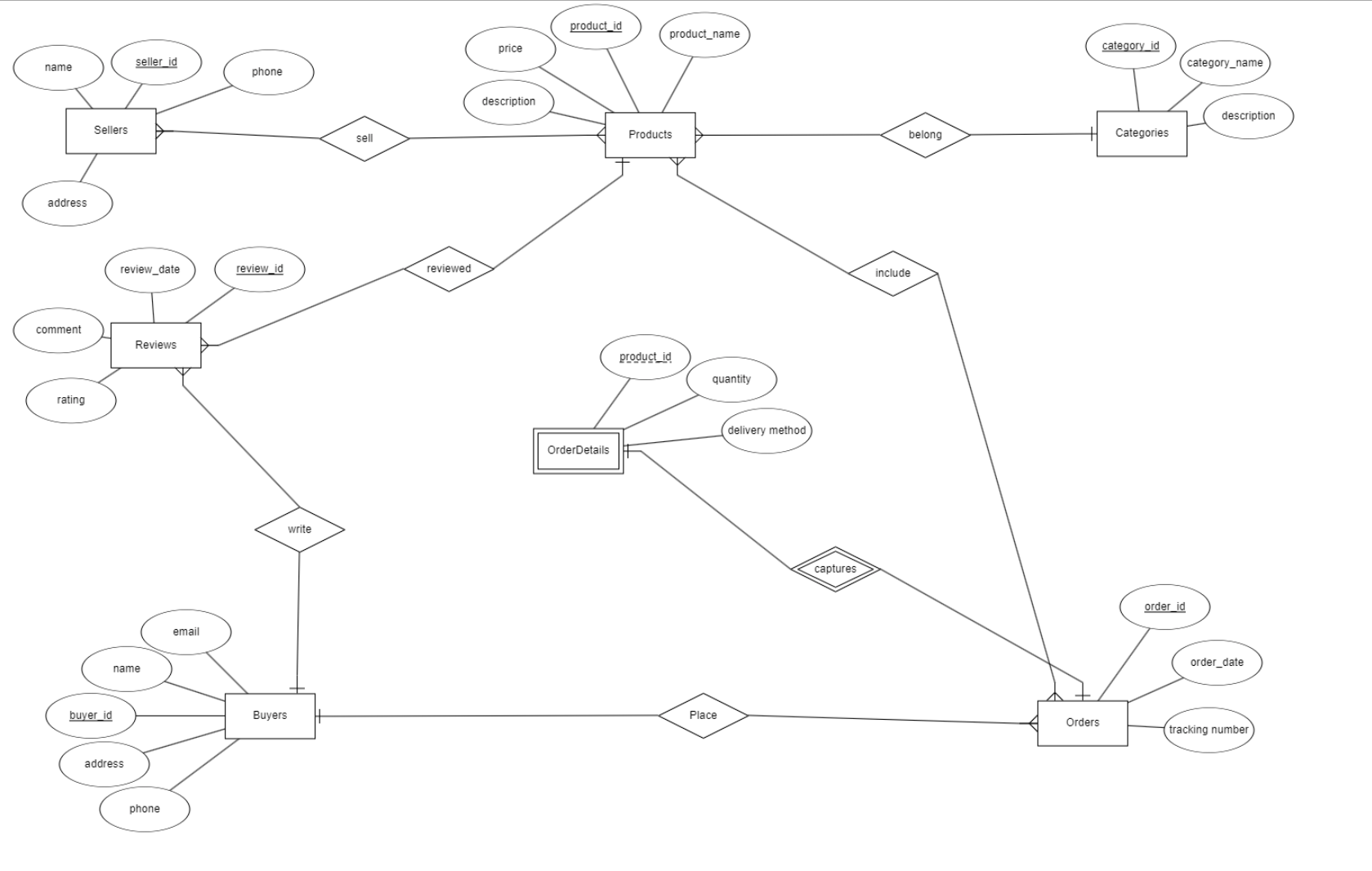
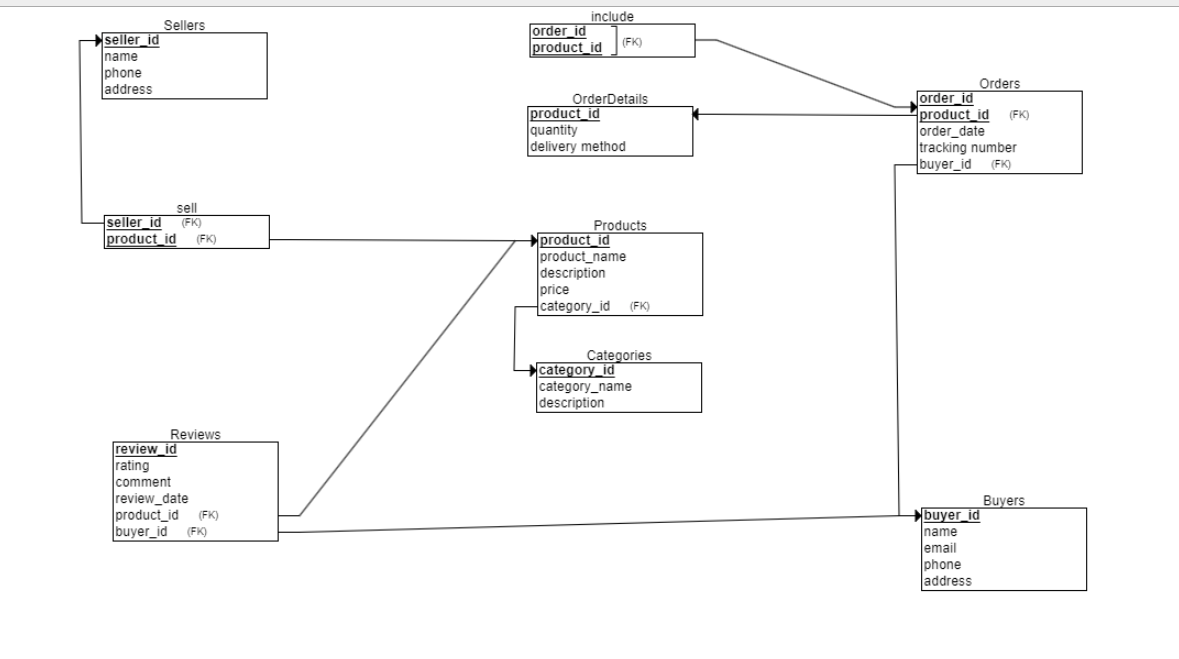
טבלאות





יצירת טבלאות

CREATE TABLE Sellers (

name VARCHAR2(20) NOT NULL, *-- Changed INT to VARCHAR2(20) for name*

seller\_id INT NOT NULL,

phone VARCHAR2(20) NOT NULL, *-- Changed INT to VARCHAR2(20) for phone numbers*

address VARCHAR2(200) NOT NULL, *-- Changed INT to VARCHAR2(200) for address*

PRIMARY KEY (seller\_id)

);

CREATE TABLE Categories (

category\_id INT NOT NULL,

category\_name VARCHAR2(200) NOT NULL, *-- Changed INT to VARCHAR2(200) for category name*

description VARCHAR2(2000), *-- Changed INT to VARCHAR2(2000) for description (optional)*

PRIMARY KEY (category\_id)

);

CREATE TABLE OrderDetails (

product\_id INT NOT NULL,

quantity INT NOT NULL,

delivery\_method VARCHAR2(20) NOT NULL,

PRIMARY KEY (product\_id)

);

CREATE TABLE Buyers (

buyer\_id INT NOT NULL,

name VARCHAR2(200) NOT NULL, *-- Changed INT to VARCHAR2(200) for name*

email VARCHAR2(255) NOT NULL, *-- Changed INT to VARCHAR2(255) for email*

phone VARCHAR2(20) NOT NULL, *-- Changed INT to VARCHAR2(20) for phone number*

address VARCHAR2(200) NOT NULL, *-- Changed INT to VARCHAR2(200) for address*

PRIMARY KEY (buyer\_id)

);

CREATE TABLE Products (

product\_id INT NOT NULL,

product\_name VARCHAR2(200) NOT NULL, *-- Changed INT to VARCHAR2(200) for product name*

description VARCHAR2(2000), *-- Changed INT to VARCHAR2(2000) for description (optional)*

price NUMBER(10,2) NOT NULL, *-- Use NUMBER for price with precision (10) and scale (2)*

category\_id INT NOT NULL,

PRIMARY KEY (product\_id),

FOREIGN KEY (category\_id) REFERENCES Categories(category\_id)

);

CREATE TABLE Orders (

order\_id INT NOT NULL,

order\_date DATE NOT NULL, *-- Use DATE for date*

tracking\_number VARCHAR2(20) NOT NULL,

product\_id INT NOT NULL,

buyer\_id INT NOT NULL,

PRIMARY KEY (order\_id, product\_id),

FOREIGN KEY (product\_id) REFERENCES OrderDetails(product\_id),

FOREIGN KEY (buyer\_id) REFERENCES Buyers(buyer\_id)

);

CREATE TABLE Reviews (

review\_id INT NOT NULL,

rating NUMBER(10,1), *-- Use NUMBER for rating with precision (10) and scale (1)*

comment\_text VARCHAR2(2000), *-- Changed INT to VARCHAR2(2000) for comment*

review\_date DATE NOT NULL, *-- Use DATE for date*

product\_id INT NOT NULL,

buyer\_id INT NOT NULL,

PRIMARY KEY (review\_id),

FOREIGN KEY (product\_id) REFERENCES Products(product\_id),

FOREIGN KEY (buyer\_id) REFERENCES Buyers(buyer\_id)

);

CREATE TABLE sell (

seller\_id INT NOT NULL,

product\_id INT NOT NULL,

PRIMARY KEY (seller\_id, product\_id),

FOREIGN KEY (seller\_id) REFERENCES Sellers(seller\_id),

FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

CREATE TABLE include\_products (

order\_id INT NOT NULL,

product\_id INT NOT NULL,

PRIMARY KEY (order\_id, product\_id),

FOREIGN KEY (order\_id, product\_id) REFERENCES Orders(order\_id, product\_id)

);

*-- dropTables.sql*

*-- Drop the tables in the reverse order of their creation to ensure all dependencies are handled properly*

DROP TABLE include\_products CASCADE CONSTRAINTS;

DROP TABLE sell CASCADE CONSTRAINTS;

DROP TABLE Reviews CASCADE CONSTRAINTS;

DROP TABLE Orders CASCADE CONSTRAINTS;

DROP TABLE Products CASCADE CONSTRAINTS;

DROP TABLE Buyers CASCADE CONSTRAINTS;

