PROJECT – DATABASES

COORDINATING PROFESSOR:

DIACONITA VLAD

STUDENT:

BOJAN VLAD-CRISTIAN

SECOND-HAND RETAIL STORE FOR LAPTOPS

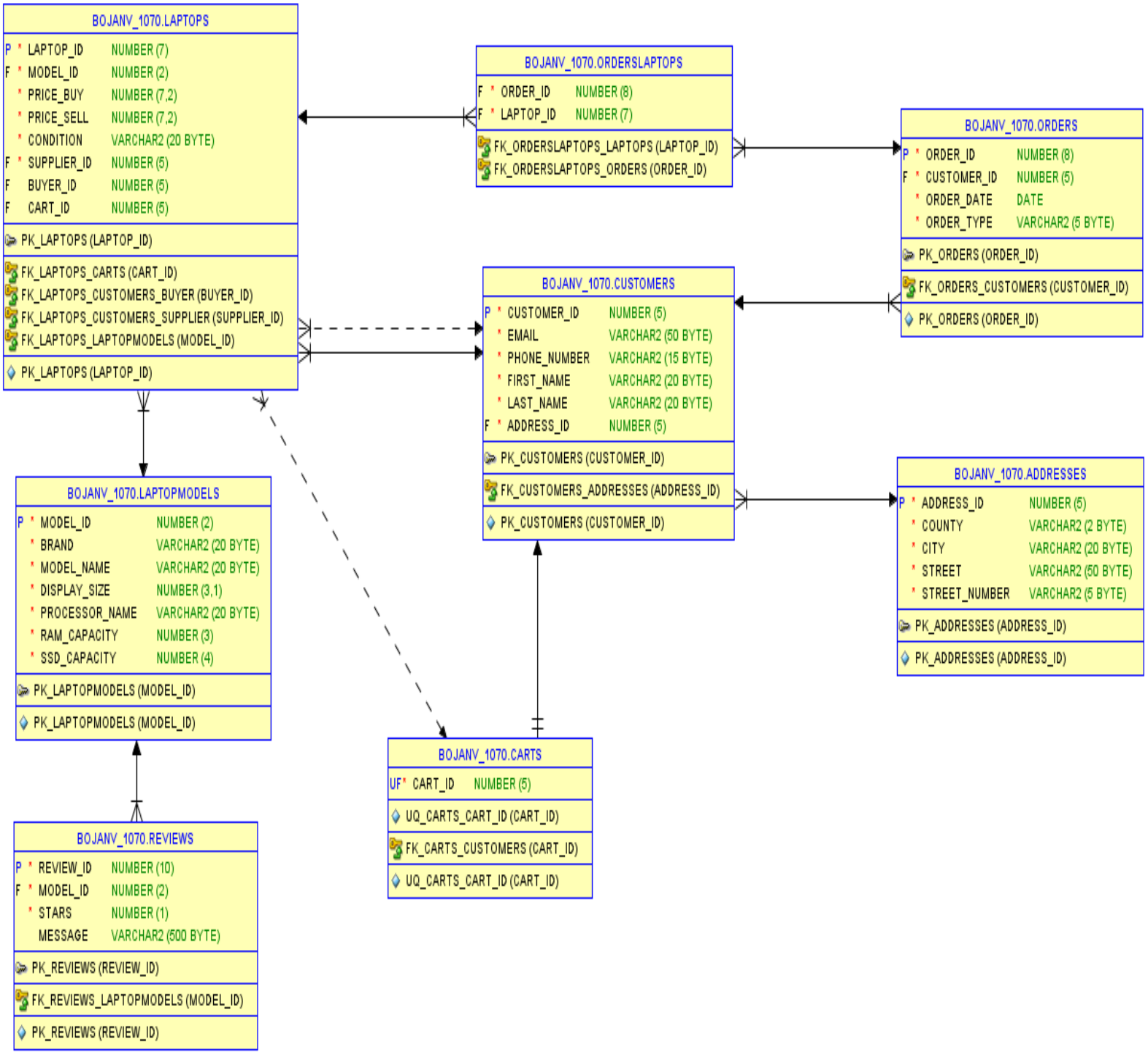
This beginner-level project is building upon the foundation, which is the database schema I created last semester, crafting a simple Second-Hand Laptop Retail Database and performing fundamental PL/SQL operations on it. While it may not emulate professional systems, it showcases my initial exploration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



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:

```
vladimir.bojansky@gmail.com +40761231231 Vlad Bojan


PL/SQL procedure successfully completed.
```

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:

```
mirceaxulescu12@gmail.com +374254894123 Mircea Xulescu
3
```

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 record has been updated!
A record has been updated!
A record has been updated!
A record has been updated!
A record has been updated!


PL/SQL procedure successfully completed.
```

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:

```
Small


PL/SQL procedure successfully completed.
```

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;
/
```