Gym management system

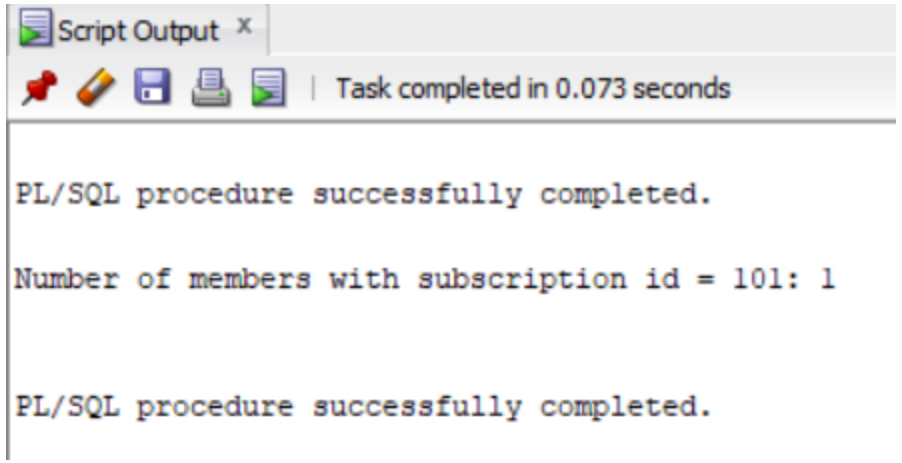
Name: Rosu Liviu-Mihai
Group:1076

Description of the project:

A Gym Database System created to manage member information, handle payments, track workout programs and overall organize a gym's information

A. Interaction with the Oracle server through SQL commands (DDL and DML) in PL/SQL blocks: using execute immediate, particularities regarding the use of the select command, as well as functions at the row and group level.

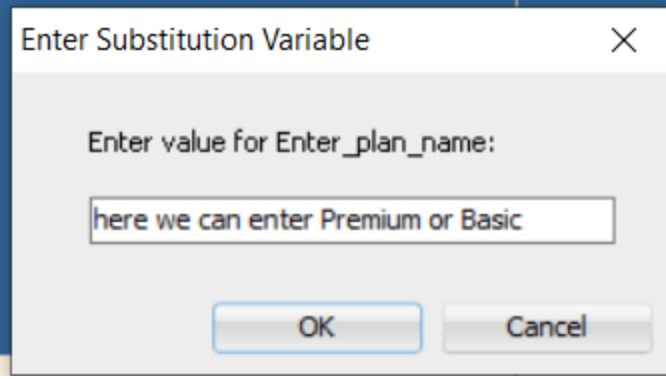
```
DECLARE
v_count NUMBER;
BEGIN
-- count how many members are
subscribed to planID 101
SELECT COUNT(*) INTO v_count
FROM members_of_gym
WHERE SubscriptionID = 101;
dbms_output.put_line('Number of
members with subscription id =
101: ' || v_count);
END;
/
```



B. Alternative and repetitive structures (IF, CASE, FOR, LOOP, WHILE).

```
DECLARE
v_plan VARCHAR(20) := '&Enter_plan_name';
BEGIN
CASE v_plan
WHEN 'Basic' THEN
dbms_output.put_line('Cost is 50$');
WHEN 'Premium' THEN
dbms_output.put_line('Cost is 100$');
ELSE
dbms_output.put_line('Unknown subscription');
END CASE;
END;
/
```

```
DECLARE
v_plan VARCHAR(20) := '&Enter_plan_name';
BEGIN
CASE v_plan
WHEN 'Basic' THEN
dbms_output.put_line('Cost is 50$');
WHEN 'Premium' THEN
dbms_output.put_line('Cost is 100$');
ELSE
dbms_output.put_line('Unknown subscription');
END CASE;
END;
/
```




```
Cost is 100$


PL/SQL procedure successfully completed.
```