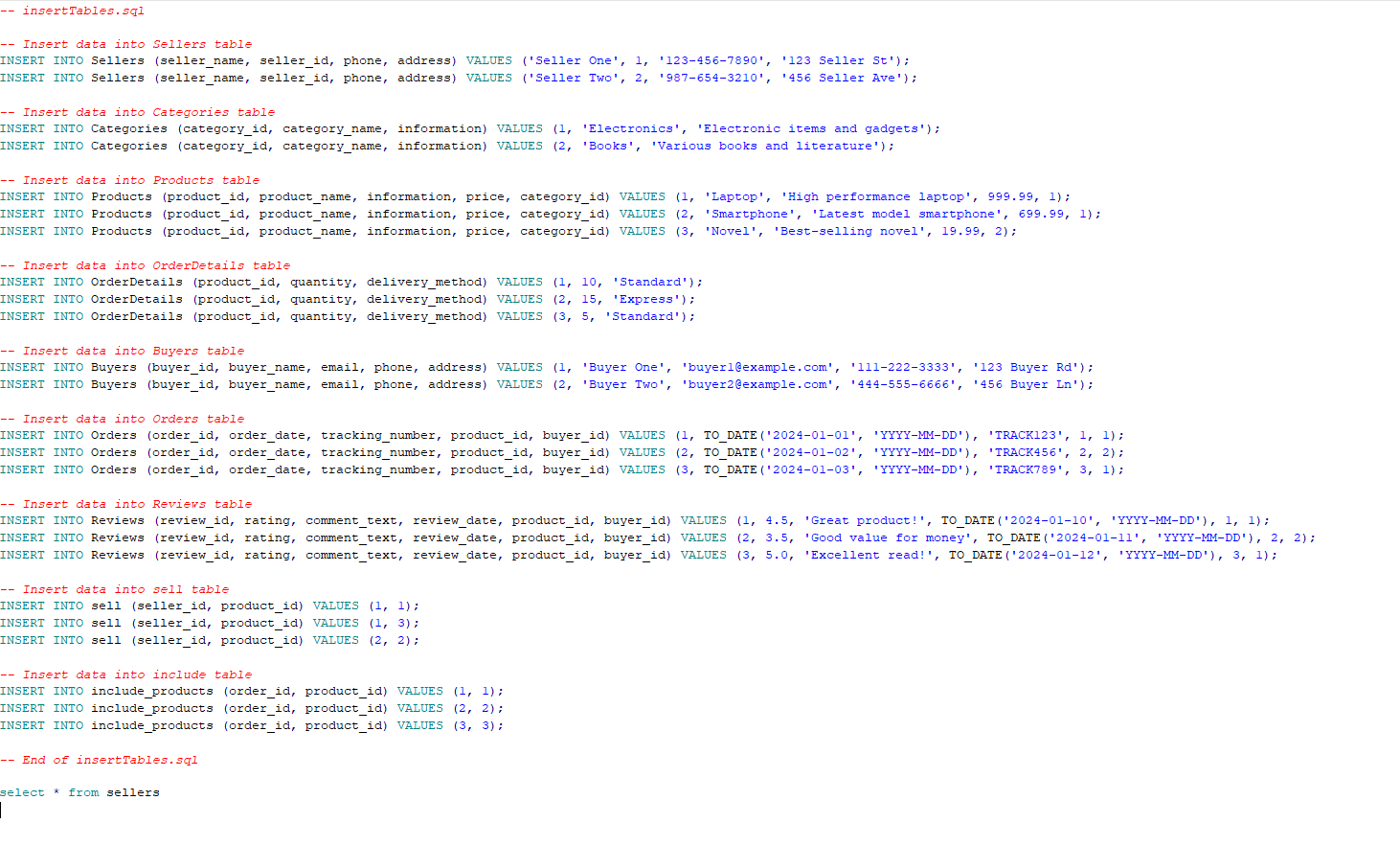
DROP TABLE OrderDetails CASCADE CONSTRAINTS;

DROP TABLE Categories CASCADE CONSTRAINTS;

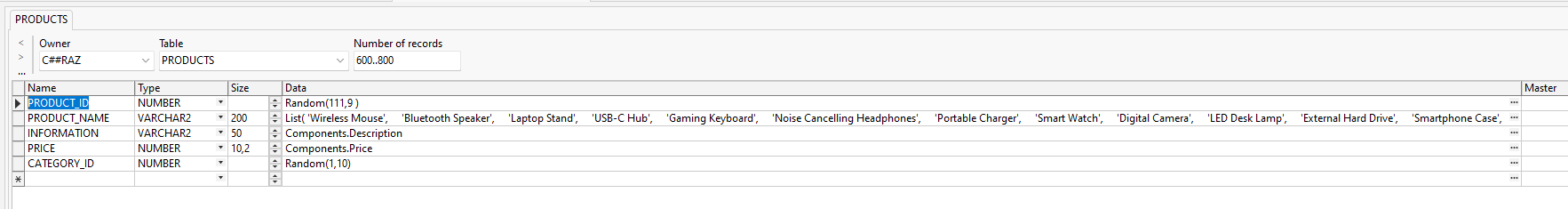
DROP TABLE Sellers CASCADE CONSTRAINTS;

