**PROJECT – DATABASES**

**COORDINATING PROFESSOR:**

**DIACONITA VLAD**

**STUDENT:**

**BOJAN VLAD-CRISTIAN**

***SECOND-HAND RETAIL STORE FOR LAPTOPS***

This beginner-level project is building upon the foundation, which is the database schema I created last semester, crafting a simple Second-Hand Laptop Retail Database and performing fundamental PL/SQL operations on it. While it may not emulate professional systems, it showcases my initial exploration of the topic.

The idea of the website whose database I have created is based on customers being able to either buy a laptop from our offer or sell us a personal device they don’t want anymore.

On the next page you will see the database schema generated using Oracle SQL Developer’s Data Modeler:

***DATABASE SCHEMA***

(unmodified from the last semester)

***A screenshot of a computer

Description automatically generated***

**PL/SQL BLOCKS**

**Requirement 1:**  *Package used to interact with customers.*

**Solution:**

*CREATE OR REPLACE PACKAGE customer\_pkg IS*

*PROCEDURE display\_customer (p\_customer\_id IN customers.customer\_id%TYPE);*

*FUNCTION get\_customer\_orders (p\_customer\_id IN customers.customer\_id%TYPE) RETURN NUMBER;*

*END customer\_pkg;*

*/*

*CREATE OR REPLACE PACKAGE BODY customer\_pkg IS*

*PROCEDURE display\_customer (p\_customer\_id IN customers.customer\_id%TYPE) IS*

*v\_email customers.email%TYPE;*

*v\_phone\_number customers.phone\_number%TYPE;*

*v\_first\_name customers.first\_name%TYPE;*

*v\_last\_name customers.last\_name%TYPE;*

*BEGIN*

*SELECT email, phone\_number, first\_name, last\_name*

*INTO v\_email, v\_phone\_number, v\_first\_name, v\_last\_name*

*FROM customers*

*WHERE customer\_id = p\_customer\_id;*

*DBMS\_OUTPUT.PUT\_LINE(*

*v\_email || ' ' ||*

*v\_phone\_number || ' ' ||*

*v\_first\_name || ' ' ||*

*v\_last\_name*

*);*

*END display\_customer;*

*FUNCTION get\_customer\_orders (p\_customer\_id IN customers.customer\_id%TYPE) RETURN NUMBER IS*

*v\_orders\_counter NUMBER;*

*BEGIN*

*SELECT COUNT(order\_id)*

*INTO v\_orders\_counter*

*FROM orders*

*WHERE customer\_id = p\_customer\_id;*

*RETURN v\_orders\_counter;*

*END get\_customer\_orders;*

*END customer\_pkg;*

*/*

*-- Display my name*

*BEGIN*

*customer\_pkg.display\_customer(1);*

*END;*

*/*

**Screenshot:**

**A close-up of a computer screen

Description automatically generated**

**Requirement 2:** *Return the customer with the most orders.*

**Solution:**

*CREATE OR REPLACE FUNCTION get\_best\_customer RETURN customers.customer\_id%TYPE IS*

*v\_customer\_id orders.customer\_id%TYPE;*

*BEGIN*

*SELECT customer\_id*

*INTO v\_customer\_id*

*FROM (*

*SELECT customer\_id*

*FROM orders*

*GROUP BY customer\_id*

*ORDER BY COUNT(order\_id) DESC*

*)*

*FETCH FIRST 1 ROWS ONLY;*

*RETURN v\_customer\_id;*

*END;*

*/*

*DECLARE*

*v\_customer\_id customers.customer\_id%TYPE;*

*BEGIN*

*v\_customer\_id := get\_best\_customer();*

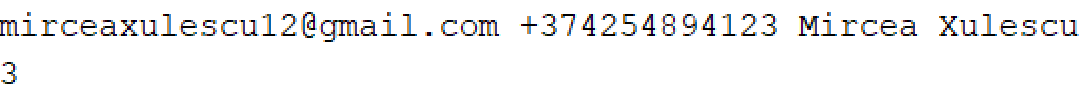
*customer\_pkg.display\_customer(v\_customer\_id);*

