# Proiect SGBD – Nedelcu Radu Gr. 232

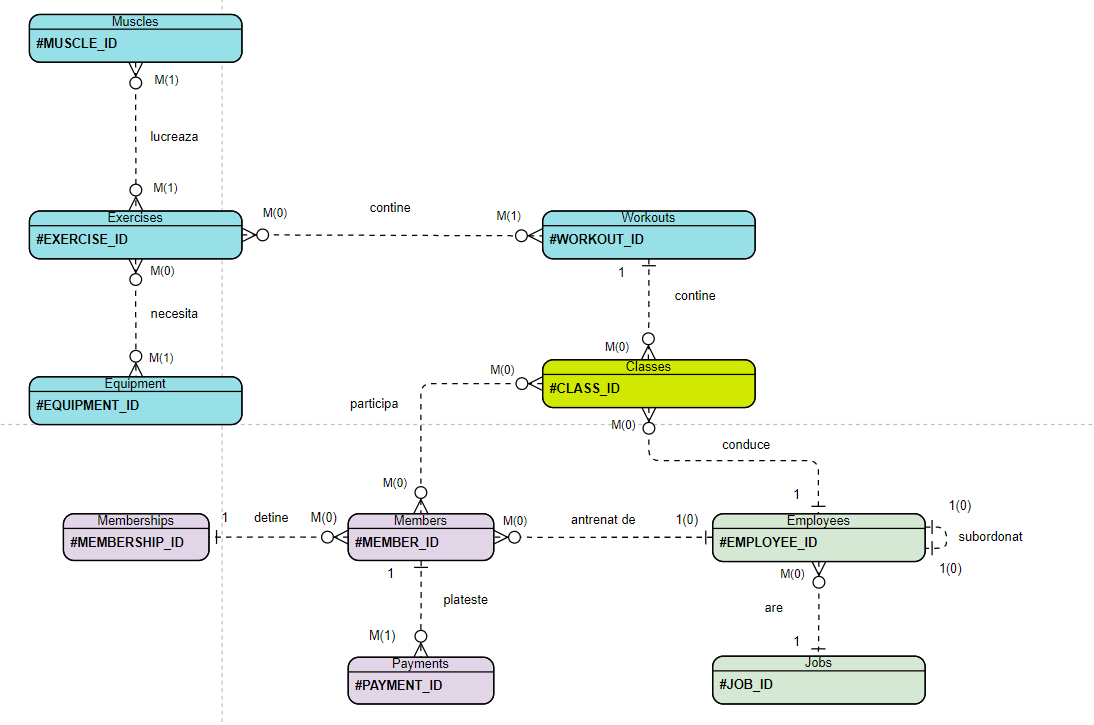
CUPRINS

1. Scurtă prezentare a bazei de date
2. Diagrama Entitate-Relație
3. Diagrama Conceptuală
4. Implementarea bazei de date
5. Popularea tabelelor cu informații coerente
6. Afișarea datelor angajaților folosind un tablou îmbricat ce conține obiecte
7. Modificarea salariului angajaților folosind un cursor
8. Crearea unei funcții ce întoarce numărul de echipamente necesare unui antrenament
9. Crearea unei proceduri ce afișează mușchii lucrați de fiecare antrenament
10. Compound trigger ce limitează numărul maxim de participanți într-o clasă
11. Trigger la nivel de linie ce verifică respectarea limitelor salariale impuse de job
12. Trigger de tip LDD folosit pentru logging.
13. Definirea pachetului ce conține toate obiectele declarate anterior
14. Crearea unui pachet care facilitează managerierea membrilor

## Scurtă prezentare a bazei de date

Baza de date modelată este potrivită pentru o sală de sport ce oferă membrilor acces la antrenori privați și antrenamente în grup (Cum ar fi un centru World Class). Aceasta este împărțită în 3 părți ce se concentrează pe membri, angajați și exerciții.

## Diagrama Entitate-Relație



## Diagrama Conceptuală

Text

Description automatically generated with medium confidence

## Implementarea Bazei De Date

### Codul SQL

CREATE TABLE jobs(

job\_id VARCHAR(20) PRIMARY KEY,

job\_title VARCHAR(30) NOT NULL,

min\_salary INT NOT NULL,

max\_salary INT NOT NULL,

UNIQUE(job\_title),

CHECK(min\_salary<=max\_salary)

);

CREATE TABLE employees(

employee\_id INT PRIMARY KEY,

first\_name VARCHAR(25) NOT NULL,

last\_name VARCHAR(25) NOT NULL,

email VARCHAR(25),

phone VARCHAR(20) NOT NULL,

hire\_date DATE DEFAULT SYSDATE NOT NULL,

salary INT NOT NULL,

commission\_pct NUMBER(2,2),

manager\_id REFERENCES employees(employee\_id) ON DELETE SET NULL, --FK

job\_id VARCHAR(20) REFERENCES jobs(job\_id) ON DELETE CASCADE, --FK

UNIQUE(first\_name,last\_name),

UNIQUE(phone),

UNIQUE(email)

);

CREATE TABLE memberships(

membership\_id INT PRIMARY KEY,

name VARCHAR(25) NOT NULL,

price INT NOT NULL,

CHECK(price>0),

UNIQUE(name)

);

CREATE TABLE members(

member\_id INT PRIMARY KEY,

first\_name VARCHAR(25) NOT NULL,

last\_name VARCHAR(25) NOT NULL,

email VARCHAR(25),

phone VARCHAR(20),

join\_date DATE DEFAULT SYSDATE NOT NULL,

membership\_id INT REFERENCES memberships(membership\_id) ON DELETE SET NULL, -- FK

trainer\_id INT REFERENCES employees(employee\_id) ON DELETE SET NULL, -- FK

UNIQUE(first\_name,last\_name),

UNIQUE(phone),

UNIQUE(email)

);

CREATE TABLE payments(

payment\_id INT PRIMARY KEY,

value INT NOT NULL,

payment\_date DATE DEFAULT SYSDATE NOT NULL,

member\_id INT REFERENCES members(member\_id) ON DELETE CASCADE,

CHECK(value>0)

);

CREATE TABLE exercises(

exercise\_id VARCHAR(30) PRIMARY KEY,

name VARCHAR(30) NOT NULL,

Description VARCHAR(300),

UNIQUE(name)

);

CREATE TABLE muscles(

muscle\_id VARCHAR(30) PRIMARY KEY,

name VARCHAR(30) NOT NULL,

location VARCHAR(100)

);

CREATE TABLE equipment(

equipment\_id INT PRIMARY KEY,

name VARCHAR(30) NOT NULL

);