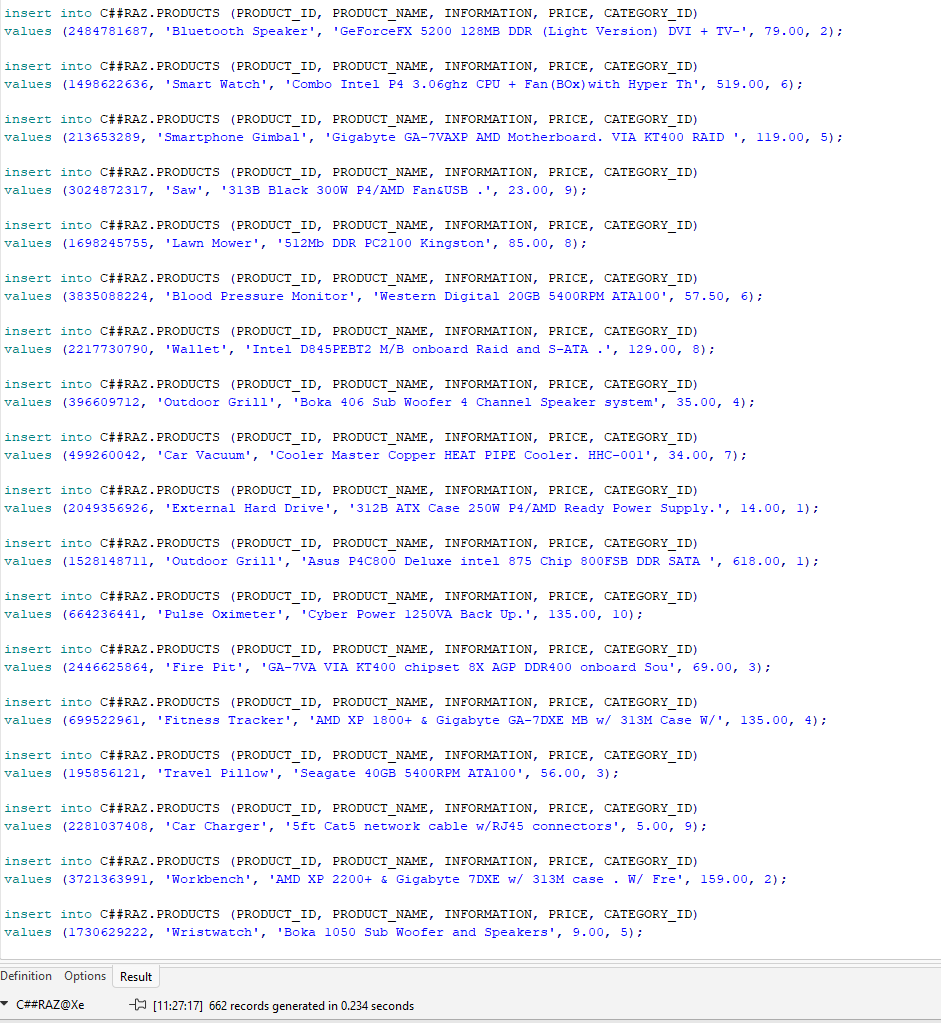
*-- End of dropTables.sql*

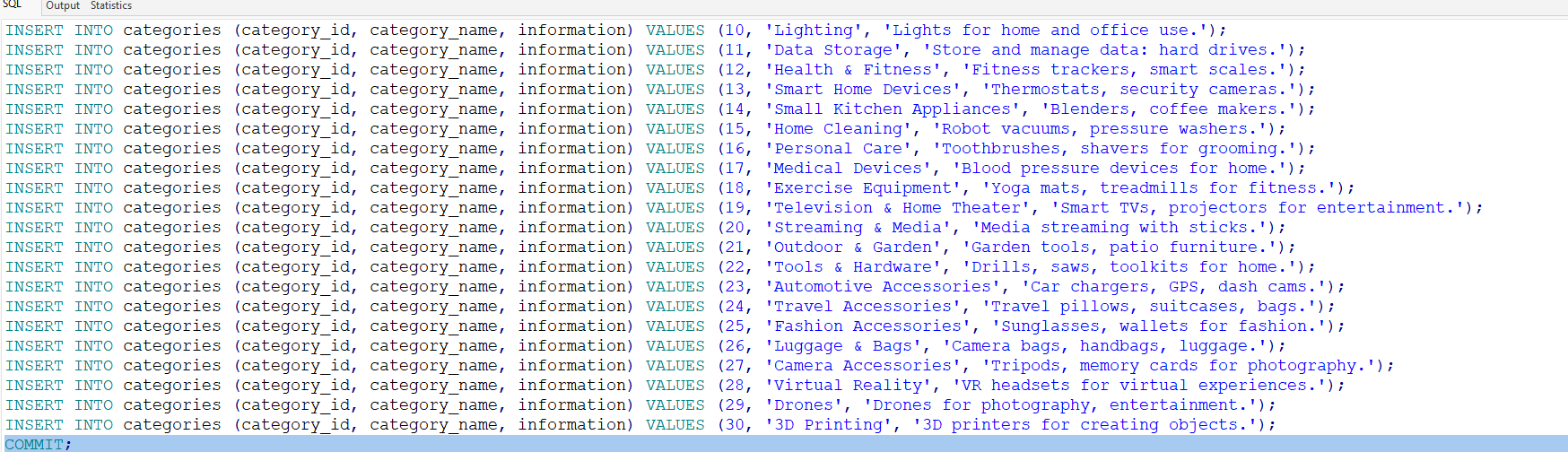
insert



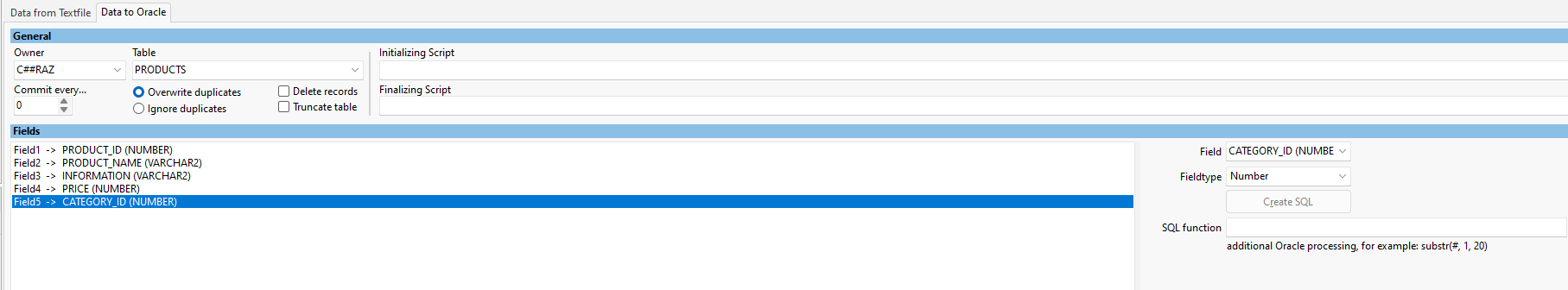
generate

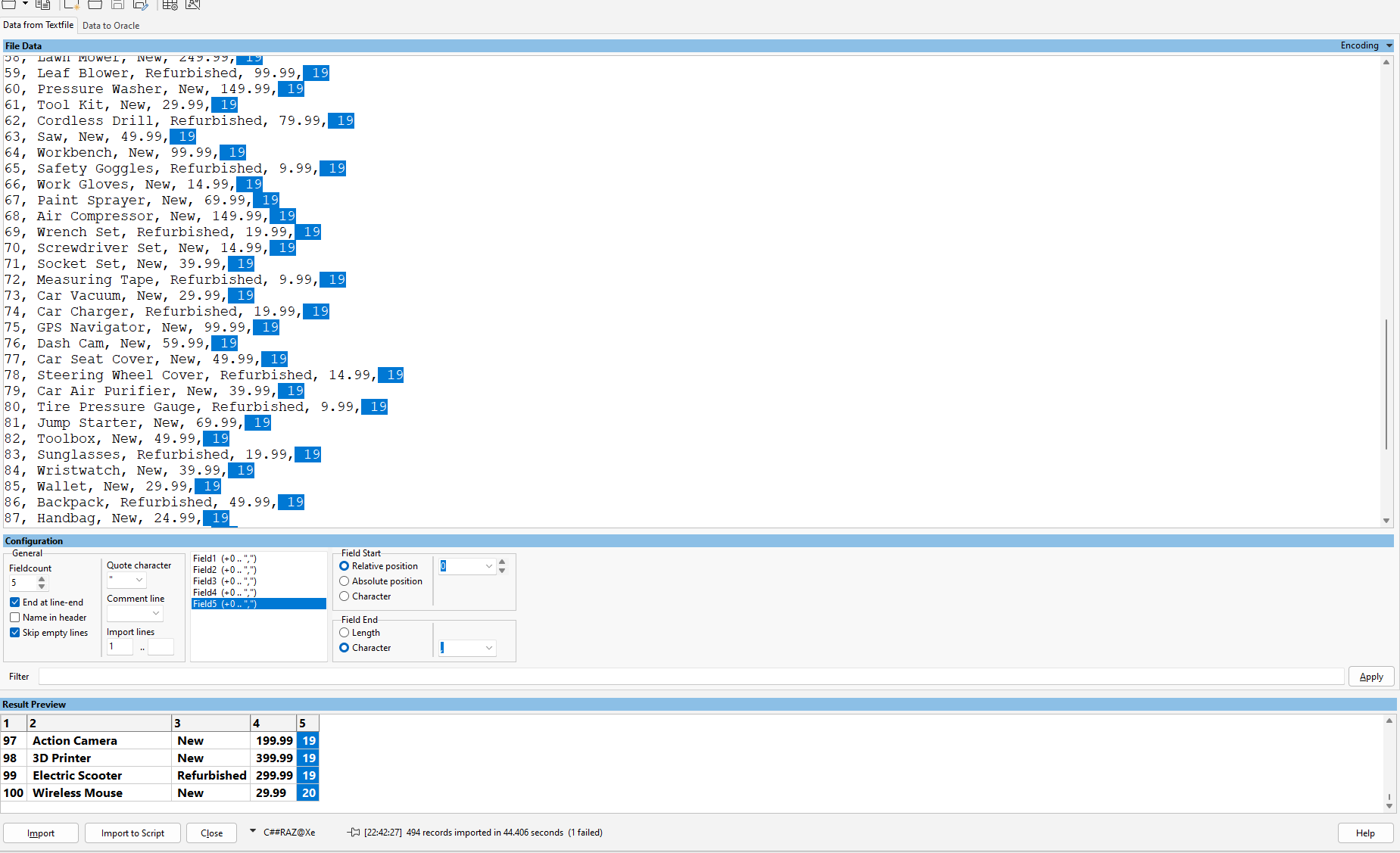




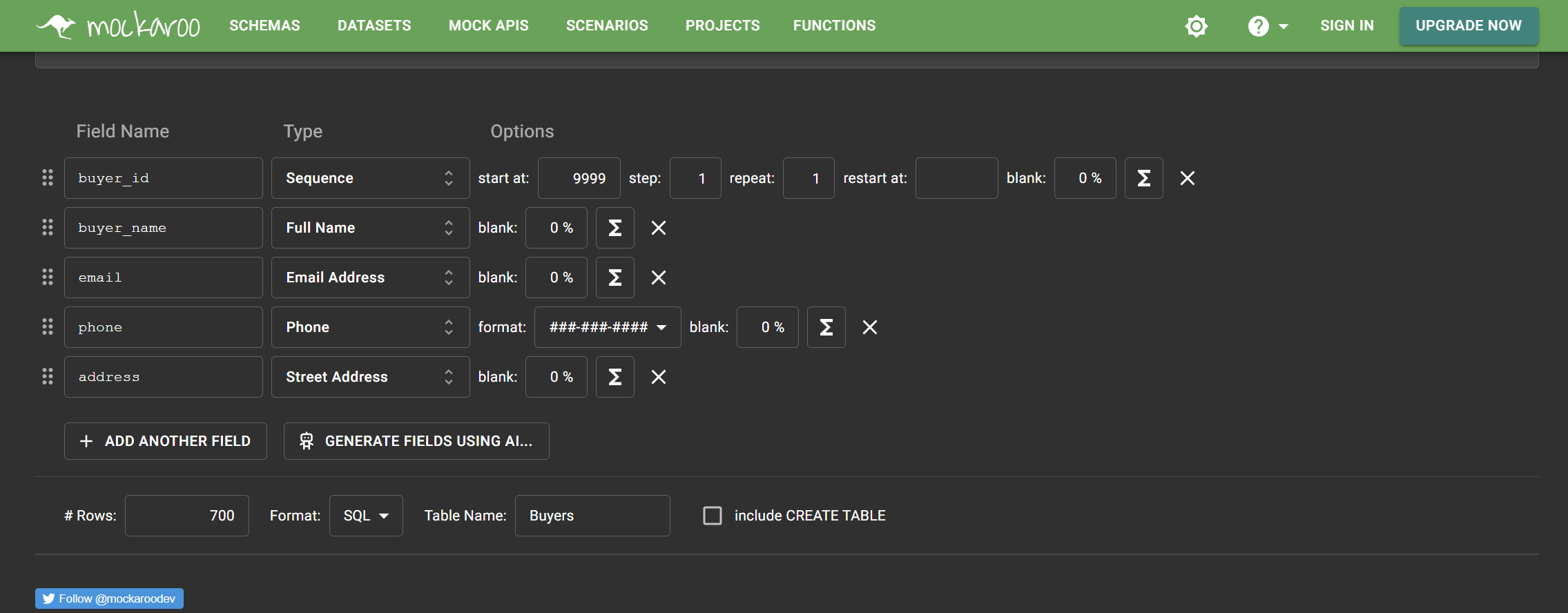


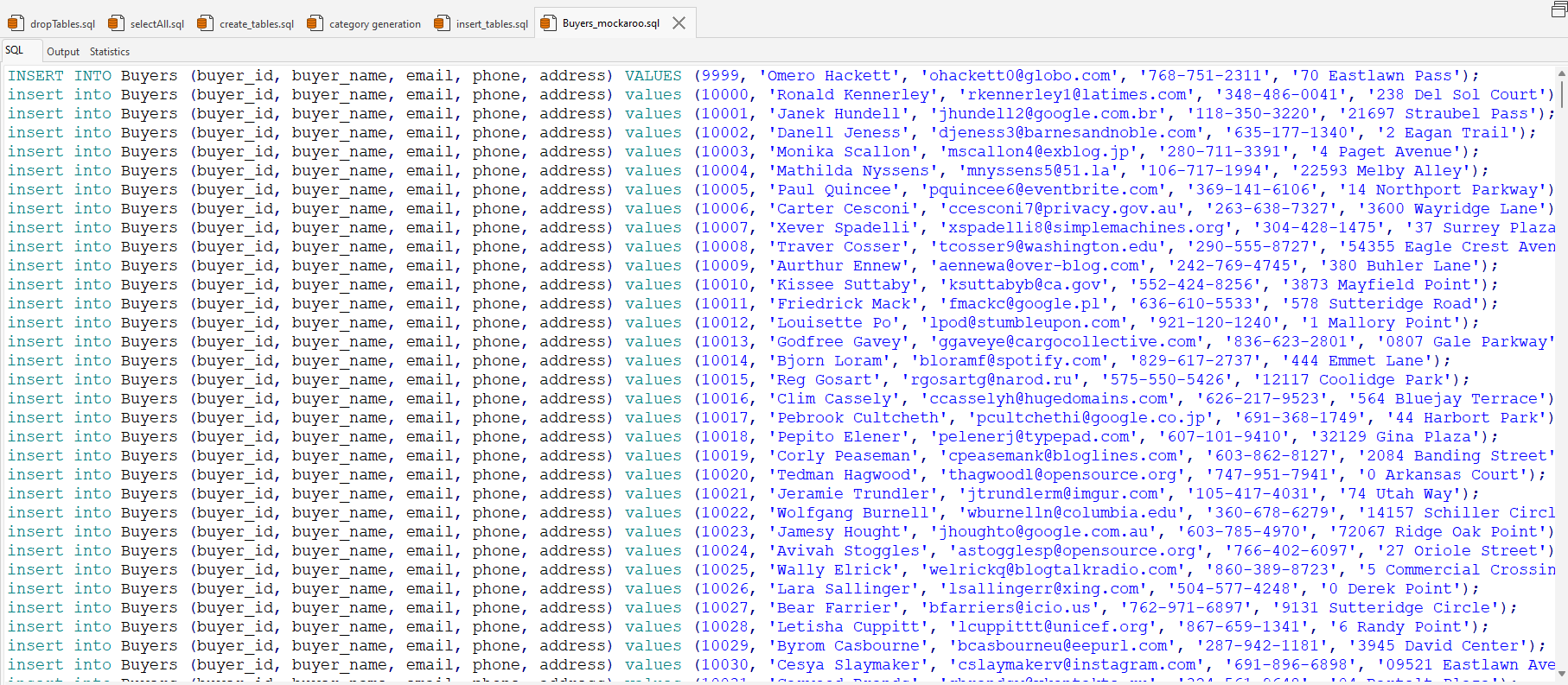
txt

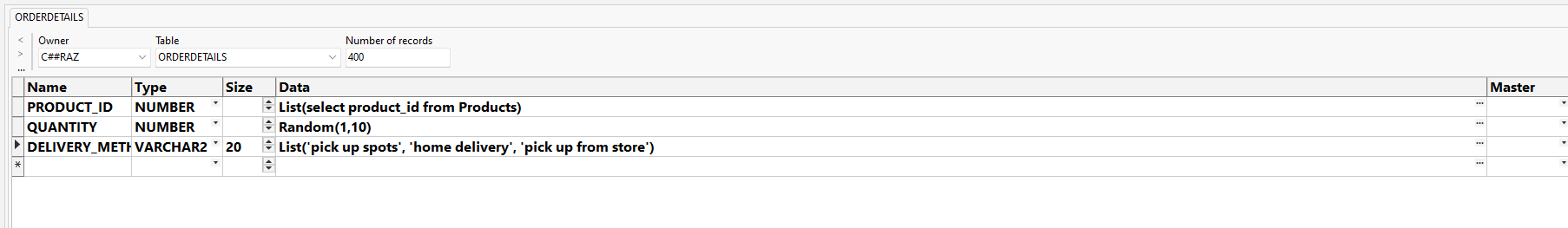