Check if a client has premium or basic membership (this case client with id 12)

```
DECLARE v_member_id
MEMBERS_OF_GYM.MemberID%TYPE := 12;
v_plan_name SUBSCRIPTION.PlanName%TYPE; BEGIN
SELECT s.PlanName INTO v_plan_name FROM
MEMBERS_OF_GYM m JOIN SUBSCRIPTION s ON
m.SubscriptionID = s.SubscriptionID WHERE
m.MemberID = v_member_id; IF v_plan_name =
'Premium Plan' THEN
DBMS_OUTPUT.PUT_LINE('Member ' || v_member_id || '
has a Premium subscription.'); ELSE
DBMS_OUTPUT.PUT_LINE('Member ' || v_member_id || '
has a Basic subscription.'); END IF; END;
```

```
PL/SQL procedure successfully completed.

Member 12 has a Premium subscription.


PL/SQL procedure successfully completed.
```

display a message based on equipment availability

```
DECLARE v_equipment_name
EQUIPMENT.EquipmentName%TYPE := 'Treadmill'; BEGIN
CASE v_equipment_name WHEN 'Treadmill' THEN
DBMS_OUTPUT.PUT_LINE('Cardio equipment available.');
WHEN 'Dumbbells' THEN
DBMS_OUTPUT.PUT_LINE('Strength training equipment
available.'); ELSE DBMS_OUTPUT.PUT_LINE('Equipment type
unknown or not available.'); END CASE; END; /
```

```
Cardio equipment available.

PL/SQL procedure successfully completed.
```

Display all members and their
subscription type

```
BEGIN FOR rec IN ( SELECT m.FullName, s.PlanName FROM
MEMBERS_OF_GYM m JOIN SUBSCRIPTION s ON
m.SubscriptionID = s.SubscriptionID ) LOOP
DBMS_OUTPUT.PUT_LINE('Member: ' || rec.FullName || ' - Plan:
' || rec.PlanName); END LOOP; END;
```

```
PL/SQL procedure successfully completed.

Member: John Doe - Plan: Basic Plan
Member: Mike Johnson - Plan: Premium Plan
Member: Alice Matthews - Plan: Basic Plan
Member: Andrew Joe - Plan: Premium Plan
Member: John Smith - Plan: Basic Plan
Member: Daniel Miller - Plan: Premium Plan
Member: Emma Martinez - Plan: Basic Plan
Member: Sophia Brown - Plan: Premium Plan
Member: Michael Williams - Plan: Basic Plan
Member: Olivia Garcia - Plan: Premium Plan
Member: Rosu Liviu-Mihai - Plan: Premium Plan
```

C. Data collections (index by table, nested table, varray).

```
--index-by table
DECLARE TYPE member_name_table IS TABLE OF
VARCHAR2(100) INDEX BY PLS_INTEGER; v_names
member_name_table; BEGIN v_names(1) := 'Liviu'; v_names(2) :=
'Mihnea'; dbms_output.put_line('Member 1: ' || v_names(1));
dbms_output.put_line('Member 2: ' || v_names(2)); END; /
```

```
Member 1: Liviu
Member 2: Mihnea
```

```
--varray
DECLARE
TYPE member_name_var_array IS VARRAY(5) OF VARCHAR2(100);
v_names member_name_var_array := member_name_var_array('John',
'Alice');
BEGIN
v_names.EXTEND;
v_names(3) := 'Bob';
FOR i IN 1 .. v_names.COUNT LOOP
DBMS_OUTPUT.PUT_LINE('Member: ' || v_names(i));
END LOOP;
END;
/
```

```
Member: John
Member: Alice
Member: Bob
```

```
--nested table (like a list but not fixed size, can grow as
needed)
DECLARE TYPE Member_id_table IS TABLE OF
NUMBER; v_member_ids Member_id_table :=
Member_id_table(); BEGIN v_member_ids.EXTEND(3);
v_member_ids(1) := 101; v_member_ids(2) := 102;
v_member_ids(3) := 103; FOR i IN 1 ..
v_member_ids.COUNT LOOP
DBMS_OUTPUT.PUT_LINE('Member ID: ' ||
v_member_ids(i)); END LOOP; END; /
```

```
Member ID: 101
Member ID: 102
Member ID: 103


PL/SQL procedure successfully completed.
```

