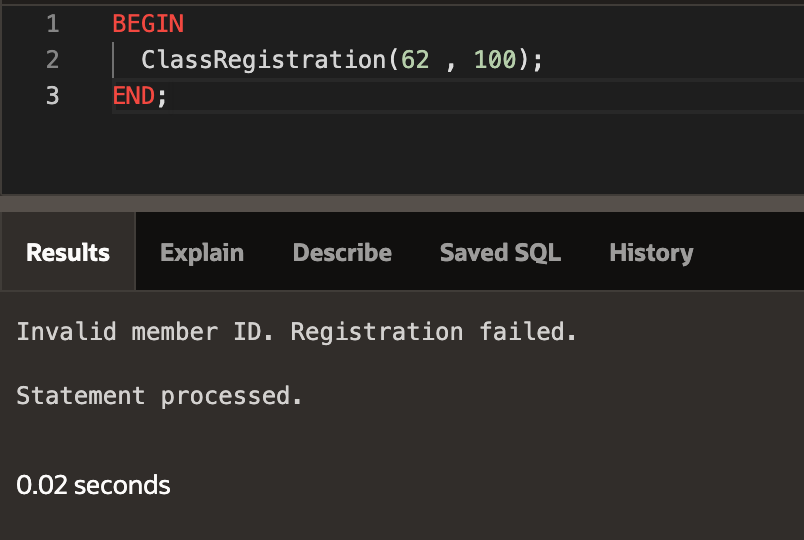
WHEN OTHERS THEN

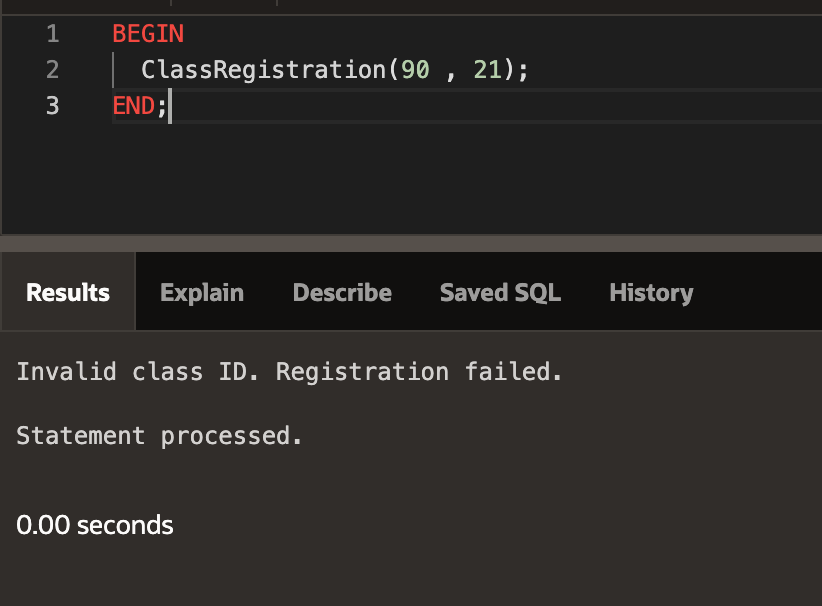
-- Display detailed error message if any error occurs during the registration process