CREATE TABLE musclesWorked(

exercise\_id VARCHAR(30) REFERENCES exercises(exercise\_id) ON DELETE CASCADE,

muscle\_id VARCHAR(30) REFERENCES muscles(muscle\_id) ON DELETE CASCADE,

CONSTRAINT musclesWorked\_PK PRIMARY KEY (exercise\_id, muscle\_id)

);

CREATE TABLE requiredEquip(

equipment\_id INT REFERENCES equipment(equipment\_id) ON DELETE CASCADE,

exercise\_id VARCHAR(30) REFERENCES exercises(exercise\_id) ON DELETE CASCADE,

CONSTRAINT requiredEquip\_PK PRIMARY KEY (equipment\_id,exercise\_id)

);

CREATE TABLE workouts(

workout\_id INT PRIMARY KEY,

name VARCHAR(20) NOT NULL,

description VARCHAR(50),

UNIQUE(name)

);

CREATE TABLE workout\_exercises(

exercise\_id VARCHAR(30) REFERENCES exercises(exercise\_id) ON DELETE CASCADE,

workout\_id INT REFERENCES workouts(workout\_id) ON DELETE CASCADE,

CONSTRAINT workout\_exercises\_PK PRIMARY KEY (exercise\_id, workout\_id)

);

CREATE TABLE classes(

class\_id INT PRIMARY KEY,

name VARCHAR(30),

day\_of\_week VARCHAR(4),

start\_time VARCHAR(5),

employee\_id INT REFERENCES employees(employee\_id) ON DELETE SET NULL,

workout\_id INT REFERENCES workouts(workout\_id) ON DELETE CASCADE

);

CREATE TABLE enrollments(

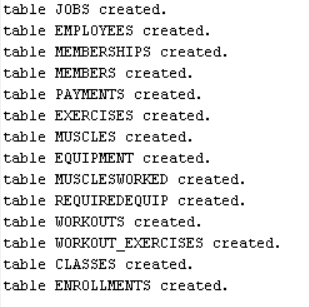
class\_id INT REFERENCES classes(class\_id) ON DELETE CASCADE,

member\_id INT REFERENCES members(member\_id) ON DELETE CASCADE,

CONSTRAINT enrollments\_PK PRIMARY KEY (class\_id, member\_id)

);

### Imagine Rulare



## Popularea tabelelor cu informații coerente

### Codul SQL

INSERT INTO equipment VALUES (1, 'Bench');

INSERT INTO equipment VALUES (2, 'Barbell');

INSERT INTO equipment VALUES (3, 'Squat Rack');

INSERT INTO equipment VALUES (4, 'Dumbbels');

INSERT INTO equipment VALUES (5, 'Leg Press Machine');

INSERT INTO equipment VALUES (6, 'Lat Pulldown Machine');

INSERT INTO equipment VALUES (7, 'Seated Row Machine');

INSERT INTO equipment VALUES (8, 'Shoulder Press Machine');

INSERT INTO equipment VALUES (9, 'Chest Press Machine');

INSERT INTO equipment VALUES (10, 'Lateral Shoulders Machine');

INSERT INTO equipment VALUES (11, 'Pulley System');

INSERT INTO muscles VALUES (1,'Pectorals','Chest');

INSERT INTO muscles VALUES (2,'Quadriceps', 'Most of the frontal leg');

INSERT INTO muscles VALUES (3, 'Biceps', 'Front arm');

INSERT INTO muscles VALUES (4, 'Triceps', 'Opposite of biceps');

INSERT INTO muscles VALUES (5, 'Lateral Shoulders',NULL);

INSERT INTO muscles values (6, 'Abs', 'Between the pubis and the fifth, sixth, and seventh ribs.');

INSERT INTO muscles VALUES (7, 'Hamstrings', 'Back of your legs, right under the glutes');

INSERT INTO muscles VALUES (8, 'Glutes', 'Dorsal area');

INSERT INTO muscles VALUES (9, 'Trapezius','Upper back');

INSERT INTO muscles VALUES (10, 'Spinal Erectors', 'Lower back');

INSERT INTO muscles VALUES (11, 'Latissimus dorsi', 'The large V-shaped muscles that connect your arms to your vertebral column');

INSERT INTO muscles VALUES (12, 'Posterior Shoulders', NULL);

INSERT INTO muscles VALUES (13, 'Anterior Shoulders', NULL);

INSERT INTO exercises VALUES ('LAT\_PULLDOWN','Seated Lat Pulldown','While your arms are extended overhead, depress and retract your scapulae (pull shoulders back and down).');

INSERT INTO exercises VALUES ('TRI\_PUSHDOWN', 'Triceps Pushdown', 'Pull the cable down until the bar touches your thighs and pause to squeeze your triceps at the bottom of the move.');

INSERT INTO exercises VALUES ('SKULL\_CRSH', 'Triceps Skullcrushers', 'Hold the dumbbell with both hands above your chest, straight up, and with the dumbbell shaft in a vertical position.');

INSERT INTO exercises VALUES ('LAT\_SH\_RAISES','Lateral Shoulder Raises','Raise the weights to the sides and up to shoulder level, then lower them again');

INSERT INTO exercises VALUES ('CH\_BAR\_PRESS','Flat Barbell Chest Press', 'Grasp a barbell with an overhand grip just wider than shoulder-width apart.');

INSERT INTO exercises VALUES ('BIC\_DB\_CURL', 'Biceps Dumbell Curl', NULL);

INSERT INTO exercises VALUES ('BIC\_BAR\_CURL', 'Biceps Barbell Curl', 'Using an underhand grip, keep your chest up and your elbows tight to your sides, initiate the move by raising your hands slightly so you feel your biceps become engaged.');

INSERT INTO exercises VALUES ('LG\_PRESS','Leg Press', 'Leg press using the leg press machine');

INSERT INTO exercises VALUES ('BAR\_SQUAT','Barbell Squat', 'Trainees lowers their hips from a standing position and then stands back up.');

INSERT INTO exercises VALUES ('DEADLIFT','Deadlift','A loaded barbell or bar is lifted off the ground to the level of the hips, torso perpendicular to the floor, before being placed back on the ground.');

INSERT INTO exercises VALUES ('SH\_DB\_PRESS', 'Dumbbel Shoulder Press', 'Press the weights directly upwards until your arms are straight and the weights touch above your head.');

INSERT INTO exercises VALUES ('BAR\_ROW','Bent Over Barbell Row', 'Lift the bar from the rack, bend forward at the hips, and keep the back straight with a slight bend in the knees.');

INSERT INTO musclesWorked VALUES ('LAT\_PULLDOWN',11);

INSERT INTO musclesWorked VALUES ('LAT\_PULLDOWN',5);

INSERT INTO musclesWorked VALUES ('TRI\_PUSHDOWN',4);

INSERT INTO musclesWorked VALUES ('SKULL\_CRSH',4);

INSERT INTO musclesWorked VALUES ('LAT\_SH\_RAISES',5);