D. Exception handling (minimum 3 implicit, 2 explicit).

Implicit

```
DECLARE v_name EQUIPMENT.EquipmentName%TYPE;
BEGIN SELECT EquipmentName INTO v_name FROM
EQUIPMENT WHERE EquipmentID = 9999; -- testing for
an id that does not exist
dbms_output.put_line('Equipment: ' || v_name);
EXCEPTION WHEN NO_DATA_FOUND THEN
dbms_output.put_line('No equipment found with that
specified ID!');
END; /
```

```
No equipment found with that specified ID!


PL/SQL procedure successfully completed.
```

```
DECLARE v_name EQUIPMENT.EquipmentName%TYPE; BEGIN
SELECT EquipmentName into v_name FROM EQUIPMENT; -- here
the program will return multiple rows because of no WHERE clause
dbms_output.put_line('Equipment: ' || v_name); EXCEPTION WHEN
TOO_MANY_ROWS THEN dbms_output.put_line('Too many rows
returned!');
END; /
```

```
Too many rows returned!


PL/SQL procedure successfully completed.
```

Explicit

Check for a trainer id, if invalid return an exception

```
DECLARE v_trainer_id TRAINERS.TrainerID%TYPE := 100; -- invalid trainer ID
v_name TRAINERS.FullName%TYPE; e_trainer_not_found EXCEPTION; BEGIN
SELECT FullName INTO v_name FROM TRAINERS WHERE TrainerID =
v_trainer_id; DBMS_OUTPUT.PUT_LINE('Trainer: ' || v_name); EXCEPTION WHEN
NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('Trainer with ID ' ||
v_trainer_id || ' not found.'); END; /
```

```
Trainer with ID 100 not found.


PL/SQL procedure successfully completed.
```

DECLARE v_subscription_duration SUBSCRIPTION.durationmonths%type := 18; -- number of months that exceeds the max (12) BEGIN IF v_subscription_duration > 12 THEN RAISE_APPLICATION_ERROR(-20001, 'Subscription period cannot exceed one month'); END IF; dbms_output.put_line('Valid subscription'); END; /

```
DECLARE
*
ERROR at line 1:
ORA-20001: Subscription period cannot exceed one month
ORA-06512: at line 5

https://docs.oracle.com/error-help/db/ora-20001/
```

With valid input, eg: 12

```
Valid subscription


PL/SQL procedure successfully completed.
```

E. Cursor management: implicit and explicit (with and without parameters, FOR UPDATE).

Implicit cursor

```
BEGIN DELETE FROM
MEMBERS_OF_GYM
WHERE SubscriptionID IS
NULL;
DBMS_OUTPUT.PUT_LINE(
SQL%ROWCOUNT || '
member(s) deleted.'); END; /
```

```
0 member(s) deleted.


PL/SQL procedure successfully completed.
```

Explicit cursor without parameters

DECLARE CURSOR member_cursor IS SELECT fullname, email FROM members_of_gym; v_name members_of_gym.fullname%type; v_email members_of_gym.email%type; BEGIN OPEN member_cursor; LOOP FETCH member_cursor INTO v_name, v_email; EXIT WHEN member_cursor%NOTFOUND; dbms_output.put_line('Member: ' || v_name || '; ' || 'Member email: ' || v_email); END LOOP; END; /

```
Member: John Doe; Member email: john.doe@gmail.com
Member: Mike Johnson; Member email: mike.johnson@gmail.com
Member: Alice Matthews; Member email: alicematthwes@gmail.com
Member: Andrew Joe; Member email: andrewjoe@gmail.com
Member: John Smith; Member email: johnsmith@gmail.com
Member: Daniel Miller; Member email: danielmiller@gmail.com
Member: Emma Martinez; Member email: emma.martinez@gmail.com
Member: Sophia Brown; Member email: sophiabrown@gmail.com
Member: Michael Williams; Member email: michaelwilliams@gmail.com
Member: Olivia Garcia; Member email: oliviagarcia@gmail.com
```