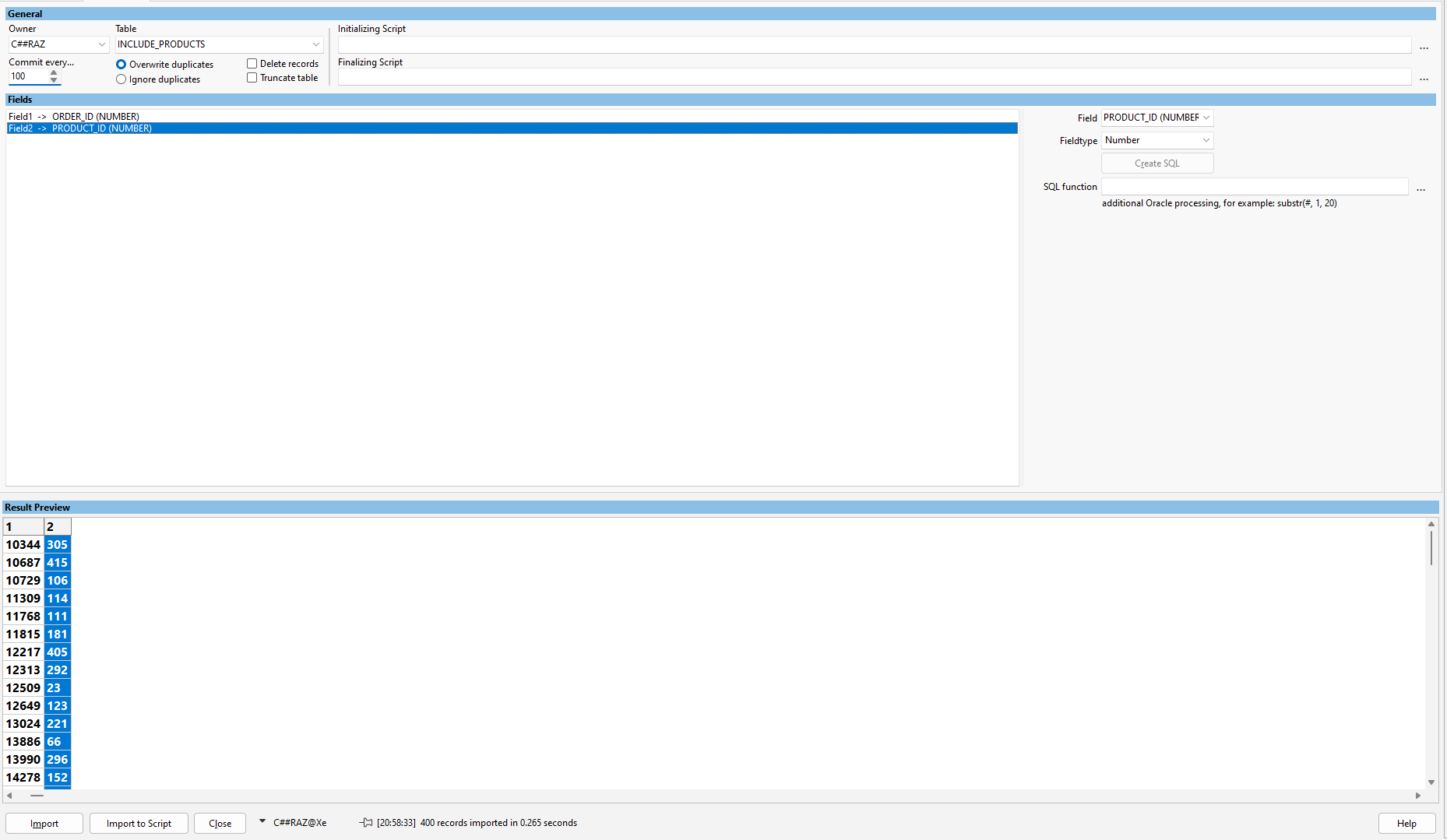


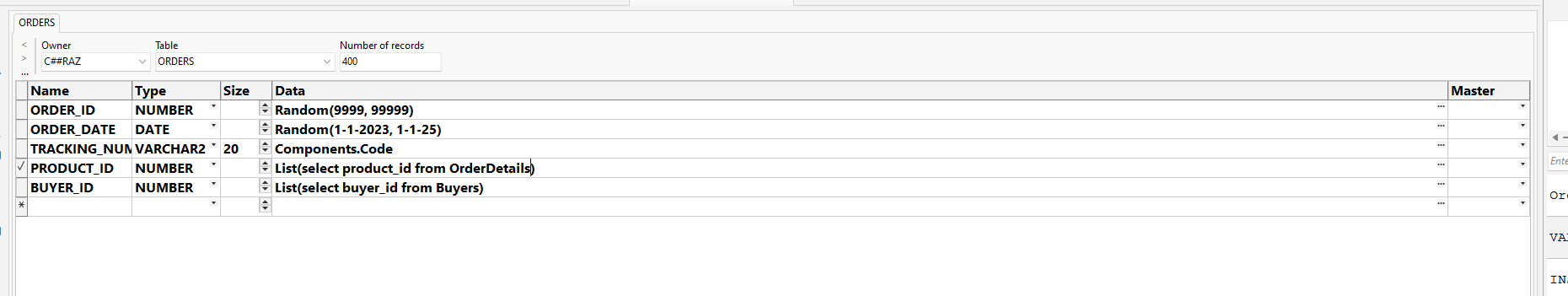
mockaroo



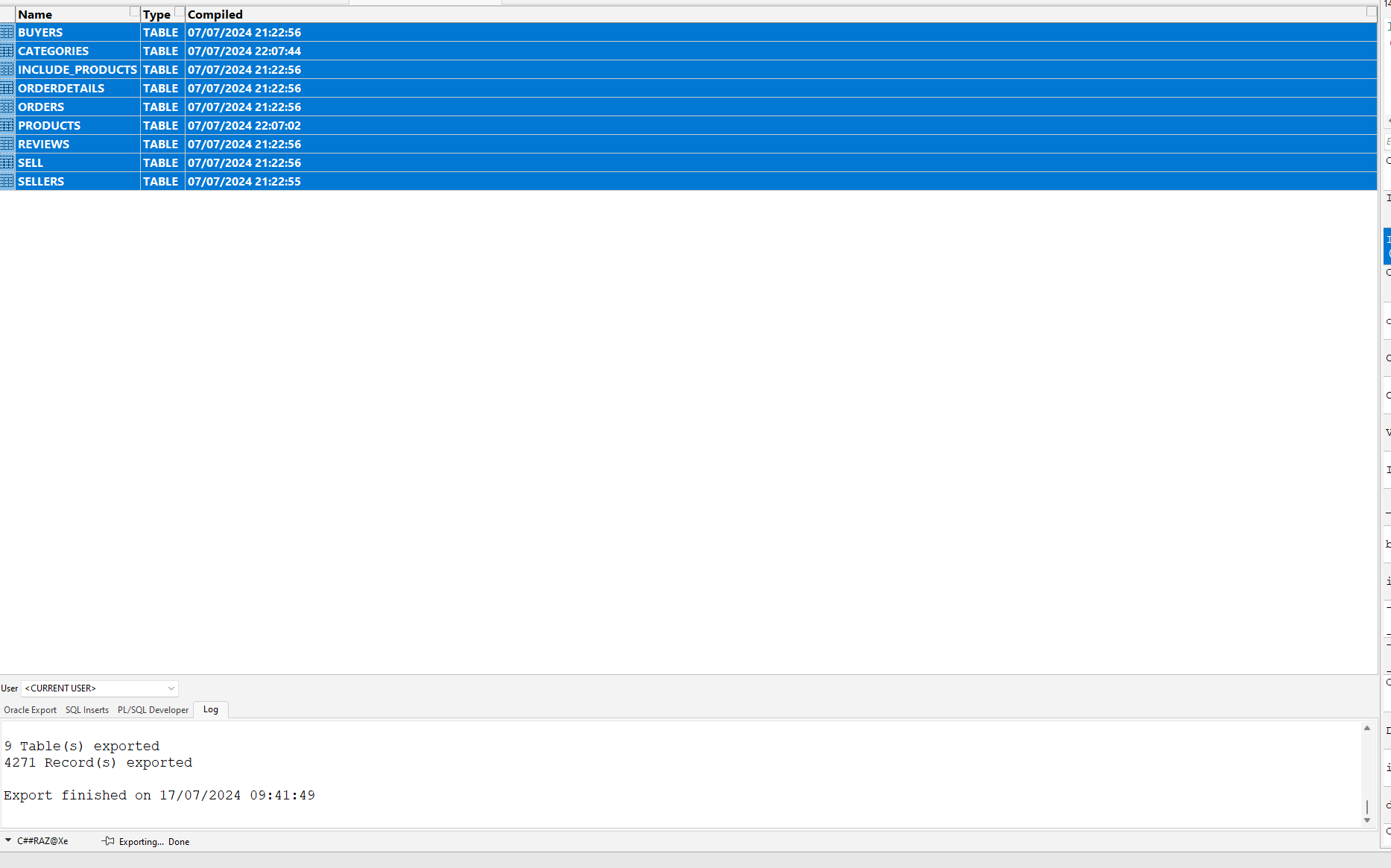








יצירת BACKUP

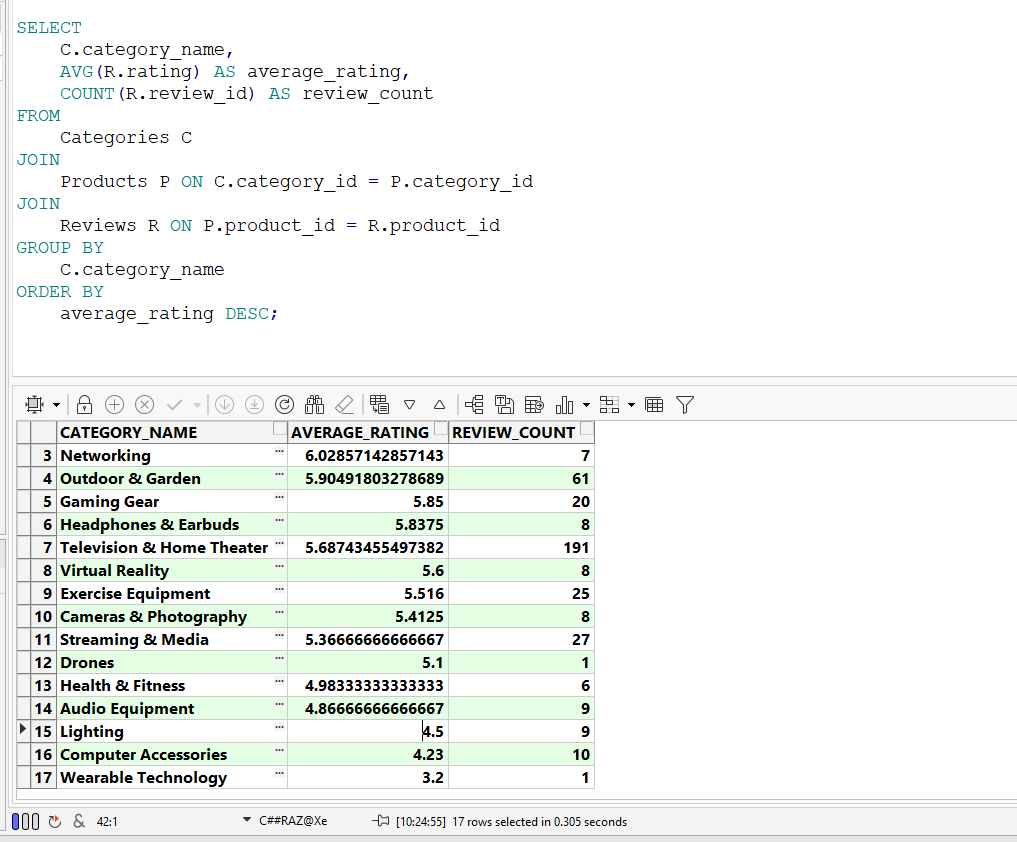


חלק 2 שאילתות

שאילתה ראשונה

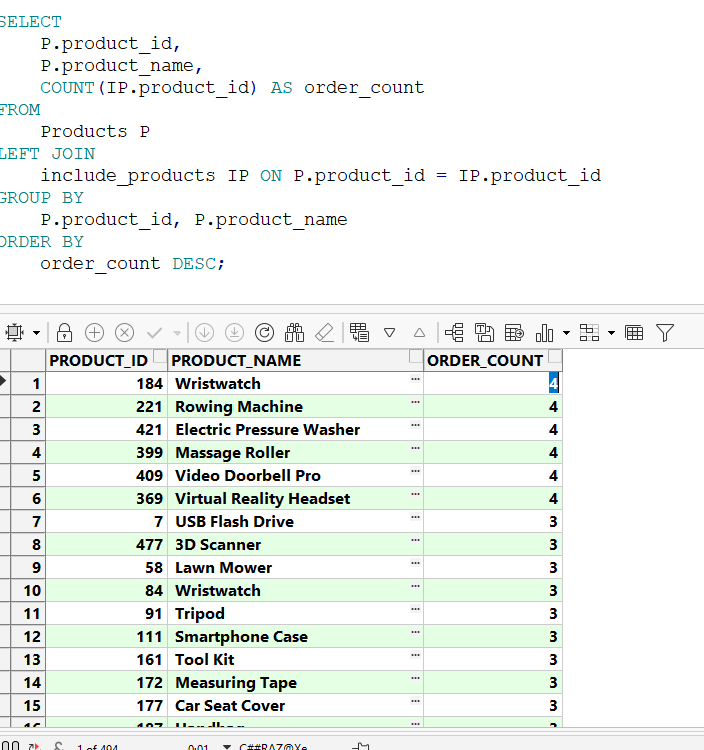
מחזיר ממוצע של דירוגים של מוצרים תחת קטגוריה מסויימת