INSERT INTO musclesWorked VALUES ('LAT\_SH\_RAISES',13);

INSERT INTO musclesWorked VALUES ('CH\_BAR\_PRESS',1);

INSERT INTO musclesWorked VALUES ('CH\_BAR\_PRESS',4);

INSERT INTO musclesWorked VALUES ('CH\_BAR\_PRESS',5);

INSERT INTO musclesWorked VALUES ('CH\_BAR\_PRESS',13);

INSERT INTO musclesWorked VALUES ('SH\_DB\_PRESS',5);

INSERT INTO musclesWorked VALUES ('SH\_DB\_PRESS',13);

INSERT INTO musclesWorked VALUES ('BAR\_SQUAT',2);

INSERT INTO musclesWorked VALUES ('BAR\_SQUAT',7);

INSERT INTO musclesWorked VALUES ('BAR\_SQUAT',8);

INSERT INTO musclesWorked VALUES ('BAR\_SQUAT',6);

INSERT INTO musclesWorked VALUES ('LG\_PRESS',2);

INSERT INTO musclesWorked VALUES ('LG\_PRESS',7);

INSERT INTO musclesWorked VALUES ('LG\_PRESS',8);

INSERT INTO musclesWorked VALUES ('DEADLIFT',2);

INSERT INTO musclesWorked VALUES ('DEADLIFT',10);

INSERT INTO musclesWorked VALUES ('DEADLIFT',9);

INSERT INTO musclesWorked VALUES ('BIC\_DB\_CURL',3);

INSERT INTO musclesWorked VALUES ('BIC\_BAR\_CURL',3);

INSERT INTO musclesWorked VALUES ('BAR\_ROW',11);

INSERT INTO requiredequip VALUES (6,'LAT\_PULLDOWN');

INSERT INTO requiredequip VALUES (11,'TRI\_PUSHDOWN');

INSERT INTO requiredequip VALUES (2,'SKULL\_CRSH');

INSERT INTO requiredequip VALUES (1,'SKULL\_CRSH');

INSERT INTO requiredequip VALUES (8,'LAT\_SH\_RAISES');

INSERT INTO requiredequip VALUES (1,'CH\_BAR\_PRESS');

INSERT INTO requiredequip VALUES (2,'CH\_BAR\_PRESS');

INSERT INTO requiredequip VALUES (4,'BIC\_DB\_CURL');

INSERT INTO requiredequip VALUES (2,'BIC\_BAR\_CURL');

INSERT INTO requiredequip VALUES (5,'LG\_PRESS');

INSERT INTO requiredequip VALUES (3,'BAR\_SQUAT');

INSERT INTO requiredequip VALUES (2,'BAR\_SQUAT');

INSERT INTO requiredequip VALUES (2,'DEADLIFT');

INSERT INTO requiredequip VALUES (1,'SH\_DB\_PRESS');

INSERT INTO requiredequip VALUES (4,'SH\_DB\_PRESS');

INSERT INTO requiredequip VALUES (2,'BAR\_ROW');

INSERT INTO workouts VALUES (1,'Push','A workout focused on pushing exercises');

INSERT INTO workouts VALUES (2,'Pull','A workout focused on pulling exercises');

INSERT INTO workouts VALUES (3,'Legs','Specially designed for massive legs');

INSERT INTO workouts VALUES (4,'Chest-Back-Triceps','Hit your whole upper body');

INSERT INTO workouts VALUES (5,'Shoulders-Biceps','Get boulder-like shoulders');

INSERT INTO workout\_exercises VALUES ('CH\_BAR\_PRESS',1);

INSERT INTO workout\_exercises VALUES ('SKULL\_CRSH',1);

INSERT INTO workout\_exercises VALUES ('SH\_DB\_PRESS',1);

INSERT INTO workout\_exercises VALUES ('LAT\_SH\_RAISES',1);

INSERT INTO workout\_exercises VALUES ('LAT\_PULLDOWN',2);

INSERT INTO workout\_exercises VALUES ('BIC\_DB\_CURL',2);

INSERT INTO workout\_exercises VALUES ('BAR\_ROW',2);

INSERT INTO workout\_exercises VALUES ('BAR\_SQUAT',3);

INSERT INTO workout\_exercises VALUES ('LG\_PRESS',3);

INSERT INTO workout\_exercises VALUES ('DEADLIFT',3);

INSERT INTO memberships VALUES (1,'Bronze',150);

INSERT INTO memberships VALUES (2,'Silver',200);

INSERT INTO memberships VALUES (3,'Golden',230);

INSERT INTO jobs VALUES ('CLN\_STF','Cleaning Staff',1400,2500);

INSERT INTO jobs VALUES ('P\_TRN', 'Personal Trainer', 3000,9000);

INSERT INTO jobs VALUES ('SA\_REP', 'Sales Representative', 1400,5000);

INSERT INTO jobs VALUES ('DESK','Front Desk Person',1400,3000);

INSERT INTO jobs VALUES ('ADM','Administration', 4000,8000);

INSERT INTO employees VALUES (1,'Dorothy','Haney','dorothy@gmail.com','973-722-4746','1-JUN-2010',7000,NULL,NULL,'ADM');

INSERT INTO employees VALUES (2,'Victor','Disney','victor23@gmail.com','563-449-0413','25-DEC-2012',1500,NULL,1,'CLN\_STF');

INSERT INTO employees VALUES (3,'Lillian','Borton','lilly@gmail.com','518-454-9577','20-NOV-2011',7000,0.3,1,'P\_TRN');

INSERT INTO employees VALUES (4,'Sally','Walden','sallynotsalty@yahoo.com','614-352-5238','20-JAN-2011',3500,NULL,3,'P\_TRN');

CREATE SEQUENCE memberseq

START WITH 1

INCREMENT BY 1

NOCYCLE;

INSERT INTO members VALUES (memberseq.nextval, 'Lorraine','Wooldridge','LorWool@gmail.com','803-256-4224','22-NOV-2011',1,NULL);

INSERT INTO members VALUES (memberseq.nextval, 'Shery','Sousa','Shery@gmail.com','832-664-1475','25-NOV-2011',3,3);

INSERT INTO members VALUES (memberseq.nextval, 'Lisa','Barclay','lbarclay@gmail.com','214-277-8993','01-NOV-2011',1,4);

INSERT INTO members VALUES (memberseq.nextval, 'Mindy','Coleman','MinColeman@yahoo.com','860-520-2878','03-FEB-2012',2,NULL);

INSERT INTO classes VALUES (1,'Killer Leg Workout','MON','19:30',3,3);

INSERT INTO classes VALUES (2,'Beach Ready Body','FRI','18:30',3,4);

INSERT INTO classes VALUES (3,'Boulder Shoulders','SUN','12:30',4,5);