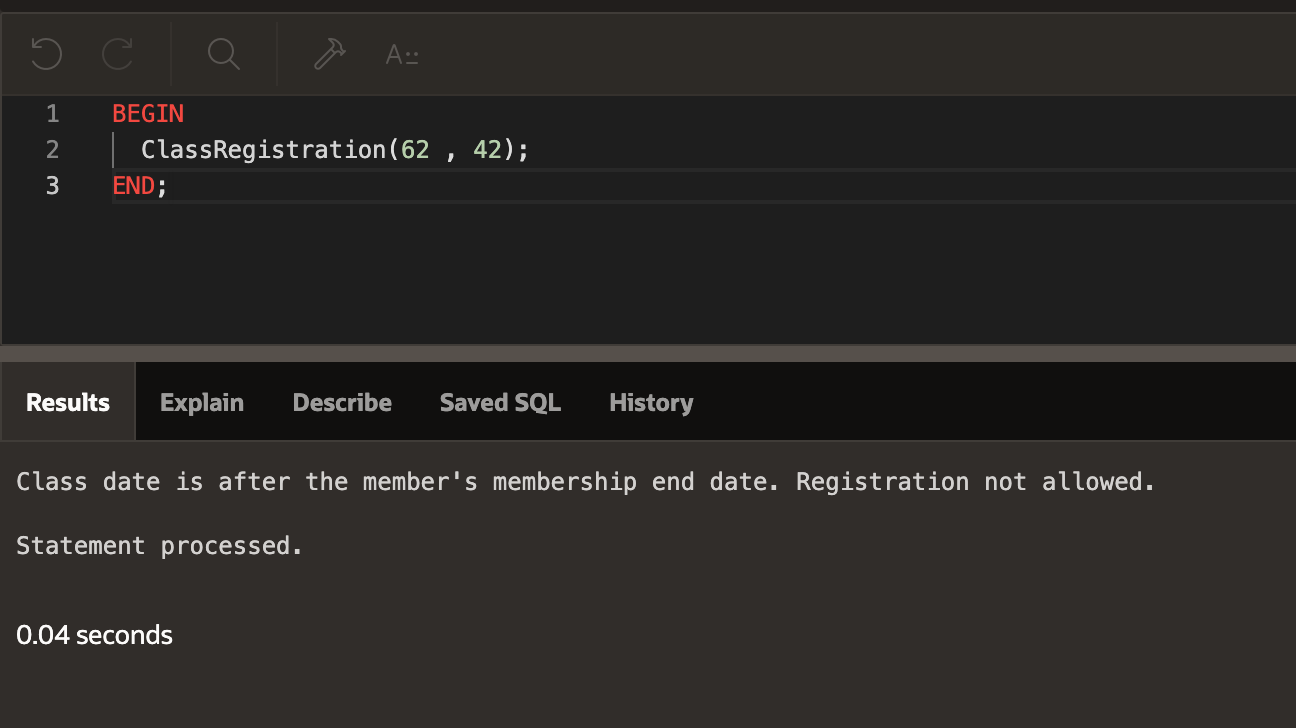
DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

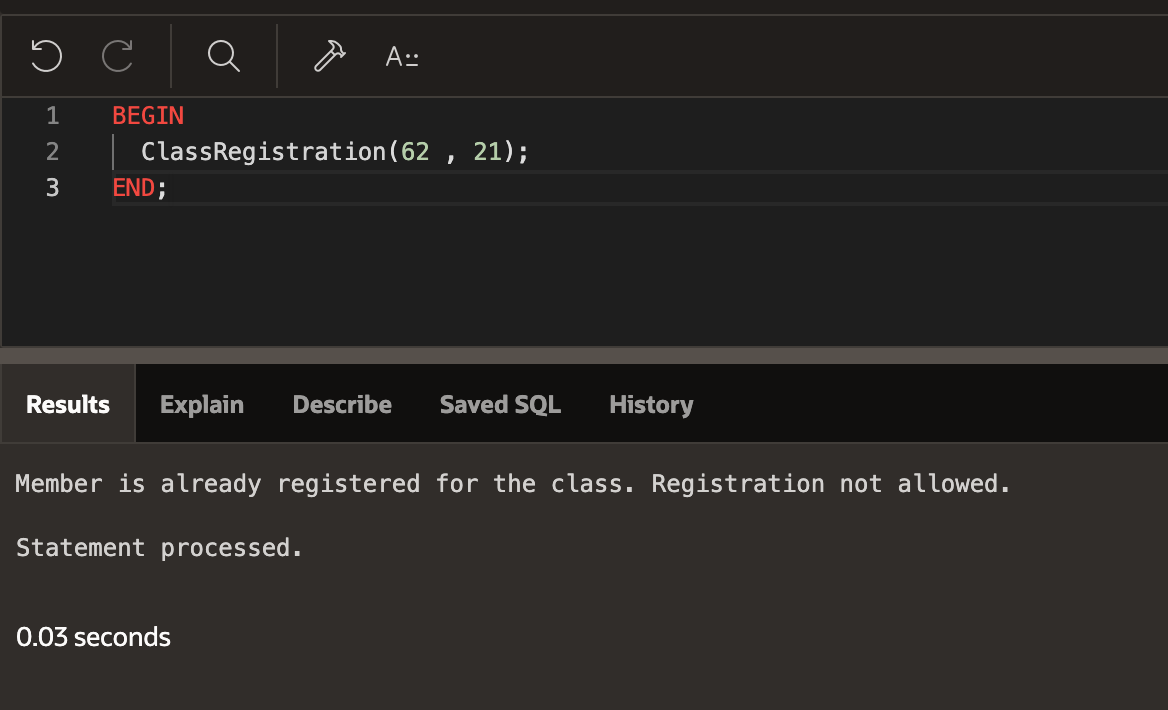
END;