וכן את מספר הביקורת שנעשו עבור על קטגוריה

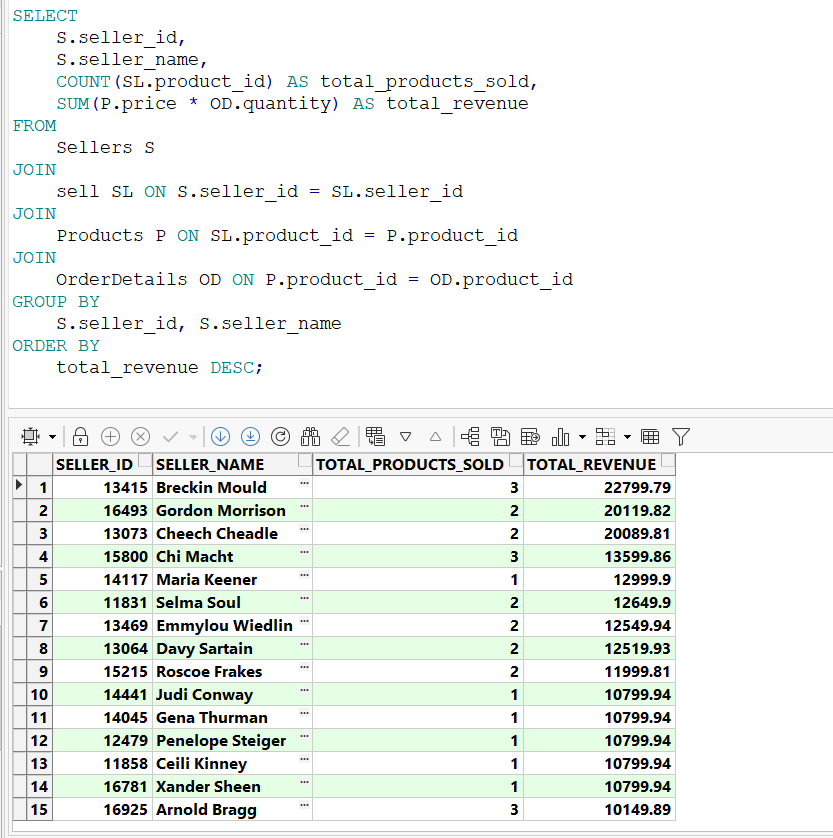


שאילתה שנייה

מחזירה רשימה של מוצרים לפי הנמכרים ביותר ואלו שעדיין לא נמכרו

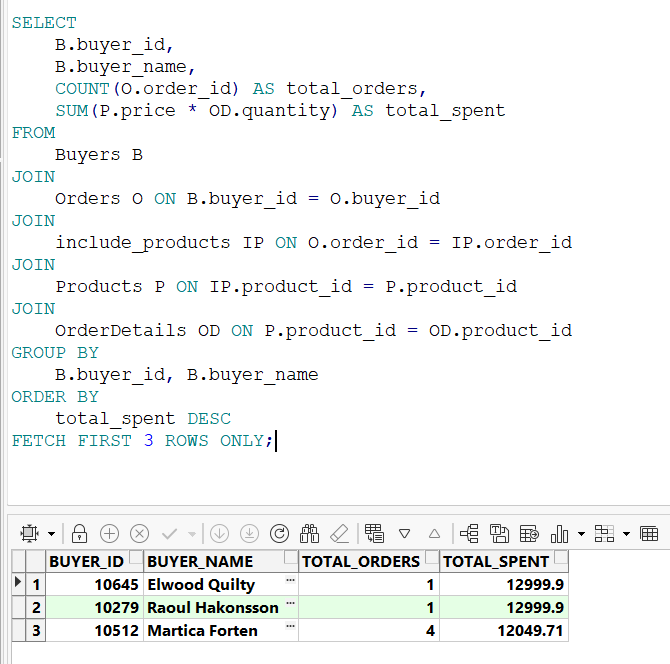


**שאילתה שלישית כמות מוצרים שנמכרו לפי מוכר וסך הרווחים שלו**

****

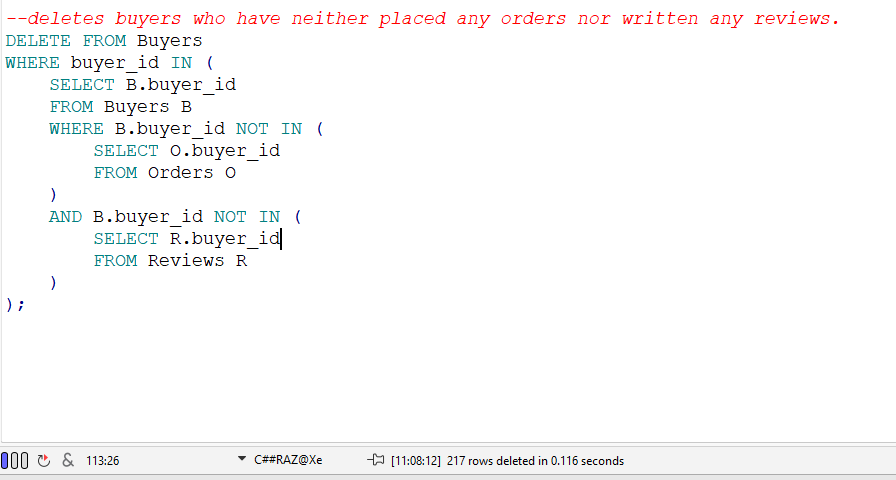
**שאילתה רביעית**

**שלושת הקונים שקנו הכי הרבה מוצרים וסכום הכסף שהוציאו**

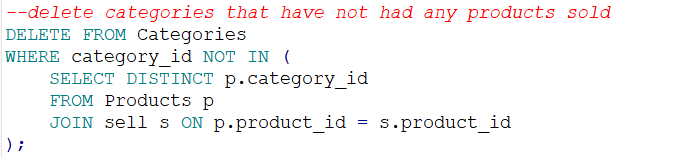
****

**שאילתות DELETE**

**מוחק קונים שלא ביצעו הזמנות או כתוב ביקורות**

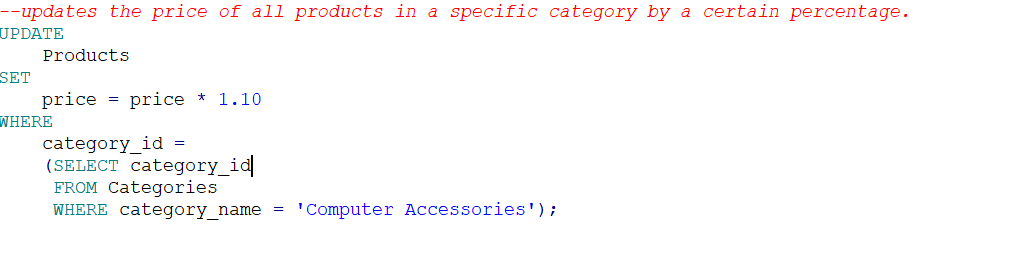
****

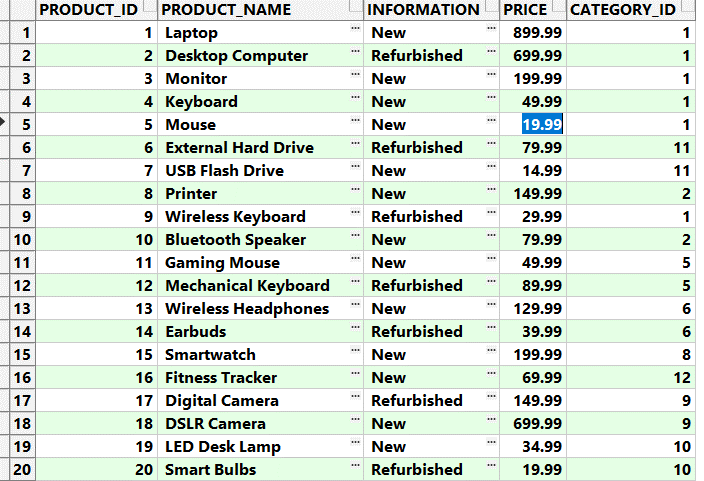
**מוחק קטגוריות שלא מוכרו מוצרים**

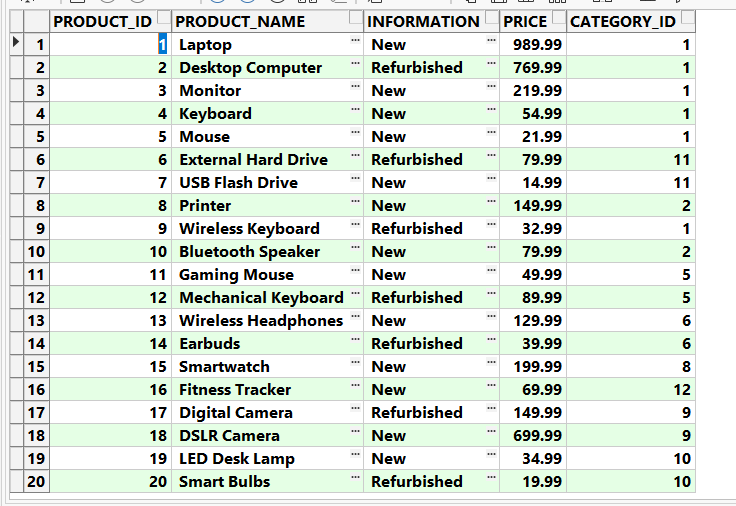
****

**שאילתות עידכון**

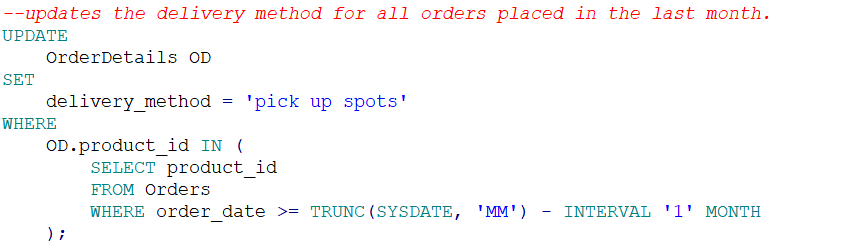
**לעדכן מחירים של קטגוריה מסויימת**

****



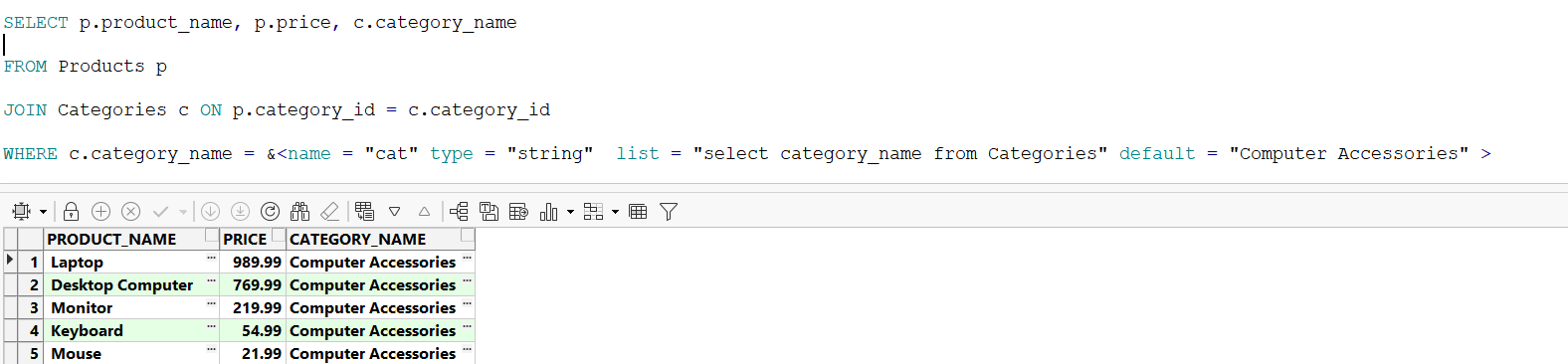


**שאילתה זו מעדכנת שיטת משלוח עבור משלוחים שנרשמו בחודש האחרון**

****

**שאילתות עם פרמטרים**

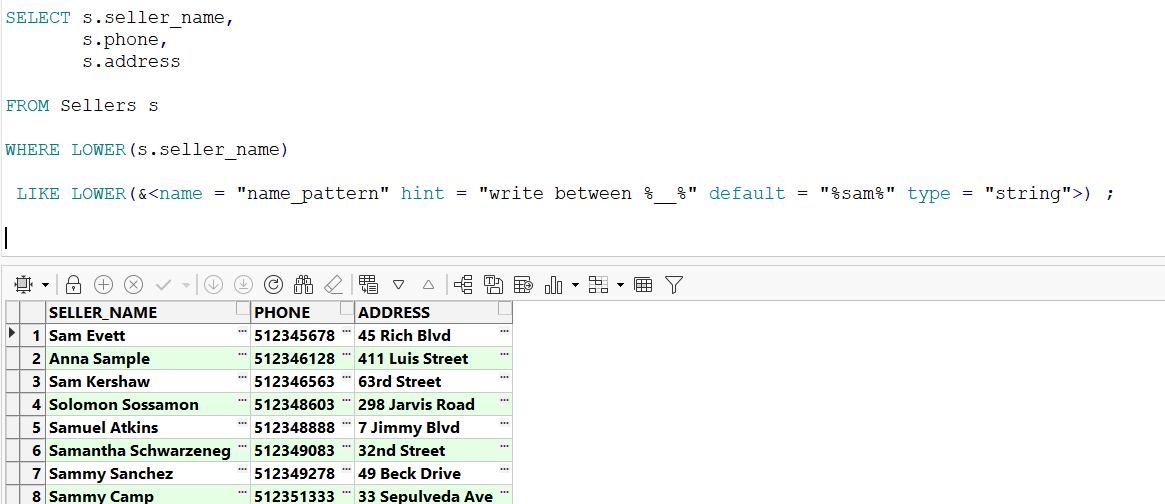
1. **שאילתה מחזירה מוצרים לפי קטגוריה מסוימת עם list של שמות על הקטגוריות**