INSERT INTO classes VALUES (4,'Beginner Pull Workout','MON','19:30',4,3);

INSERT INTO classes VALUES (5,'Advanced Push Workout','TUE','15:15',3,1);

INSERT INTO enrollments VALUES (1,1);

INSERT INTO enrollments VALUES (2,1);

INSERT INTO enrollments VALUES (5,1);

INSERT INTO enrollments VALUES (3,2);

INSERT INTO enrollments VALUES (5,2);

INSERT INTO enrollments VALUES (4,2);

INSERT INTO enrollments VALUES (3,3);

INSERT INTO enrollments VALUES (5,3);

INSERT INTO enrollments VALUES (4,3);

INSERT INTO enrollments VALUES (1,4);

INSERT INTO enrollments VALUES (2,4);

CREATE SEQUENCE paymentseq

START WITH 1

INCREMENT BY 1

NOCYCLE;

INSERT INTO payments VALUES (paymentseq.nextval,150,'22-NOV-2011',1);

INSERT INTO payments VALUES (paymentseq.nextval,150,'22-DEC-2011',1);

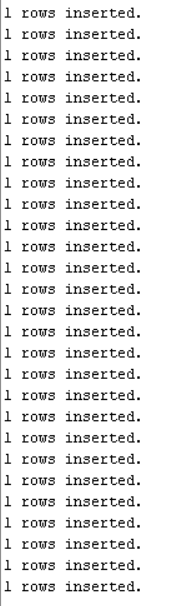
INSERT INTO payments VALUES (paymentseq.nextval,150,'22-JAN-2012',1);

INSERT INTO payments VALUES (paymentseq.nextval,230,'25-NOV-2011',2);

INSERT INTO payments VALUES (paymentseq.nextval,150,'01-NOV-2011',3);

INSERT INTO payments VALUES (paymentseq.nextval,200,'03-FEB-2012',4);

### Imagine Rulare



## Afișarea datelor angajaților folosind un tablou îmbricat ce conține obiecte

### Codul SQL

CREATE OR REPLACE TYPE employee\_typ AS OBJECT (

employee\_id INT,

first\_name VARCHAR(25),

last\_name VARCHAR(25),

email VARCHAR(25),

phone VARCHAR(20),

salary INT,

commission\_pct NUMBER(2,2),

MAP MEMBER FUNCTION getId RETURN NUMBER, -- Pt comparari, nu e folosit, dar voiam sa vad cum merge

MEMBER FUNCTION getSalary RETURN NUMBER,

MEMBER PROCEDURE showData (SELF IN OUT NOCOPY employee\_typ)

-- NOCOPY e asemanator cu & din C++, ii spune compilatorului sa nu mai faca inca o copie obiectului.

-- SELF e asemanator cu this din C++

);

/

CREATE OR REPLACE TYPE BODY employee\_typ AS

MAP MEMBER FUNCTION getId RETURN NUMBER IS

BEGIN

RETURN employee\_id;

END;

MEMBER FUNCTION getSalary RETURN NUMBER IS

BEGIN

RETURN salary\*(1+commission\_pct);

END;

MEMBER PROCEDURE showData (SELF IN OUT NOCOPY employee\_typ) IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('---- Employee ID: '|| employee\_id || ' Full Name: ' || first\_name || ' '||last\_name || ' ----');

DBMS\_OUTPUT.PUT\_LINE('Email: ' || email || ' Phone: ' || phone);

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

END;

END;

/

CREATE OR REPLACE PROCEDURE showEmployees IS

TYPE t\_employee\_typ IS TABLE OF employee\_typ;

t\_emp t\_employee\_typ;

BEGIN

SELECT employee\_typ(employee\_id, first\_name, last\_name, email, phone, salary, commission\_pct)

BULK COLLECT INTO t\_emp FROM employees;

FOR j IN t\_emp.FIRST..t\_emp.LAST LOOP

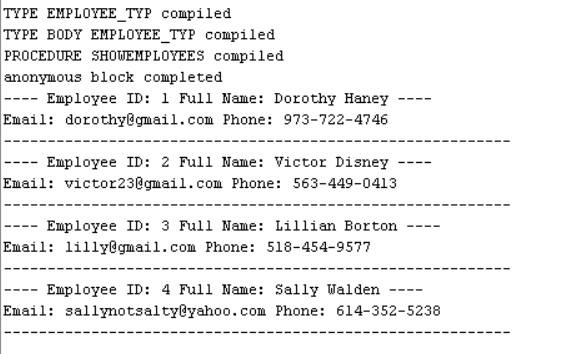
t\_emp(j).showData();

END LOOP;

END showEmployees;

/

### Imagine Rulare



## Modificarea salariilor angajaților folosind un cursor

### Codul SQL

--7 Pt v\_option = 1, se modifica salariul angajatilor cu salariul > threshold, altfel cu salariul < threshold.

CREATE OR REPLACE PROCEDURE modifySalary (v\_option INTEGER, v\_add INTEGER, v\_sal INTEGER)

IS

TYPE empref IS REF CURSOR;

c\_emp empref;

v\_emp employees%ROWTYPE;

BEGIN

IF v\_option = 1 THEN

OPEN c\_emp FOR

'SELECT \* FROM employees WHERE salary > :bind\_var FOR UPDATE OF SALARY NOWAIT'

USING v\_sal;

DBMS\_OUTPUT.PUT\_LINE('Adding ' || v\_add || ' RON to employees whose monthly salary is BIGGER than '||v\_sal || ' RON');

ELSIF v\_option = 2 THEN

OPEN c\_emp FOR

'SELECT \* FROM employees WHERE salary < :bind\_var FOR UPDATE OF SALARY WAIT 15'

USING v\_sal;

DBMS\_OUTPUT.PUT\_LINE('Adding ' || v\_add || ' RON to employees whose monthly salary is SMALLER than '||v\_sal || ' RON');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Invalid Option. Must be 1 for > or 2 for <');

END IF;

LOOP

FETCH c\_emp INTO v\_emp;

EXIT WHEN c\_emp%NOTFOUND;

UPDATE employees

SET salary = salary+v\_add

WHERE employee\_id = v\_emp.employee\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated ID: '||v\_emp.employee\_id|| ' Full Name: ' || v\_emp.first\_name || ' ' || v\_emp.last\_name || ' Salary: ' || v\_emp.salary || ' -> ' || (v\_emp.salary+v\_add));

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Processed '||c\_emp%ROWCOUNT || ' lines.');

CLOSE c\_emp;

END modifySalary;