*DBMS\_OUTPUT.PUT\_LINE(customer\_pkg.get\_customer\_orders(v\_customer\_id));*

*END;*

*/*

**Screenshot:**

****

**Requirement 3:** *Increase sell price of every laptop with the inflation rate.*

**Solution:**

*CREATE OR REPLACE PROCEDURE adjust\_to\_inflation (p\_inflation\_rate IN NUMBER) IS*

*CURSOR c\_laptops IS*

*SELECT price\_sell*

*FROM laptops*

*FOR UPDATE;*

*v\_price\_sell laptops.price\_sell%TYPE;*

*BEGIN*

*OPEN c\_laptops;*

*LOOP*

*FETCH c\_laptops INTO v\_price\_sell;*

*EXIT WHEN c\_laptops%NOTFOUND;*

*UPDATE laptops*

*SET price\_sell = v\_price\_sell \* (1 + p\_inflation\_rate)*

*WHERE CURRENT OF c\_laptops;*

*END LOOP;*

*CLOSE c\_laptops;*

*END;*

*/*

*BEGIN*

*adjust\_to\_inflation(0.1);*

*END;*

*/*

**Screenshot:**

**A white background with black text

Description automatically generated**

**Requirement 4:** *Procedure to display the size of a laptop based on the display size.*

**Solution:**

*CREATE OR REPLACE PROCEDURE laptop\_size (v\_laptop\_id IN laptops.laptop\_id%TYPE) IS*

*v\_display\_size laptopmodels.display\_size%TYPE;*

*v\_model\_id laptops.model\_id%TYPE;*

*BEGIN*

*SELECT model\_id*

*INTO v\_model\_id*

*FROM laptops*

*WHERE laptop\_id = v\_laptop\_id;*

*SELECT display\_size*

*INTO v\_display\_size*

*FROM laptopmodels*

*WHERE model\_id = v\_model\_id;*

*CASE*

*WHEN v\_display\_size < 15 THEN DBMS\_OUTPUT.PUT\_LINE('Small');*

*WHEN v\_display\_size < 16 THEN DBMS\_OUTPUT.PUT\_LINE('Medium');*

*ELSE DBMS\_OUTPUT.PUT\_LINE('Big');*

*END CASE;*

*EXCEPTION*

*WHEN NO\_DATA\_FOUND*

*THEN DBMS\_OUTPUT.PUT\_LINE('NO\_DATA\_FOUND');*

*WHEN TOO\_MANY\_ROWS*

*THEN DBMS\_OUTPUT.PUT\_LINE('TOO\_MANY\_ROWS');*

*WHEN VALUE\_ERROR*

*THEN DBMS\_OUTPUT.PUT\_LINE('VALUE\_ERROR');*

*END;*

*/*

*BEGIN*

*laptop\_size(1);*

*END;*

*/*

**Screenshot:**

**A black text on a white background

Description automatically generated**

**Requirement 5:** *Function to return the number of times a laptop model was ordered.*

**Solution:**

*CREATE OR REPLACE FUNCTION number\_of\_orders (v\_model\_id IN laptops.model\_id%TYPE) RETURN NUMBER IS*