****

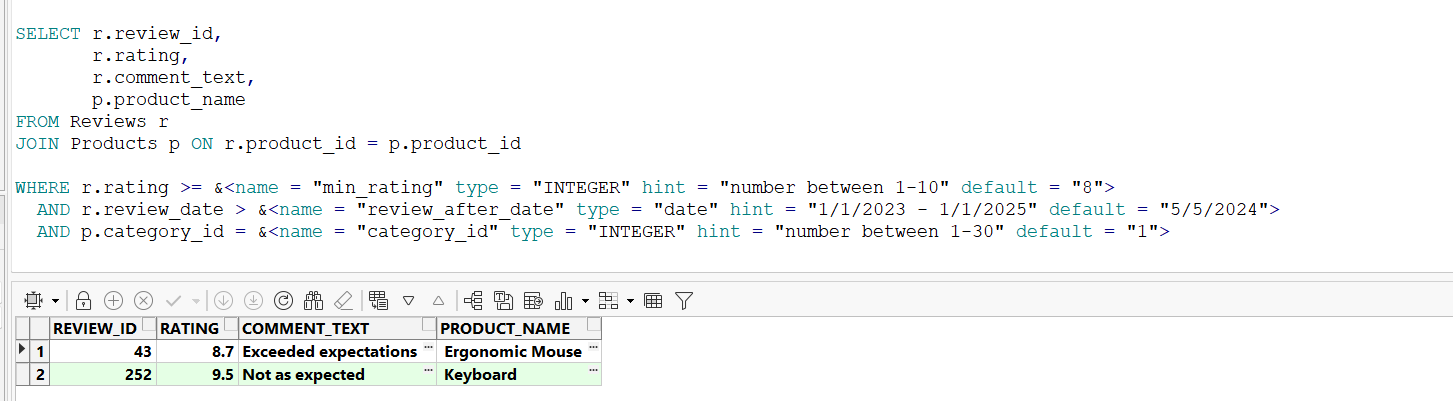
**2.מחזיר את כל ההזמנות שבוצעו בטווח תאריכים מסוים**

****

**3.חיפוש מוכר לפי תבנית מסוימת , שתתקבל תהיה של כל מוכר שעומד בתבנית**

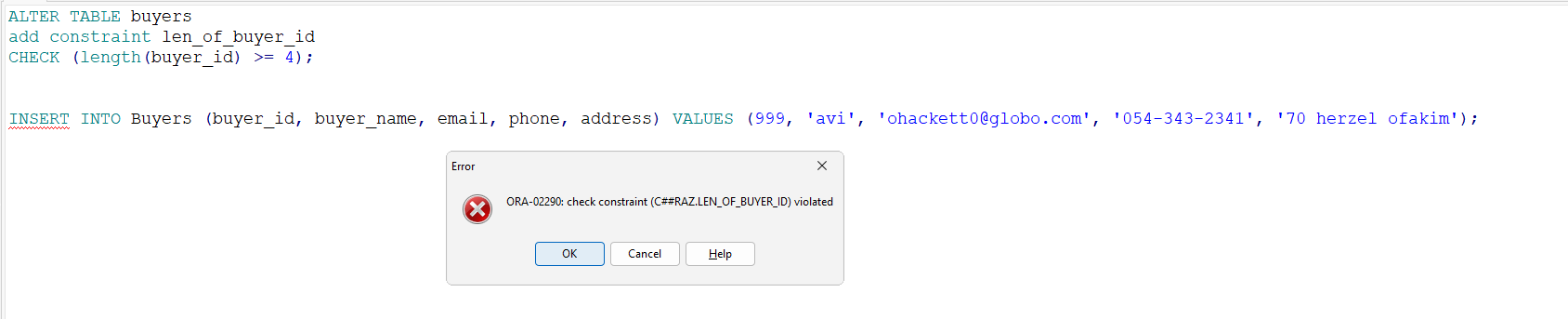
****

**4.שאילתה שמחזירה ביקורות לפי קטגוריה מסוימת החל מדירוג מסוים והחל מתאריך מסוים**

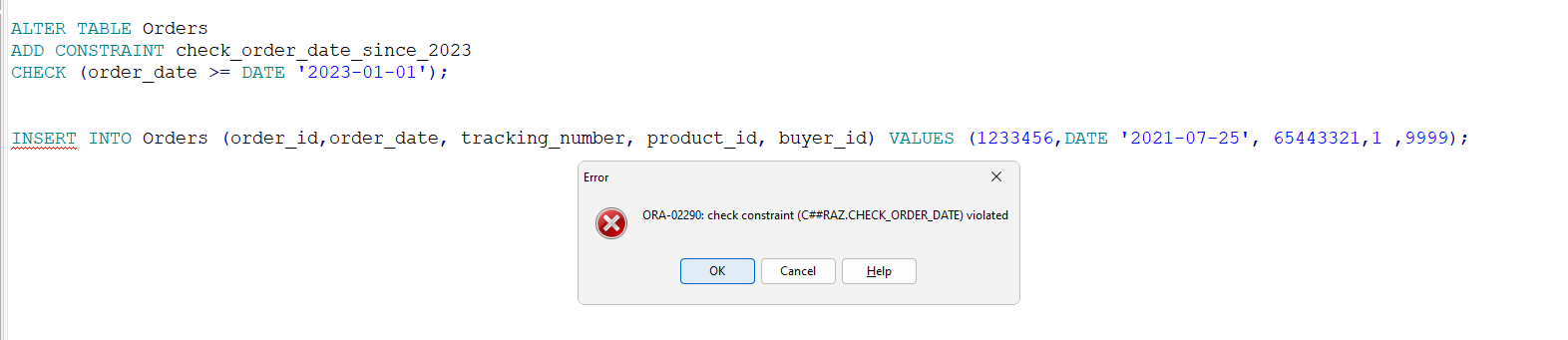
****

**אילוצים**

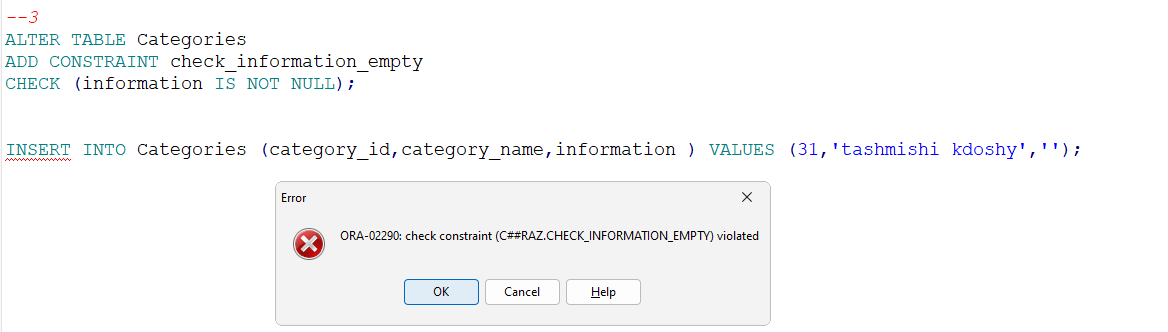
אילוץ ראשון הגבלת אורך העמודה של buyer\_id בטבלה buyers למינימום של ארבע ספרות



אילוץ שני – אין אפשרות להוסיף הזמנה עם תאריך הזמנה שקדם לתאריך הקמת החנות



אילוץ שלישי – אין אפשרות להוסיף קטגוריה חדשה של מוצרים ללא תיאור הקטגוריה information



חלק שלישי

תכנות

פרוצדורה ראשונה

הפרוצדורה תרוץ על רשימת עם הקונים עם CURSER החל מתאריך מסויים שינתן כפרמטר

לכל קונה שרכש מוצר יותר מעשרה פעמים יקבל הנחה של עשרה אחוזים

לאחר מכן התוכנית תדפיס את שמות הקונים ואת המחיר שישלמו לפני ההנחה אם ניתנה להם ואחרי ההנחה

*--checks if a buyer has ordered more than 10 items, applies a 10% discount,*

*-- and then prints the buyer's name along with the original and discounted prices*

*-- exlicit cursor , loop ,excepetion , conditions*

CREATE OR REPLACE PROCEDURE ApplyDiscountAndPrint (

p\_order\_date IN DATE

) IS

CURSOR order\_cursor IS

SELECT b.buyer\_id, b.buyer\_name, o.order\_id, p.product\_id, ip.quantity, p.price

FROM Orders o

JOIN include\_products ip ON o.order\_id = ip.order\_id

JOIN Buyers b ON o.buyer\_id = b.buyer\_id

JOIN Products p ON o.product\_id = p.product\_id

WHERE o.order\_date >= p\_order\_date;

v\_buyer\_name Buyers.buyer\_name%TYPE;

v\_buyer\_id Buyers.buyer\_id%TYPE;

v\_order\_id Orders.order\_id%TYPE;

v\_product\_id Products.product\_id%TYPE;

v\_quantity INT;

v\_price NUMBER(10, 2);

v\_total\_price NUMBER(10, 2);

v\_discounted\_price NUMBER(10, 2);

v\_discount\_rate NUMBER := 0.1; *-- 10% discount*

BEGIN

IF p\_order\_date < TO\_DATE('01/01/2023', 'DD/MM/YYYY') THEN

RAISE\_APPLICATION\_ERROR(-20001, 'invalid date ' || TO\_CHAR(p\_order\_date, 'DD/MM/YYYY'));

END IF;

OPEN order\_cursor;

LOOP

FETCH order\_cursor INTO v\_buyer\_id, v\_buyer\_name, v\_order\_id, v\_product\_id, v\_quantity, v\_price;

EXIT WHEN order\_cursor%NOTFOUND;

*-- Calculate total price for the order*

v\_total\_price := v\_quantity \* v\_price;

*-- Apply discount if quantity is greater than 10*

IF v\_quantity > 10 THEN

v\_discounted\_price := v\_total\_price \* (1 - v\_discount\_rate);

ELSE

v\_discounted\_price := v\_total\_price; *-- No discount*

END IF;

*-- Print buyer's name and price details*

DBMS\_OUTPUT.PUT\_LINE('Buyer: ' || v\_buyer\_name);

DBMS\_OUTPUT.PUT\_LINE('Original Price: ' || v\_total\_price);

DBMS\_OUTPUT.PUT\_LINE('Discounted Price: ' || v\_discounted\_price);

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

END LOOP;

CLOSE order\_cursor;

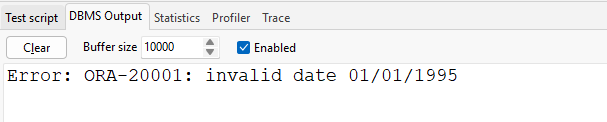
EXCEPTION

WHEN OTHERS THEN

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

END ApplyDiscountAndPrint;