/

BEGIN

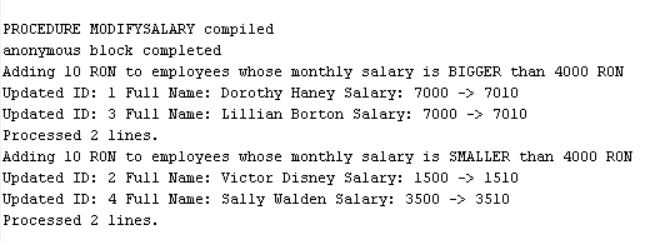
modifysalary(1,10,4000);

modifysalary(2,10,4000);

END;

/

### Imagine Rulare



## Crearea unei funcții ce întoarce numărul de echipamente necesare unui antrenament

### Codul SQL

CREATE OR REPLACE FUNCTION necessaryEquipment(v\_name workouts.name%TYPE)

RETURN NUMBER IS

v\_count INTEGER;

BEGIN

SELECT COUNT(\*)

INTO v\_count

FROM workouts

WHERE workouts.name = v\_name;

--Nu putem avea count > 1 din cauza constrangerii UNIQUE(name)

IF v\_count = 0 THEN

raise NO\_DATA\_FOUND;

END IF;

SELECT COUNT(DISTINCT e.Name)

INTO v\_count

FROM workouts wo, workout\_exercises w, requiredequip r, equipment e, exercises ex

WHERE w.workout\_id= wo.workout\_id AND w.exercise\_id = ex.exercise\_id AND ex.exercise\_id = r.exercise\_id AND r.equipment\_id=e.equipment\_id AND UPPER(wo.name) = UPPER(v\_name);

RETURN v\_count;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20000, 'There is no workout with this name');

END necessaryEquipment;

/

BEGIN

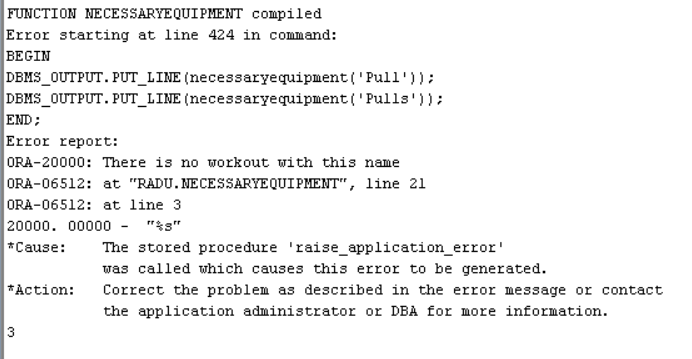
DBMS\_OUTPUT.PUT\_LINE(necessaryequipment('Pull'));

DBMS\_OUTPUT.PUT\_LINE(necessaryequipment('Pulls'));

END;

/

### Imagine Rulare



## Crearea unei proceduri ce afișează mușchii lucrați de fiecare antrenament

### Codul SQL

CREATE OR REPLACE PROCEDURE p\_musclesWorked

IS

v\_count INTEGER;

TYPE refcursor IS REF CURSOR;

CURSOR c\_work IS

SELECT ws.name, CURSOR ( SELECT DISTINCT m.Name

FROM workouts wo, workout\_exercises w, musclesWorked mw, muscles m, exercises ex

WHERE w.workout\_id= wo.workout\_id AND w.exercise\_id = ex.exercise\_id AND ex.exercise\_id = mw.exercise\_id AND mw.muscle\_id=m.muscle\_id AND wo.name =ws.name)

FROM workouts ws;

v\_work\_name workouts.name%TYPE;

v\_cursor refcursor;

v\_muscle\_name muscles.name%TYPE;

BEGIN

OPEN c\_work;

LOOP

FETCH c\_work INTO v\_work\_name, v\_cursor;

EXIT WHEN c\_work%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE ('------ WORKOUT: '||v\_work\_name|| ' ------');

LOOP

FETCH v\_cursor INTO v\_muscle\_name;

EXIT WHEN v\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(v\_cursor%ROWCOUNT || '. '||v\_muscle\_name);

END LOOP;

END LOOP;

CLOSE c\_work;

END p\_musclesWorked;

/

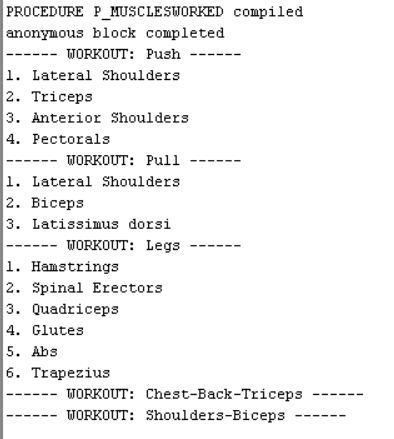
BEGIN

p\_musclesWorked;

END;

/

### Imagine Rulare



## Compound trigger ce limitează numărul maxim de participanți într-o clasă

### Codul SQL

CREATE OR REPLACE TRIGGER limit\_members\_in\_class

FOR INSERT ON enrollments

COMPOUND TRIGGER

max\_number constant number := 3;

TYPE t\_class\_count IS TABLE OF INTEGER INDEX BY BINARY\_INTEGER;

class\_count t\_class\_count;

TYPE t\_class\_IDs IS TABLE OF enrollments.class\_id%TYPE;

c\_class\_IDs t\_class\_IDs;

TYPE t\_counts IS TABLE OF INTEGER;

counts t\_counts;

BEFORE STATEMENT IS

BEGIN

SELECT COUNT(\*), class\_id

BULK COLLECT INTO counts, c\_class\_IDs

FROM enrollments

GROUP BY class\_id;

FOR i IN 1..c\_class\_IDs.COUNT() LOOP

class\_count(c\_class\_IDs(i)) := counts(i);

END LOOP;

END BEFORE STATEMENT;

AFTER EACH ROW IS

BEGIN

IF class\_count(:NEW.class\_id) + 1 > max\_number THEN

DBMS\_OUTPUT.PUT\_LINE('Too many members in Class ID: '||:NEW.class\_id);

DBMS\_OUTPUT.PUT\_LINE('Maximum number of allowed members: '|| max\_number);

RAISE\_APPLICATION\_ERROR(-20004, 'Too many members in the same class');

END IF;

END AFTER EACH ROW;

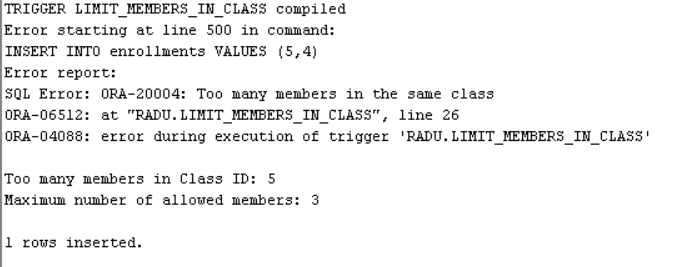
END limit\_members\_in\_class;