Explicit cursor with paramters

```
DECLARE CURSOR c_subscription(p_cost NUMBER) IS
SELECT FullName FROM MEMBERS_OF_GYM m JOIN
SUBSCRIPTION s ON m.SubscriptionID =
s.SubscriptionID WHERE s.Cost > p_cost; v_name
MEMBERS_OF_GYM.FullName%TYPE; BEGIN OPEN
c_subscription(50); -- Only members with plans costing
more than 50 LOOP FETCH c_subscription INTO v_name;
EXIT WHEN c_subscription%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('Member: ' || v_name); END
LOOP; CLOSE c_subscription; END; /
```

```
PL/SQL procedure successfully completed.

Member: Mike Johnson
Member: Andrew Joe
Member: Daniel Miller
Member: Sophia Brown
Member: Olivia Garcia
Member: Rosu Liviu-Mihai
```

Cursor UPDATE

```
DECLARE CURSOR c IS SELECT MemberID, PhoneNumber FROM MEMBERS_OF_GYM WHERE SubscriptionID = 101 FOR UPDATE;
v_id MEMBERS_OF_GYM.MemberID%TYPE;
v_phone MEMBERS_OF_GYM.PhoneNumber%TYPE;
BEGIN OPEN c; LOOP FETCH c INTO v_id, v_phone; EXIT WHEN c%NOTFOUND;
UPDATE MEMBERS_OF_GYM SET PhoneNumber = '0000000000' WHERE CURRENT OF c;
DBMS_OUTPUT.PUT_LINE('Updated member ID: ' || v_id); END LOOP; CLOSE c; END; /
```

```
PL/SQL procedure successfully completed.

Updated member ID: 1
```

We can see member 'John doe' with **member id: 1** now has the phone number updated and filled with 0's

	MEMBERID	FULLNAME	EMAIL	PHONENUMBER	DATEOFBIRTH	SUBSCRIPTIONID	JOIN_DATE
1	1	John Doe	john.doe@gmail.com	0000000000	15-MAY-90	101	24-MAY-25
2	2	Mike Johnson	mike.johnson@gmail.com	0987654321	20-JUL-95	102	24-MAY-25
3	3	Alice Matthews	alicematthwes@gmail.com	1234098765	25-OCT-98	103	24-MAY-25
4	4	Andrew Joe	andrewjoe@gmail.com	5432167890	04-NOV-99	104	24-MAY-25
5	5	John Smith	johnsmith@gmail.com	0238495607	21-JUL-93	105	24-MAY-25
6	6	Daniel Miller	danielmiller@gmail.com	0659231837	20-FEB-03	106	24-MAY-25
7	7	Emma Martinez	emma.martinez@gmail.com	0436586978	25-OCT-98	107	24-MAY-25
8	8	Sophia Brown	sophiabrown@gmail.com	0475697821	04-NOV-99	108	24-MAY-25
9	9	Michael Williams	michaelwilliams@gmail.com	0348695869	04-JUN-02	109	24-MAY-25
10	10	Olivia Garcia	oliviagarcia@gmail.com	0492182738	04-APR-01	110	24-MAY-25
11	12	Rosu Liviu-Mihai	rosuliviu23@stud.ase.ro	0767691019	28-MAY-04	112	24-MAY-25

F. Functions, procedures, inclusion in packages (minimum 3 functions, 3 procedures, and a package that includes other functions and procedures).

PROCEDURES:

```
CREATE OR REPLACE PROCEDURE register_member( p_member_id IN
MEMBERS_OF_GYM.memberid%type, p_full_name IN MEMBERS_OF_GYM.fullname%type,
p_email IN MEMBERS_OF_GYM.email%type, p_phone_number IN
MEMBERS_OF_GYM.phonenumber%type, p_date_of_birth IN
MEMBERS_OF_GYM.dateofbirth%type, p_subscription_id IN
MEMBERS_OF_GYM.subscriptionid%type, p_join_date IN
MEMBERS_OF_GYM.join_date%type, p_address IN MEMBERS_OF_GYM.address%type ) IS
v_sub_exists NUMBER; v_id_exists NUMBER; BEGIN SELECT COUNT(*) INTO v_sub_exists
FROM SUBSCRIPTION WHERE SubscriptionID = p_subscription_id; IF v_sub_exists = 0
THEN dbms_output.put_line('Invalid subscription ID: ' || p_subscription_id); RETURN; END IF;
SELECT COUNT(*) INTO v_id_exists FROM MEMBERS_OF_GYM WHERE memberid =
p_member_id; IF v_id_exists > 0 THEN dbms_output.put_line('Member with id: ' ||
p_member_id || ' already exists'); RETURN; END IF; INSERT INTO MEMBERS_OF_GYM(
memberid, fullname, email, phonenumber, dateofbirth, subscriptionid, join_date, address )
VALUES( p_member_id, p_full_name, p_email, p_phone_number, p_date_of_birth,
p_subscription_id, p_join_date, p_address ); dbms_output.put_line('Member: ' || p_full_name
|| ' registered successfully!'); COMMIT; EXCEPTION WHEN DUP_VAL_ON_INDEX THEN
dbms_output.put_line('Membere with same ID or email already exists!'); WHEN OTHERS
THEN dbms_output.put_line('Unexpected error. ' || SQLERRM); END; /
```