אם הכנסנו כפרמטר תאריך שהוא לא תקין נקבל חריגה



פונקציה ראשונה

הפונקציה תקבל דירוג כפרמטר וטווח תאריכים ותחזיר את רשימת המוכרים שקיבלו על המוצרים שלהם ביקורות עם דירוג נמוך מתחת לדירוג שניתן כפרטמר בטווח תאריכים מסוים

*--function that receives a rating as a parameter and returns a list of sellers*

*-- who have products with ratings below the given rating between the range of the param dates*

*-- refcursor ,exception*

CREATE OR REPLACE FUNCTION GetSellersWithLowRatedProducts(

p\_rating IN NUMBER,

p\_start\_date IN DATE,

p\_end\_date IN DATE

) RETURN SYS\_REFCURSOR IS

v\_sellers SYS\_REFCURSOR;

BEGIN

IF p\_rating < 1 OR p\_rating > 5 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Invalid rating. Rating must be between 1 and 5.');

ELSIF p\_start\_date > p\_end\_date THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Invalid date range. Start date must be before end date.');

END IF;

OPEN v\_sellers FOR

SELECT DISTINCT s.seller\_id, se.seller\_name, AVG(r.rating) OVER (PARTITION BY p.product\_id) AS avg\_rating

FROM Reviews r

JOIN Products p ON r.product\_id = p.product\_id

JOIN sell s ON p.product\_id = s.product\_id

JOIN sellers se ON s.seller\_id = se.seller\_id

WHERE r.rating < p\_rating

AND r.review\_date BETWEEN p\_start\_date AND p\_end\_date;

RETURN v\_sellers;

EXCEPTION

WHEN OTHERS THEN

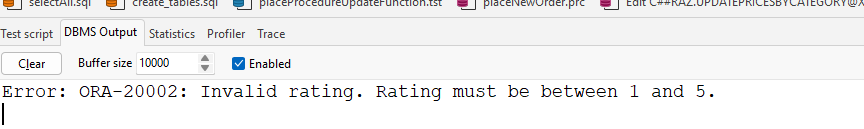
DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

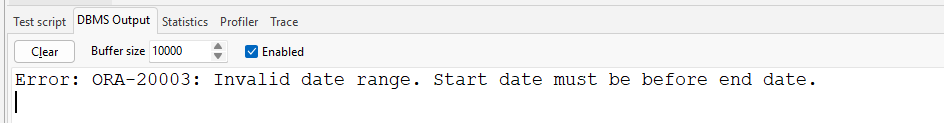
RETURN NULL;

END GetSellersWithLowRatedProducts;

אם המשתמש ישלח בתור פרמטר של דירוג מספר קטן מ1 או גדול מ5 הוא יקבל חריגה וכן תאריכים מוקדם לפני מאוחר

דוגמה





תוכנית ראשית שתקרא לפרוצדורה הראשונה ApplyDiscountAndPrint ולפונקציה הראשונה

GetSellersWithLowRatedProducts

התוכנית תשתמש בREFCURSER כדי להדפיס את כל שמות הקונים עם ביקורות נמוכות

שחזרו מהפונקציה

DECLARE

v\_sellers SYS\_REFCURSOR; *-- Cursor for fetching sellers with low-rated products*

v\_seller\_id INT;

v\_seller\_name VARCHAR2(50);

v\_avg\_rating NUMBER(10, 2);

BEGIN

*-- Enable DBMS\_OUTPUT with a buffer size of 1,000,000 bytes*

DBMS\_OUTPUT.ENABLE(1000000);

ApplyDiscountAndPrint(p\_order\_date => :p\_order\_date);

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

*-- Call the GetSellersWithLowRatedProducts function to get sellers with products rated below 3*

*-- with reviews within the date range from '2023-01-01' to '2025-01-01'*

v\_sellers := GetSellersWithLowRatedProducts(3, TO\_DATE('2023-01-01', 'YYYY-MM-DD'), TO\_DATE('2025-01-01', 'YYYY-MM-DD'));

LOOP

FETCH v\_sellers INTO v\_seller\_id, v\_seller\_name, v\_avg\_rating;

EXIT WHEN v\_sellers%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Seller ID: ' || v\_seller\_id || ', Name: ' || v\_seller\_name || ', Average Rating: ' || v\_avg\_rating);

END LOOP;

CLOSE v\_sellers;

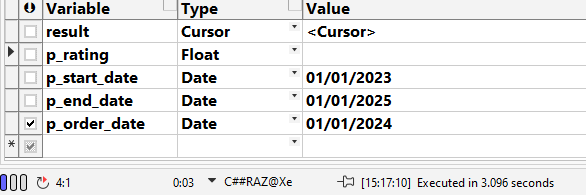
EXCEPTION

WHEN OTHERS THEN

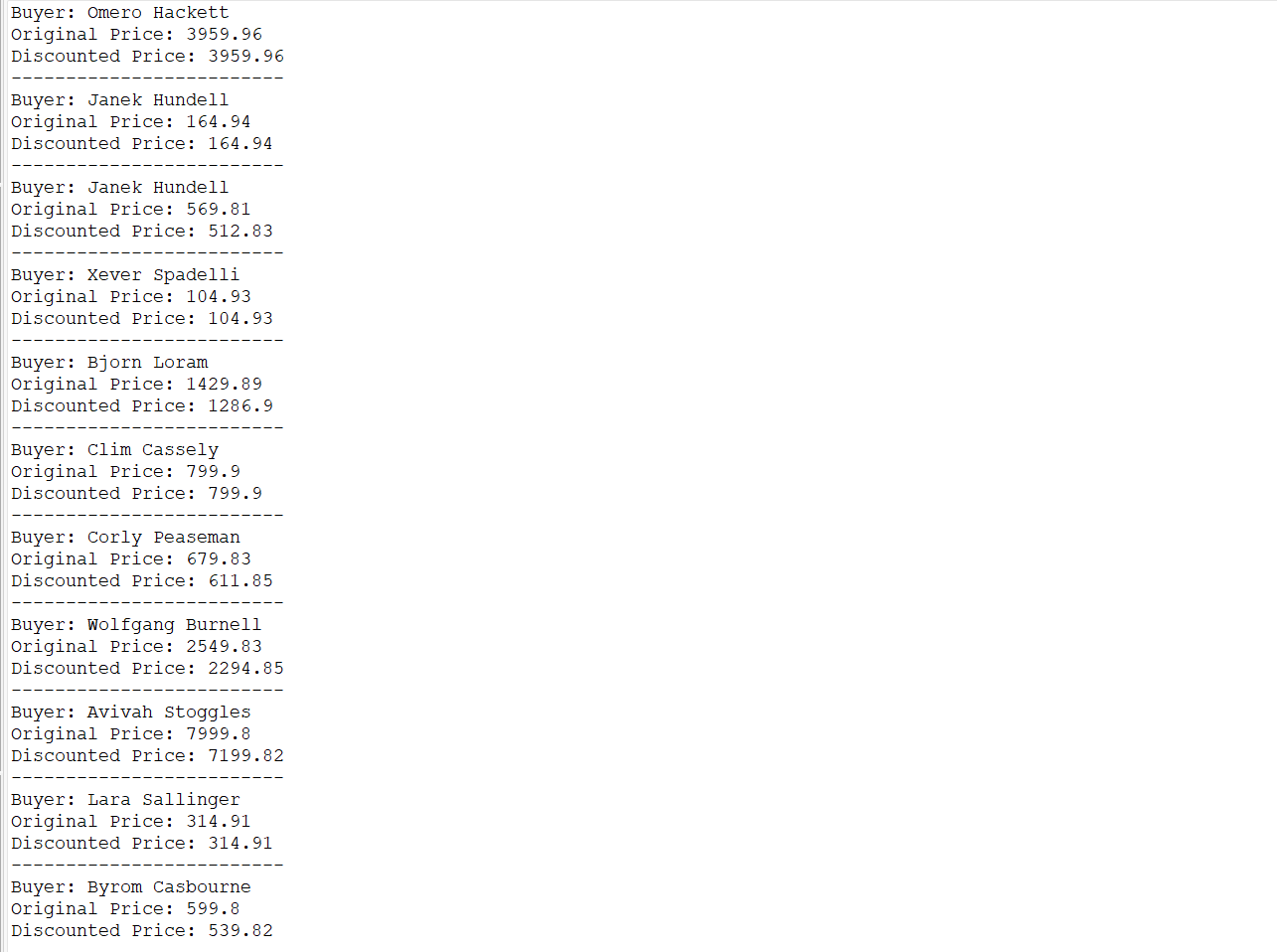
DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

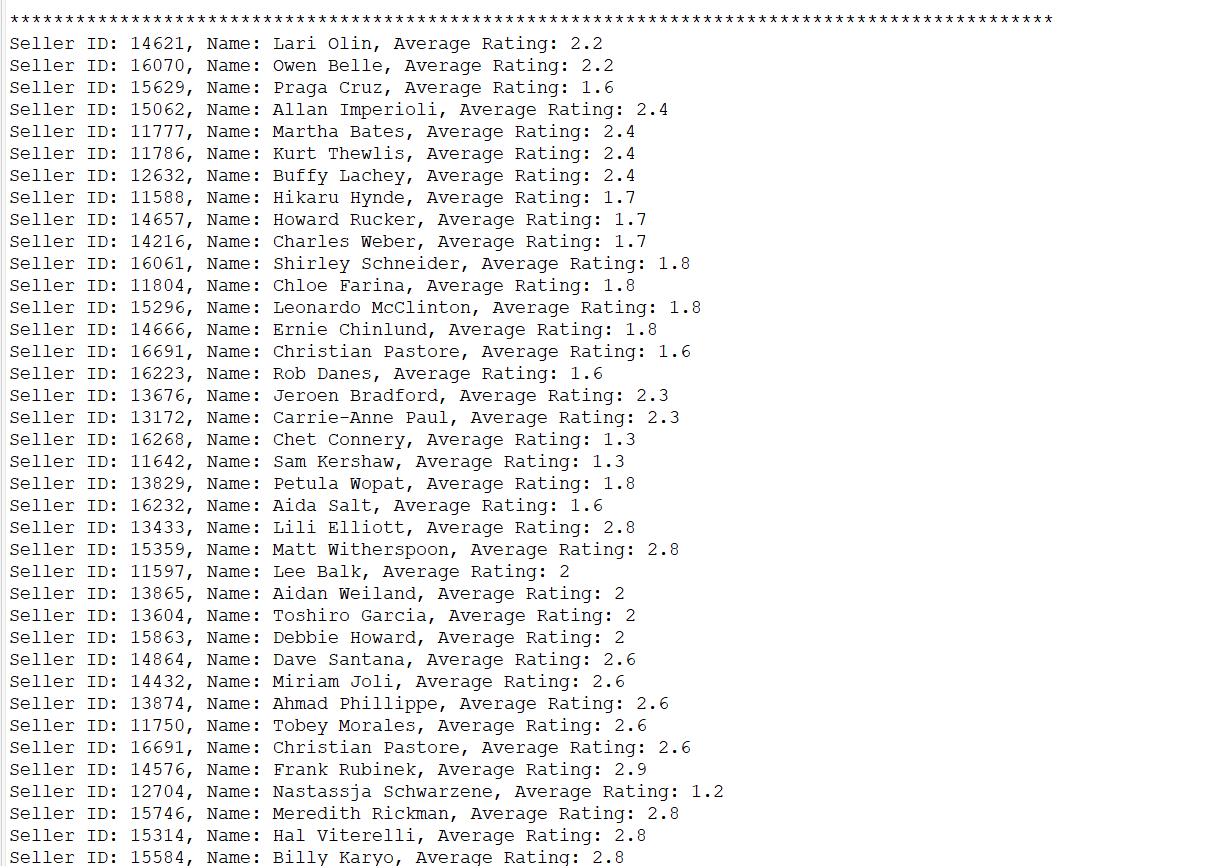
דוגמאות ריצה



חלק מרשימת הקונים שחלקם קיבלו הנחה



וחלק מרשימת המוכרים שקיבלו דירוג נמוך מ3



פרוצדורה שנייה

מבצעת הזמנה חדשה אם המלאי של המוצר לא תואם את ההזמנה חריגה תופיע אחרת תתוסף הזמנה חדשה המלאי של המוצר יתעדכן ואם לא תהיה שום בעיה יתבצע COMMIT למסד הנתונים