*v\_number\_of\_orders NUMBER;*

*v\_counter NUMBER;*

*inexistent\_model EXCEPTION;*

*BEGIN*

*SELECT COUNT(\*)*

*INTO v\_counter*

*FROM laptopmodels*

*WHERE model\_id = v\_model\_id;*

*IF v\_counter = 0 THEN*

*RAISE inexistent\_model;*

*END IF;*

*SELECT COUNT(l.model\_id)*

*INTO v\_number\_of\_orders*

*FROM orders o*

*JOIN orderslaptops ol on ol.order\_id = o.order\_id*

*JOIN laptops l on l.laptop\_id = ol.laptop\_id*

*WHERE o.order\_type = 'Buy' AND l.model\_id = v\_model\_id*

*GROUP BY l.model\_id;*

*RETURN v\_number\_of\_orders;*

*EXCEPTION*

*WHEN inexistent\_model*

*THEN DBMS\_OUTPUT.PUT\_LINE('The given model\_id doesn''t exist!');*

*WHEN NO\_DATA\_FOUND*

*THEN RETURN 0;*

*END;*

*/*

*DECLARE*

*orders\_counter NUMBER;*

*BEGIN*

*orders\_counter := number\_of\_orders(2);*

*DBMS\_OUTPUT.PUT\_LINE('For the given laptop model, there are ' || orders\_counter || ' orders');*

*END;*

*/*

**Requirement 6:** *Function to return the most ordered laptop model.*

**Solution:**

*CREATE OR REPLACE FUNCTION most\_ordered\_model RETURN laptopmodels.model\_id%TYPE IS*

*CURSOR c\_laptop\_models IS*

*SELECT model\_id*

*FROM laptopmodels;*

*v\_best\_seller laptopmodels.model\_id%TYPE;*

*v\_best\_seller\_orders NUMBER := 0;*

*v\_current\_model\_orders NUMBER;*

*BEGIN*

*FOR i IN c\_laptop\_models LOOP*

*v\_current\_model\_orders := number\_of\_orders(i.model\_id);*

*IF v\_current\_model\_orders > v\_best\_seller\_orders THEN*

*v\_best\_seller := i.model\_id;*

*v\_best\_seller\_orders := v\_current\_model\_orders;*

*END IF;*

*END LOOP;*

*RETURN v\_best\_seller;*

*END;*

*/*

*DECLARE*

*v\_model\_id laptopmodels.model\_id%TYPE;*

*BEGIN*

*v\_model\_id := most\_ordered\_model();*

*DBMS\_OUTPUT.PUT\_LINE('The most popular model is ' || v\_model\_id);*

*END;*

*/*

**Requirement 7:** *Raise the price of the most popular(most ordered) laptop by 10%.*

**Solution:**

*CREATE OR REPLACE PROCEDURE raise\_price\_of\_best\_seller IS*

*v\_most\_ordered\_model laptopmodels.model\_id%TYPE;*

*BEGIN*

*v\_most\_ordered\_model := most\_ordered\_model;*

*UPDATE laptops*

*SET price\_sell = price\_sell \* 1.1*

*WHERE model\_id = v\_most\_ordered\_model;*

*END;*

*/*

*BEGIN*

*raise\_price\_of\_best\_seller();*

*END;*

*/*

**Requirement 8:** *Display laptops within a certain price range.*

**Solution:**

*CREATE OR REPLACE PROCEDURE display\_laptops (v\_min\_price IN NUMBER, v\_max\_price IN NUMBER) IS*

*CURSOR c\_laptops(p\_min\_price NUMBER, p\_max\_price NUMBER) IS*

*SELECT laptop\_id, model\_id, price\_sell*

*FROM laptops*

*WHERE price\_sell BETWEEN p\_min\_price AND p\_max\_price;*

*v\_laptops\_row c\_laptops%ROWTYPE;*

*BEGIN*

*OPEN c\_laptops(v\_min\_price, v\_max\_price);*

*LOOP*

*FETCH c\_laptops INTO v\_laptops\_row;*

*EXIT WHEN c\_laptops%NOTFOUND;*

*DBMS\_OUTPUT.PUT\_LINE('Laptop ID: ' || v\_laptops\_row.laptop\_id || '; ' ||*

*'Model ID: ' || v\_laptops\_row.model\_id || '; ' ||*

*'Price: ' || v\_laptops\_row.price\_sell);*

*END LOOP;*

*CLOSE c\_laptops;*

*END;*

*/*

*BEGIN*

*display\_laptops(1000,5000);*

*END;*

*/*

**Requirement 9:** *Store and retrieve the id and phone number of each customer between an interval of ids(index by table).*