Output for the previous procedure

```
PL/SQL procedure successfully completed.

Member: Test Again registered successfully!
```

	MEMBERID	FULLNAME	EMAIL	PHONENUMBER	DATEOFBIRTH	SUBSCRIPTIONID	JOIN_DATE	ADDRESS
5	5	John Smith	johnsmith@gmail.com	0250495007	21-OCT-95	105	24-MAY-25	(null)
6	6	Daniel Miller	danielmiller@gmail.com	0659231837	20-FEB-03	106	24-MAY-25	(null)
7	7	Emma Martinez	emma.martinez@gmail.com	0436586978	25-OCT-98	107	24-MAY-25	(null)
8	8	Sophia Brown	sophiabrown@gmail.com	0475697821	04-NOV-99	108	24-MAY-25	(null)
9	9	Michael Williams	michaelwilliams@gmail.com	0348695869	04-JUN-02	109	24-MAY-25	(null)
10	10	Olivia Garcia	oliviagarcia@gmail.com	0492182738	04-APR-01	110	24-MAY-25	(null)
11	12	Rosu Liviu-Mihai	rosuliviu23@stud.ase.ro	0767691019	28-MAY-04	112	24-MAY-25	(null)
12	14	Test Member	testmember@gmail.com	0771234567	14-MAY-90	110	26-MAY-25	test street name.
13	15	Test Again	anotheremail@gmail.com	0770000000	01-JAN-95	101	26-MAY-25	Another street

```
--show member subscription by member id
CREATE OR REPLACE PROCEDURE show_member_subscription(p_member_id
MEMBERS_OF_GYM.MemberID%TYPE)
IS
v_name MEMBERS_OF_GYM.FULLNAME%TYPE;
v_plan SUBSCRIPTION.PLANNAME%TYPE;
v_cost SUBSCRIPTION.COST%TYPE;

BEGIN
SELECT m.FullName, s.PlanName, s.Cost
INTO v_name, v_plan, v_cost
FROM MEMBERS_OF_GYM m
JOIN SUBSCRIPTION s on m.SubscriptionID = s.SubscriptionID
WHERE m.MemberID = p_member_id;

dbms_output.put_line('Member: ' || v_name);
dbms_output.put_line('Subscription Plan: ' || v_plan);
dbms_output.put_line('Cost: $' || v_cost);

EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line('No member found with ID: ' || p_member_id);
END;
/

-- testing show_member_sbuscription
BEGIN
show_member_subscription(1); --id 1 is valid and 101 is not valid
END;
/
```

```
Member: John Doe
Subscription Plan: Basic Plan
Cost: $50


PL/SQL procedure successfully completed.
```

```
No member found with ID: 101


PL/SQL procedure successfully completed.
```

```
CREATE OR REPLACE PROCEDURE
count_members_by_plan(p_plan_name IN
SUBSCRIPTION.PlanName%TYPE) IS v_count NUMBER;
BEGIN SELECT COUNT(*) INTO v_count FROM
MEMBERS_OF_GYM m JOIN SUBSCRIPTION s on
m.SubscriptionID = s.SubscriptionID WHERE
s.PlanName = p_plan_name;
dbms_output.put_line('Number of members subscribed
to ' || p_plan_name || ': ' || v_count); END; /
```

```
SELECT DISTINCT PlanName FROM SUBSCRIPTION;
```

```
Number of members subscribed to Basic Plan: 6

PL/SQL procedure successfully completed.

Number of members subscribed to Premium Plan: 7

PL/SQL procedure successfully completed.
```