אחרת נבצע rollback למצב הקודם של בסיס הנתונים שנשמר בsavepoint

*--place new order,if stock is insufficent exception will raise*

*--otherwise the new order will placed and the stock will update and then if no other error will raise will commit the update*

CREATE OR REPLACE PROCEDURE PlaceNewOrder (

buyerId IN INT,

orderId IN INT,

orderDate IN DATE,

trackingNumber IN VARCHAR2,

product IN INT,

quantity IN INT

) IS

v\_stock INT;

BEGIN

*-- Start transaction*

SAVEPOINT sp\_before\_order; *-- save for rollback the action*

*-- Check stock*

SELECT p.stock

INTO v\_stock

FROM Products p

WHERE p.product\_id = product;

IF v\_stock < quantity THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient stock for product ' || product);

END IF;

*-- Insert into Orders table*

INSERT INTO Orders (order\_id, order\_date, tracking\_number, product\_id, buyer\_id)

VALUES (orderId, orderDate, trackingNumber, product, buyerId);

*-- Update stoc*

UPDATE Products

SET stock = stock - quantity

WHERE product\_id = product;

*-- Insert into include\_products table*

INSERT INTO include\_products (order\_id, product\_id,Quantity)

VALUES (orderId, product,quantity);

*-- Commit transaction*

COMMIT;

EXCEPTION

WHEN OTHERS THEN

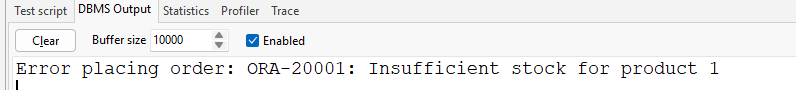
*-- Rollback to savepoint in case of error*

ROLLBACK TO sp\_before\_order;

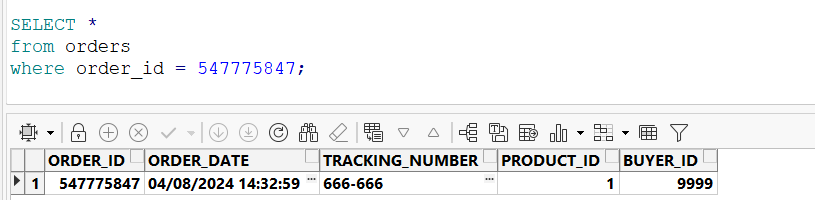
RAISE;

END PlaceNewOrder;

אם כמות המוצרים בהזמנה תעלה על כמות המלאי של המוצר תעלה חריגה



דוגמת ריצה



פונקציה שנייה

בעקבות עליית מחירים בשוק נצטרך להעלות את המחיר של מוצרים ממחלקה מסוימת שנכנסו לחנות החל מתאריך מסויים שינתן לפוקנציה ביחד עם הקטגוריה ,הפונקציה תעלה את המחיר של כל המוצרים בקטגוריה בחמישה אחוזים ותדווח לנו את כמות המוצרים שעודכנו

*-- the fucntion will update the prices of all the products from the given category*

CREATE OR REPLACE FUNCTION UpdatePricesByCategory (

startDate IN DATE,

categoryId IN INT

) RETURN VARCHAR2 IS

CURSOR productCursor IS

SELECT product\_id, price

FROM Products

WHERE category\_id = categoryId AND available\_date >= startDate ;

productRecord productCursor%ROWTYPE;

updatedCount INT := 0;

BEGIN

*-- Open the cursor*

OPEN productCursor;

*-- Loop through each product*

LOOP

FETCH productCursor INTO productRecord;

EXIT WHEN productCursor%NOTFOUND;

*-- Update the product price by 5%*

UPDATE Products

SET price = price \* 1.05

WHERE product\_id = productRecord.product\_id;

*-- Increment the count of updated products*

updatedCount := updatedCount + 1;

END LOOP;

*-- Close the cursor*

CLOSE productCursor;

*-- Return a success message with the number of updated products*

RETURN 'Updated ' || updatedCount || ' products successfully.';

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

*-- Handle the case where no matching products are found*

RETURN 'No products found for the given category and date range.';

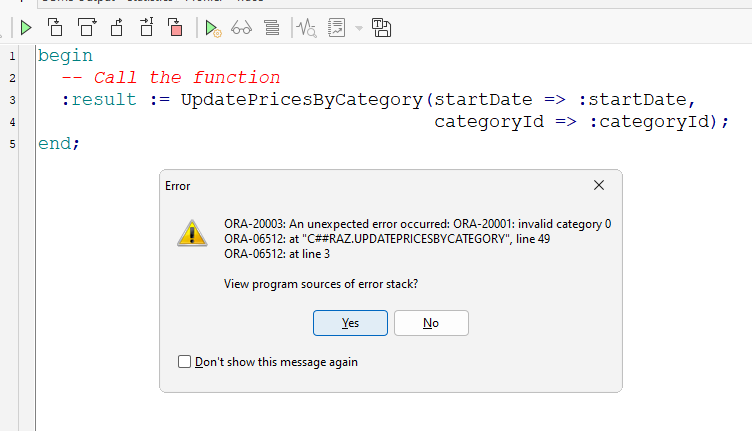
WHEN OTHERS THEN

*-- Handle other unexpected errors*

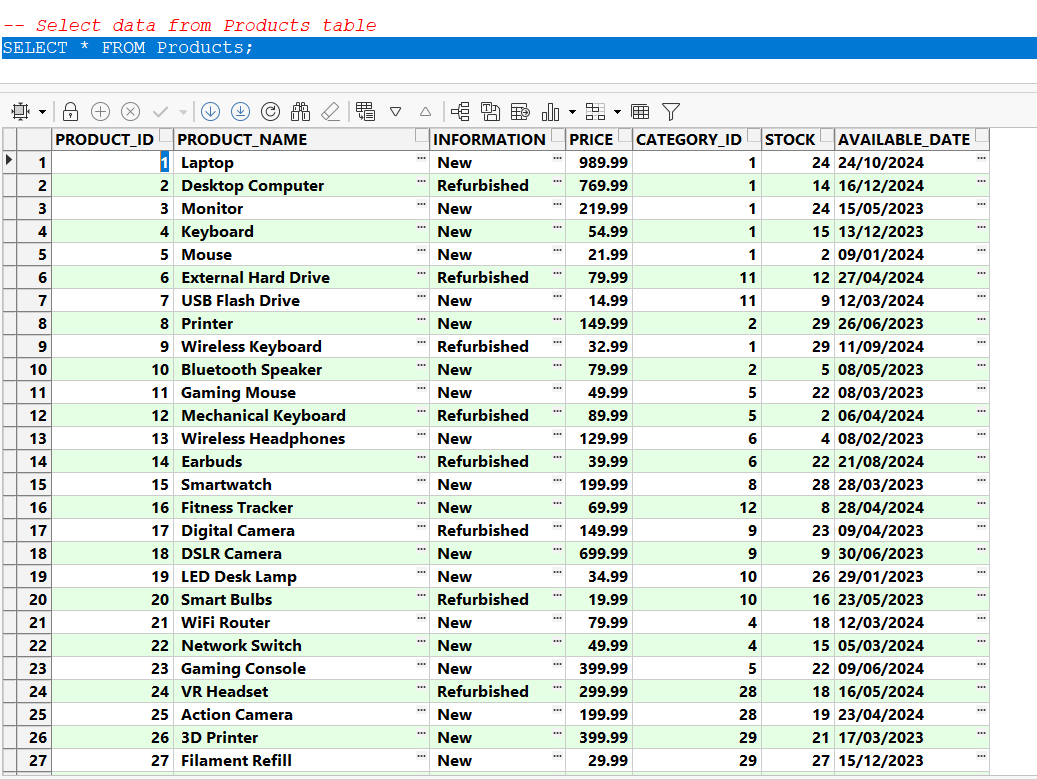
RAISE\_APPLICATION\_ERROR(-20003, 'An unexpected error occurred: ' || SQLERRM);

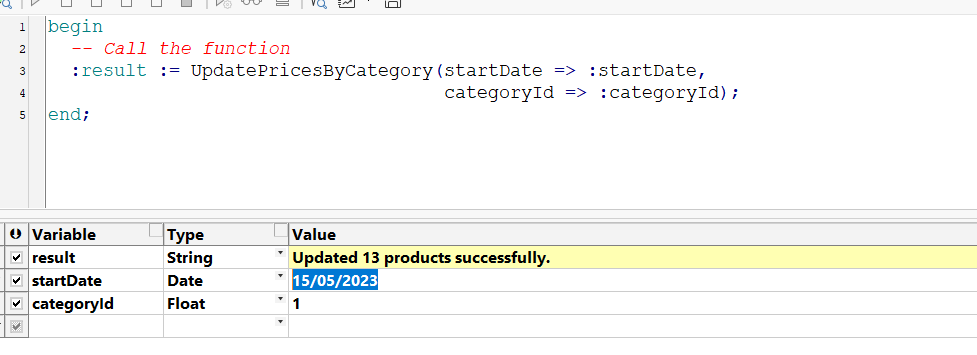
END UpdatePricesByCategory;

חריגה:אם המשתמש הכניס קטגוריה לא תקפה

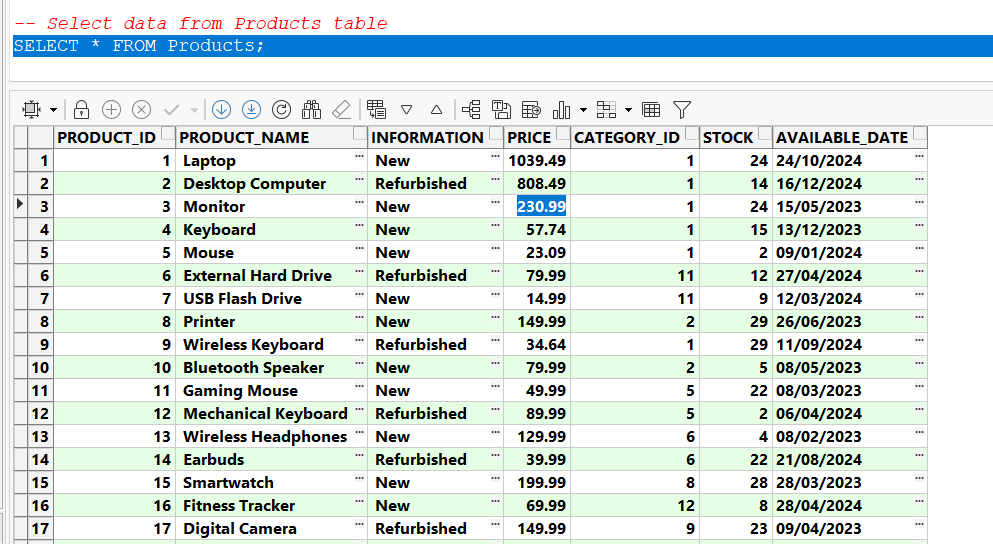


לפני העדכון





אחרי



תוכנית ראשית שקוראת לפרוצדורה PlaceNewOrder ו UpdatePricesByCategory

DECLARE

v\_buyerId INT := 9999; *-- Example buyer ID*

v\_orderId INT := 547775847; *-- Example order ID*

v\_orderDate DATE := SYSDATE; *-- Current date as order date*

v\_trackingNumber VARCHAR2(20) := '666-666'; *-- Example tracking number*

v\_product INT := 1; *-- Example product IDs*

v\_quantity INT := 4; *-- Example quantities*

result VARCHAR2(50);

BEGIN

*-- Call the PlaceNewOrder procedure*

PlaceNewOrder(

buyerId => v\_buyerId,

orderId => v\_orderId,

orderDate => v\_orderDate,

trackingNumber => v\_trackingNumber,

product => v\_product,

quantity => v\_quantity

);

*-- Output success message*

DBMS\_OUTPUT.PUT\_LINE('Order placed successfully.');

:result := UpdatePricesByCategory(startDate => :startDate,

categoryId => :categoryId);

DBMS\_OUTPUT.PUT\_LINE(result);

EXCEPTION

WHEN OTHERS THEN

*-- Output error message*

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

END;