**Solution:**

*CREATE OR REPLACE PROCEDURE manage\_customers\_phones (p\_id\_left IN customers.customer\_id%TYPE, p\_id\_right IN customers.customer\_id%TYPE) IS*

*TYPE phones\_table IS TABLE OF customers.phone\_number%TYPE INDEX BY PLS\_INTEGER;*

*t\_phones phones\_table;*

*i customers.email%TYPE;*

*BEGIN*

*SELECT phone\_number*

*BULK COLLECT INTO t\_phones*

*FROM customers*

*WHERE customer\_id BETWEEN p\_id\_left AND p\_id\_right;*

*i := t\_phones.FIRST;*

*WHILE i IS NOT NULL LOOP*

*DBMS\_OUTPUT.PUT\_LINE(i || ' -> ' || t\_phones(i));*

*i := t\_phones.NEXT(i);*

*END LOOP;*

*END;*

*/*

*BEGIN*

*manage\_customers\_phones(1,3);*

*END;*

*/*

**Requirement 10:** *Store and retrieve multiple reviews for a given laptop model(nested table).*

**Solution:**

*CREATE OR REPLACE PROCEDURE manage\_reviews (v\_model\_id IN laptops.model\_id%TYPE) IS*

*TYPE reviews\_table IS TABLE OF reviews.review\_id%TYPE;*

*t\_reviews reviews\_table;*

*BEGIN*

*SELECT review\_id*

*BULK COLLECT INTO t\_reviews*

*FROM reviews*

*WHERE model\_id = v\_model\_id;*

*FOR i IN 1 .. t\_reviews.COUNT LOOP*

*DBMS\_OUTPUT.PUT\_LINE(i || ' -> ' || t\_reviews(i));*

*END LOOP;*

*END;*

*/*

*BEGIN*

*manage\_reviews(2);*

*END;*

*/*

**Requirement 11:** *Store and retrieve the first <n> lowest star reviews for a certain laptop model(varray).*

**Solution:**

*CREATE OR REPLACE PROCEDURE display\_lowest\_ratings (v\_model\_id IN laptops.model\_id%TYPE, n IN NUMBER) IS*

*TYPE reviews\_varray IS VARRAY(100) OF reviews.review\_id%TYPE;*

*varray\_reviews reviews\_varray;*

*BEGIN*

*IF n > 100 THEN*

*-- Handle the situation where n exceeds the maximum size of the VARRAY*

*RAISE\_APPLICATION\_ERROR(-20001, 'Requested number of reviews exceeds maximum capacity.');*

*END IF;*

*SELECT review\_id*

*BULK COLLECT INTO varray\_reviews*

*FROM reviews*

*WHERE model\_id = v\_model\_id AND ROWNUM <= n*

*ORDER BY stars;*

*FOR i IN 1 .. varray\_reviews.COUNT LOOP*

*DBMS\_OUTPUT.PUT\_LINE('Review ID -> ' || varray\_reviews(i));*

*END LOOP;*

*END;*

*/*

*BEGIN*

*display\_lowest\_ratings(5,3);*

*END;*

*/*

**Requirement 12:** *Trigger used to log updates on the laptops table.*

**Solution:**

*CREATE OR REPLACE TRIGGER log\_updates*

*AFTER UPDATE ON laptops*

*BEGIN*

*DBMS\_OUTPUT.PUT\_LINE('A record has been updated!');*

*END;*

*/*

**Requirement 13:** *Trigger used to check if the prices for buy and sell are valid after an insert or update on the laptops table.*

**Solution:**

*CREATE OR REPLACE TRIGGER check\_laptop\_price*

*BEFORE INSERT OR UPDATE ON laptops*

*FOR EACH ROW*

*BEGIN*

*IF :NEW.price\_buy > :NEW.price\_sell THEN*

*RAISE\_APPLICATION\_ERROR(-20000, 'Buying price cannot be higher than selling price!');*

*END IF;*

*END;*

*/*