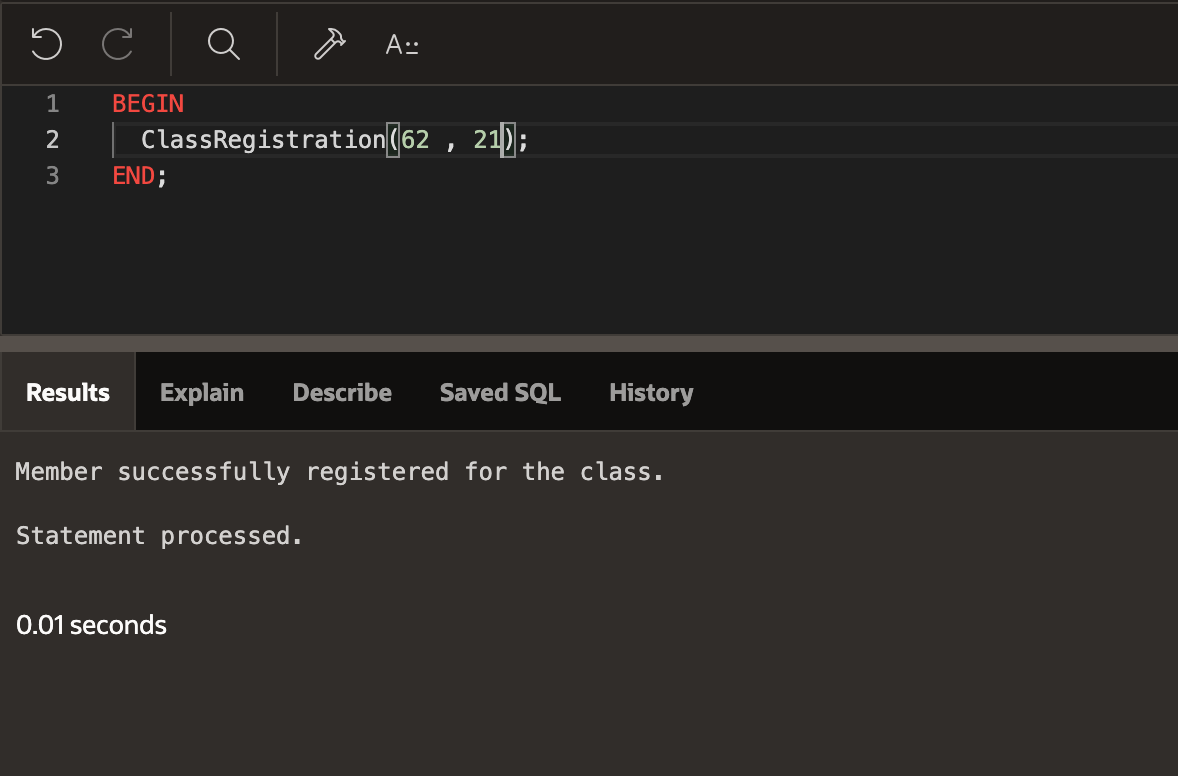
END;











CREATE OR REPLACE PROCEDURE UpdateMembershipStatus AS

BEGIN

-- Update monthly memberships

UPDATE membership

SET status = ' Expired'

WHERE status = ' Active' AND membership\_type\_id IN (

SELECT membership\_type\_id

FROM membership\_type

WHERE period = 'monthly'

) AND SYSDATE > ADD\_MONTHS(start\_date, 1);

-- Update yearly memberships

UPDATE membership

SET status = ' Expired'

WHERE status = ' Active' AND membership\_type\_id IN (

SELECT membership\_type\_id

FROM membership\_type

WHERE period = 'yearly'

) AND SYSDATE > ADD\_MONTHS(start\_date, 12);

COMMIT;

END UpdateMembershipStatus;

-- Create a job to run the UpdateMembershipStatus procedure every day at a specific time

BEGIN

DBMS\_SCHEDULER.create\_job (

job\_name => 'UPDATE\_MEMBERSHIP\_JOB',

job\_type => 'PLSQL\_BLOCK',

job\_action => 'BEGIN UpdateMembershipStatus; END;',

start\_date => SYSTIMESTAMP,

repeat\_interval => 'FREQ=DAILY; BYHOUR=0; BYMINUTE=0; BYSECOND=0',

enabled => TRUE

);

END;

/