/

INSERT INTO enrollments VALUES (5,4); -- Not Working, class 5 is already full

INSERT INTO enrollments VALUES (4,1); -- Working

### Imagine Rulare



## Trigger la nivel de linie ce verifică dacă salariul respectă limitele impuse de job

### Codul SQL

CREATE OR REPLACE TRIGGER check\_bounds

BEFORE INSERT OR UPDATE OF salary ON employees

FOR EACH ROW

DECLARE

v\_min INTEGER;

v\_max INTEGER;

BEGIN

SELECT min\_salary, max\_salary

INTO v\_min, v\_max

FROM employees e, jobs j

WHERE e.job\_id=j.job\_id AND j.job\_id = :NEW.job\_id;

IF (:NEW.salary<v\_min) OR (:NEW.salary > v\_max) THEN

raise\_application\_error(-20002, 'Salary is out of bounds for this job');

END IF;

END;

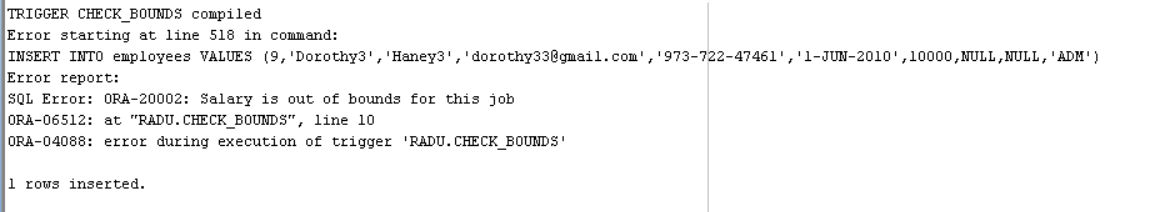
/

-- ADM are min 4000, max 8000

INSERT INTO employees VALUES (9,'Dorothy3','Haney3','dorothy33@gmail.com','973-722-47461','1-JUN-2010',10000,NULL,NULL,'ADM');

INSERT INTO employees VALUES (10,'Dorothy2','Haney2','dorothy32@gmail.com','973-722-47462','1-JUN-2010',5000,NULL,NULL,'ADM');

### Imagine Rulare



## Trigger de tip LDD folosit pentru logging

### Codul SQL

CREATE TABLE audit\_log

( username VARCHAR2(30),

command VARCHAR2(100),

OBJECT\_NAME VARCHAR2(50),

time TIMESTAMP,

ERROR VARCHAR2(100));

CREATE OR REPLACE TRIGGER audit\_everything

AFTER DDL OR SERVERERROR ON SCHEMA

BEGIN

IF SYS.SYSEVENT = 'SERVERERROR' THEN

INSERT INTO audit\_log VALUES (SYS.LOGIN\_USER,'ERROR',SYS.DICTIONARY\_OBJ\_NAME,SYSDATE, DBMS\_STANDARD.SERVER\_ERROR(1)); --Top stack error

ELSE

INSERT INTO audit\_log VALUES (SYS.LOGIN\_USER,SYS.SYSEVENT,SYS.DICTIONARY\_OBJ\_NAME,SYSDATE, NULL);

END IF;

END audit\_everything;

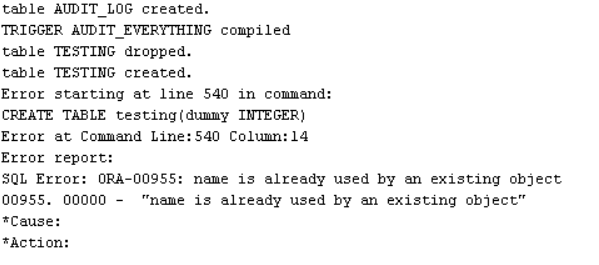
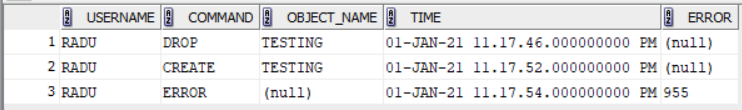
/

DROP TABLE testing;

CREATE TABLE testing(dummy INTEGER);

SELECT \* FROM audit\_log;

### Imagini Rulare

## Definirea pachetului ce conține toate obiectele declarate anterior

### Codul SQL

CREATE OR REPLACE PACKAGE p13 AS

PROCEDURE showEmployees;

PROCEDURE modifySalary (v\_option INTEGER, v\_add INTEGER, v\_sal INTEGER);

FUNCTION necessaryEquipment(v\_name workouts.name%TYPE) RETURN NUMBER;

PROCEDURE p\_musclesWorked;

END p13;

/

CREATE OR REPLACE PACKAGE BODY p13 AS

PROCEDURE showEmployees IS

TYPE t\_employee\_typ IS TABLE OF employee\_typ;

t\_emp t\_employee\_typ;

BEGIN

SELECT employee\_typ(employee\_id, first\_name, last\_name, email, phone, salary, commission\_pct)

BULK COLLECT INTO t\_emp FROM employees;

FOR j IN t\_emp.FIRST..t\_emp.LAST LOOP

t\_emp(j).showData();

END LOOP;

END showEmployees;

PROCEDURE modifySalary (v\_option INTEGER, v\_add INTEGER, v\_sal INTEGER)

IS

TYPE empref IS REF CURSOR;

c\_emp empref;

v\_emp employees%ROWTYPE;

BEGIN

IF v\_option = 1 THEN

OPEN c\_emp FOR

'SELECT \* FROM employees WHERE salary > :bind\_var FOR UPDATE OF SALARY NOWAIT'

USING v\_sal;

DBMS\_OUTPUT.PUT\_LINE('Adding ' || v\_add || ' RON to employees whose monthly salary is BIGGER than '||v\_sal || ' RON');

ELSIF v\_option = 2 THEN

OPEN c\_emp FOR

'SELECT \* FROM employees WHERE salary < :bind\_var FOR UPDATE OF SALARY WAIT 15'

USING v\_sal;

DBMS\_OUTPUT.PUT\_LINE('Adding ' || v\_add || ' RON to employees whose monthly salary is SMALLER than '||v\_sal || ' RON');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Invalid Option. Must be 1 for > or 2 for <');

END IF;

LOOP

FETCH c\_emp INTO v\_emp;

EXIT WHEN c\_emp%NOTFOUND;

UPDATE employees

SET salary = salary+v\_add

WHERE employee\_id = v\_emp.employee\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated ID: '||v\_emp.employee\_id|| ' Full Name: ' || v\_emp.first\_name || ' ' || v\_emp.last\_name || ' Salary: ' || v\_emp.salary || ' -> ' || (v\_emp.salary+v\_add));

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Processed '||c\_emp%ROWCOUNT || ' lines.');

CLOSE c\_emp;