	PLANNAME
1	Basic Plan
2	Premium Plan

FUNCTIONS:

```
CREATE OR REPLACE FUNCTION
get_member_age(p_member_id IN NUMBER)
RETURN NUMBER IS v_age NUMBER; BEGIN
SELECT FLOOR(MONTHS_BETWEEN(SYSDATE,
dateofbirth) / 12) INTO v_age FROM
MEMBERS_OF_GYM WHERE memberid =
p_member_id; RETURN v_age;
END; /

SELECT get_member_age(1) AS age FROM DUAL;
```

	AGE
1	35

```
--function to calculate BMI
CREATE OR REPLACE FUNCTION
calculate_bmi(p_weight IN NUMBER, p_height
IN NUMBER) RETURN NUMBER IS BEGIN
RETURN ROUND(p_weight / (p_height *
p_height), 2); END; / --testing

SELECT calculate_bmi(80, 1.88) as bmi FROM
DUAL;
```

	BMI
1	22.63

G.Triggers

Statement triggers

CREATE OR REPLACE TRIGGER trigger_after_insert_members AFTER INSERT ON MEMBERS_OF_GYM BEGIN dbms_output.put_line('New members inserted into MEMBERS_OF_GYM table.'); END; /

CREATE OR REPLACE TRIGGER trigger_before_update_train_session BEFORE UPDATE ON TRAINING_SESSION DECLARE v_old_count NUMBER; BEGIN dbms_output.put_line('Updating TRAINING_SESSION records.'); END; /

```
1 row inserted.


Commit complete.

Updating TRAINING_SESSION records.


1 row updated.
```

```
CREATE OR REPLACE TRIGGER trigger_before_update_train_session
BEFORE UPDATE ON TRAINING_SESSION
DECLARE
v_old_count NUMBER;
BEGIN
dbms_output.put_line('Updating TRAINING_SESSION records.');
END;
/

INSERT INTO TRAINING_SESSION (TrainingSessionID, TrainingSessionName, TrainerID, Schedule)
VALUES (4, 'Morning yoga', 4, TO_TIMESTAMP('2025-06-12 10:30:00', 'YYYY-MM-DD HH24:MI:SS'));

COMMIT;

UPDATE TRAINING_SESSION
SET TRAININGSESSIONNAME = 'Evening yoga'
WHERE TRAININGSESSIONID = 4;
```

Row level triggers

```
CREATE OR REPLACE TRIGGER trigger_after_update_members
AFTER UPDATE OF SUBSCRIPTIONID ON MEMBERS_OF_GYM
FOR EACH ROW
BEGIN
dbms_output.put_line('Member ' || :OLD.FULLNAME || '
subscription changed from ' || :OLD.SUBSCRIPTIONID || ' to ' ||
:NEW.SUBSCRIPTIONID);
END;
/

UPDATE MEMBERS_OF_GYM
SET SUBSCRIPTIONID = 105
WHERE MEMBERID = 1;
```

```
Trigger TRIGGER_AFTER_UPDATE_MEMBERS compiled

Member John Doe subscription changed from 101 to 105


1 row updated.
```

```
CREATE OR REPLACE TRIGGER
trigger_after_update_email AFTER UPDATE OF EMAIL
ON MEMBERS_OF_GYM FOR EACH ROW BEGIN
DBMS_OUTPUT.PUT_LINE('Email for member ' ||
:OLD.FullName || ' changed from ' || :OLD.Email || ' to '
|| :NEW.Email);
END; /

UPDATE MEMBERS_OF_GYM
SET Email = 'new_email_from_trigger@example.com'
WHERE MemberID = 1;
```

```
Trigger TRIGGER_AFTER_UPDATE_EMAIL compiled

Email for member John Doe changed from john.doe@gmail.com to new_email_from_trigger@example.com


1 row updated.
```