END modifySalary;

FUNCTION necessaryEquipment(v\_name workouts.name%TYPE)

RETURN NUMBER IS

v\_count INTEGER;

BEGIN

SELECT COUNT(\*)

INTO v\_count

FROM workouts

WHERE workouts.name = v\_name;

--Nu putem avea count > 1 din cauza constrangerii UNIQUE(name)

IF v\_count = 0 THEN

raise NO\_DATA\_FOUND;

END IF;

SELECT COUNT(DISTINCT e.Name)

INTO v\_count

FROM workouts wo, workout\_exercises w, requiredequip r, equipment e, exercises ex

WHERE w.workout\_id= wo.workout\_id AND w.exercise\_id = ex.exercise\_id AND ex.exercise\_id = r.exercise\_id AND r.equipment\_id=e.equipment\_id AND UPPER(wo.name) = UPPER(v\_name);

RETURN v\_count;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20000, 'There is no workout with this name');

END necessaryEquipment;

PROCEDURE p\_musclesWorked

IS

v\_count INTEGER;

TYPE refcursor IS REF CURSOR;

CURSOR c\_work IS

SELECT ws.name, CURSOR ( SELECT DISTINCT m.Name

FROM workouts wo, workout\_exercises w, musclesWorked mw, muscles m, exercises ex

WHERE w.workout\_id= wo.workout\_id AND w.exercise\_id = ex.exercise\_id AND ex.exercise\_id = mw.exercise\_id AND mw.muscle\_id=m.muscle\_id AND wo.name =ws.name)

FROM workouts ws;

v\_work\_name workouts.name%TYPE;

v\_cursor refcursor;

v\_muscle\_name muscles.name%TYPE;

BEGIN

OPEN c\_work;

LOOP

FETCH c\_work INTO v\_work\_name, v\_cursor;

EXIT WHEN c\_work%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE ('------ WORKOUT: '||v\_work\_name|| ' ------');

LOOP

FETCH v\_cursor INTO v\_muscle\_name;

EXIT WHEN v\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(v\_cursor%ROWCOUNT || '. '||v\_muscle\_name);

END LOOP;

END LOOP;

CLOSE c\_work;

END p\_musclesWorked;

END p13;

/

### Imagine Rulare



## Crearea unui pachet ce facilitează managerierea membrilor

### Codul SQL

--14 Un pachet folosit la gestionarea membrilor

-- functie care primeste nume, prenume antrenor si returneaza ID --hidden

-- functie care primeste nume, prenume membru si returneaza ID --hidden

-- functie care primeste nume abonament si returneaza ID

-- functie care primeste o zi din saptamana, o ora de inceput si returneaza o lista de clase ( TYPE class\_list) -- public

-- Nth most popular trainer -- public

-- procedura care modifica antrenorul unui membru -- public

-- proceduri pt upgrade/downgrade/change la abonamentul unui membru si adauga automat plata --public (se presupune ca ID-urile sunt ordonate crescator)

-- proceduri pt delete member (overloaded dupa id, nume+prenume) -- public

-- functie pt add member ce intoarce noul ID -- public

CREATE OR REPLACE PACKAGE member\_manager AS

TYPE class\_list IS TABLE OF classes%ROWTYPE;

FUNCTION getClasses (v\_day VARCHAR2, v\_hour VARCHAR2) RETURN class\_list;

FUNCTION nthPopular (n NUMBER) RETURN employees.employee\_id%TYPE;

PROCEDURE changeTrainer(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE, v\_first\_e employees.first\_name%TYPE, v\_last\_e employees.last\_name%TYPE);

PROCEDURE upgradeMembership(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE);

PROCEDURE downgradeMembership(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE);

PROCEDURE deleteMember(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE);

PROCEDURE deleteMember(v\_id members.member\_id%TYPE);

FUNCTION addMember(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE,v\_email members.email%TYPE,v\_phone members.phone%TYPE,v\_membership members.membership\_id%TYPE,v\_trainer members.trainer\_id%TYPE)

RETURN members.member\_id%TYPE;

PROCEDURE changeMembership(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE, v\_membership\_name memberships.name%TYPE);

END member\_manager;

/

CREATE OR REPLACE PACKAGE BODY member\_manager AS

FUNCTION getMembershipId(v\_membership memberships.name%TYPE) RETURN memberships.membership\_id%TYPE

IS

v\_id memberships.membership\_id%TYPE;

BEGIN

SELECT membership\_id INTO v\_ID FROM memberships WHERE UPPER(name) = UPPER(v\_membership);

RETURN v\_ID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20000, 'There is no membership with this name');

-- nu pot avea TOO MANY ROWS din cauza constrangerii de UNIQUE(name)

END getMembershipId;

FUNCTION getEmployeeId (v\_first employees.first\_name%TYPE, v\_last employees.last\_name%TYPE)

RETURN employees.employee\_id%TYPE IS

v\_ID employees.employee\_id%TYPE;

BEGIN

SELECT employee\_id INTO v\_ID FROM employees WHERE UPPER(first\_name) = UPPER(v\_first) AND UPPER(last\_name) = UPPER(v\_last);

RETURN v\_ID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20000, 'There is no employee with this name');

-- nu pot avea TOO MANY ROWS din cauza constrangerii de UNIQUE(first\_name,last\_name)

END getEmployeeId;

FUNCTION getMemberId (v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE)

RETURN members.member\_id%TYPE IS

v\_ID members.member\_id%TYPE;

BEGIN

SELECT member\_id INTO v\_ID FROM members WHERE UPPER(first\_name) = UPPER(v\_first) AND UPPER(last\_name) = UPPER(v\_last);

RETURN v\_ID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20001, 'There is no member with this name');

-- nu pot avea TOO MANY ROWS din cauza constrangerii de UNIQUE(first\_name,last\_name)

END getMemberId;

FUNCTION getClasses (v\_day VARCHAR2, v\_hour VARCHAR2) RETURN class\_list IS

t\_classes class\_list;

BEGIN

SELECT \* BULK COLLECT INTO t\_classes FROM classes WHERE UPPER(v\_day) = day\_of\_week AND start\_time>v\_hour;

IF t\_classes.COUNT = 0 THEN

RAISE NO\_DATA\_FOUND;

END IF;

RETURN t\_classes;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'There is no class in the requested day that starts after the given hour');

END getClasses;

FUNCTION nthPopular (n NUMBER) RETURN employees.employee\_id%TYPE IS

v\_id employees.employee\_id%TYPE;

CURSOR desc\_trainer IS

SELECT trainer\_id FROM members WHERE trainer\_id IS NOT NULL GROUP BY trainer\_id ORDER BY COUNT(\*) DESC;

BEGIN

OPEN desc\_trainer;

FOR i IN 1..n LOOP

FETCH desc\_trainer INTO v\_id;

END LOOP;

IF desc\_trainer%ROWCOUNT < n THEN

DBMS\_OUTPUT.PUT\_LINE('N is bigger than the number of trainers, returning id of the least popular trainer');

END IF;

CLOSE desc\_trainer;

RETURN v\_id;

END nthPopular;

PROCEDURE changeTrainer(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE, v\_first\_e employees.first\_name%TYPE, v\_last\_e employees.last\_name%TYPE) IS

v\_member\_id members.member\_id%TYPE := getmemberid(v\_first,v\_last);

v\_trainer\_id employees.employee\_id%TYPE := getemployeeid(v\_first\_e,v\_last\_e);

v\_old\_trainer\_id employees.employee\_id%TYPE;

BEGIN

SELECT trainer\_id INTO v\_old\_trainer\_id FROM members WHERE member\_id = v\_member\_id;

UPDATE members SET trainer\_id = v\_trainer\_id WHERE member\_id = v\_member\_id;

IF v\_old\_trainer\_id IS NOT NULL THEN

DBMS\_OUTPUT.PUT\_LINE('Trainer changed sucessfully from '|| v\_old\_trainer\_id || ' to ' || v\_trainer\_id || '!');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Trainer ' || v\_trainer\_id || ' sucessfully assigned!');

END IF;

END changeTrainer;

PROCEDURE upgradeMembership(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE) IS

v\_member\_id members.member\_id%TYPE := getmemberid(v\_first,v\_last);

v\_currentMembership members.membership\_id%TYPE;

v\_maxMembership members.membership\_id%TYPE;

v\_value payments.value%TYPE;

BEGIN

SELECT MAX(membership\_id) INTO v\_maxMembership FROM memberships;

SELECT membership\_id INTO v\_currentMembership FROM members WHERE member\_id = v\_member\_id;

IF v\_currentmembership < v\_maxMembership THEN

UPDATE members SET membership\_id = membership\_id+1 WHERE member\_id = v\_member\_id;

SELECT price INTO v\_value FROM memberships WHERE membership\_id = v\_currentMembership+1;

INSERT INTO payments VALUES (paymentseq.nextval,v\_value,SYSDATE,v\_member\_id);

DBMS\_OUTPUT.PUT\_LINE('Membership upgraded sucessfully!');

ELSE

RAISE\_APPLICATION\_ERROR(-20004,'Member already has the best available membership');

END IF;

END upgradeMembership;

PROCEDURE downgradeMembership(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE) IS

v\_member\_id members.member\_id%TYPE := getmemberid(v\_first,v\_last);

v\_currentMembership members.membership\_id%TYPE;

v\_value payments.value%TYPE;

BEGIN

SELECT membership\_id INTO v\_currentMembership FROM members WHERE member\_id = v\_member\_id;

IF v\_currentMembership > 1 THEN

UPDATE members SET membership\_id = membership\_id-1 WHERE member\_id = v\_member\_id;

SELECT price INTO v\_value FROM memberships WHERE membership\_id = v\_currentMembership;

INSERT INTO payments VALUES (paymentseq.nextval,v\_value,SYSDATE,v\_member\_id);

DBMS\_OUTPUT.PUT\_LINE('Membership downgraded sucessfully!');

ELSE

RAISE\_APPLICATION\_ERROR(-20005,'Member already has the lowest available membership');

END IF;

END downgradeMembership;

PROCEDURE deleteMember(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE) IS

v\_member\_id members.member\_id%TYPE := getmemberid(v\_first,v\_last);

BEGIN

DELETE FROM members WHERE member\_id = v\_member\_id;

DBMS\_OUTPUT.PUT\_LINE('Member deleted sucessfully!');

END deleteMember;

PROCEDURE deleteMember(v\_id members.member\_id%TYPE) IS

BEGIN

DELETE FROM members WHERE member\_id = v\_id;

DBMS\_OUTPUT.PUT\_LINE('Member deleted by id sucessfully!');

END deleteMember;

FUNCTION addMember(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE,v\_email members.email%TYPE,v\_phone members.phone%TYPE,v\_membership members.membership\_id%TYPE,v\_trainer members.trainer\_id%TYPE)

RETURN members.member\_id%TYPE IS

v\_newId members.member\_id%TYPE := memberseq.nextval;

BEGIN

INSERT INTO members VALUES (v\_newId,v\_first,v\_last,v\_email,v\_phone,SYSDATE,v\_membership,v\_trainer);

RETURN v\_newId;

END addMember;

PROCEDURE changeMembership(v\_first members.first\_name%TYPE, v\_last members.last\_name%TYPE, v\_membership\_name memberships.name%TYPE) IS

v\_membership\_id memberships.membership\_id%TYPE:= getMembershipId(v\_membership\_name);

v\_member\_id members.member\_id%TYPE := getmemberid(v\_first,v\_last);

v\_value payments.value%TYPE;

BEGIN

UPDATE members SET membership\_id = v\_membership\_id WHERE member\_id = v\_member\_id;

SELECT price INTO v\_value FROM memberships WHERE membership\_id = v\_membership\_id;

INSERT INTO payments VALUES (paymentseq.nextval,v\_value,SYSDATE,v\_member\_id);

DBMS\_OUTPUT.PUT\_LINE('Membership changed sucessfully!');

END changeMembership;

END member\_manager;

/

DECLARE

t\_classes member\_manager.class\_list := member\_manager.getClasses('MON','19:00');

v\_id NUMBER;

BEGIN

FOR j in t\_classes.FIRST..t\_classes.LAST LOOP

DBMS\_OUTPUT.PUT\_LINE(t\_classes(j).name);

END LOOP;

-- t\_classes := member\_manager.getClasses('MON','20:00'); -- Eroare deoarece nu exista nicio clasa care sa respecte conditiile

DBMS\_OUTPUT.PUT\_LINE(member\_manager.nthPopular(2));

DBMS\_OUTPUT.PUT\_LINE(member\_manager.nthPopular(5)); -- Afiseaza un warning si returneaza antrenorul cu cei mai putini clienti deoarece nu exista destui antrenori

member\_manager.changeTrainer('Lorraine','Wooldridge','Lillian','Borton'); -- Daca nu exista antrenorul/membrul o sa dea eroare din cauza functiilor getMemberId si getEmployeeId

member\_manager.downgradeMembership('Shery','Sousa'); -- Daca are deja minimul o sa dea eroare

member\_manager.upgradeMembership('Mindy','Coleman'); -- Daca are deja maximul o sa dea eroare

member\_manager.changeMembership('Mindy','Coleman','Bronze'); -- Daca nu exista abonamentul o sa dea eroare

member\_manager.deleteMember('Shery','Sousa');

member\_manager.deleteMember(1);

END;

/

### Imagine Rulare

