**PROJECT REPORT- *UNIVERSITY FINANCIALDATABASE MANAGEMENT SYSTEM***

**Submitted by:**Mohit

**Problem statement:**

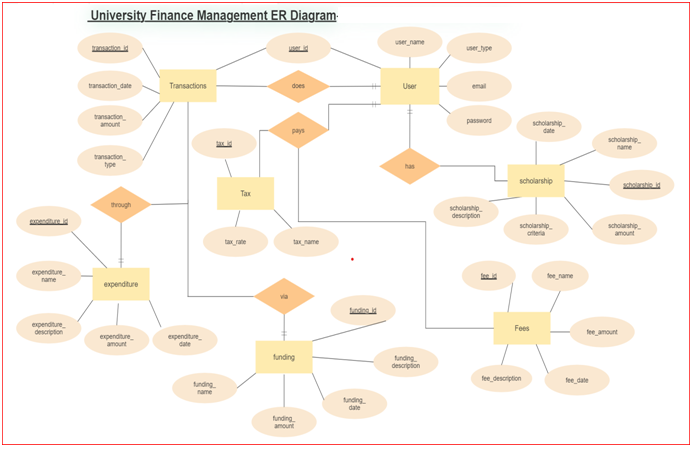
To create a system using PL/SQL for a University Financial Record Keeping and Management functionalities.

**Description of problem:**

The University Finances Management System is designed to help universities manage their finances efficiently. It helps in tracking income and expenses, managing scholarships and funding, and maintaining fee records. This system is developed using PL/SQL, which is a procedural language designed specifically for the Oracle Database. It provides powerful features like stored procedures, functions, triggers, cursors, and packages, which can be used to build complex systems.

**Proposed Entity-Relationship Diagram**

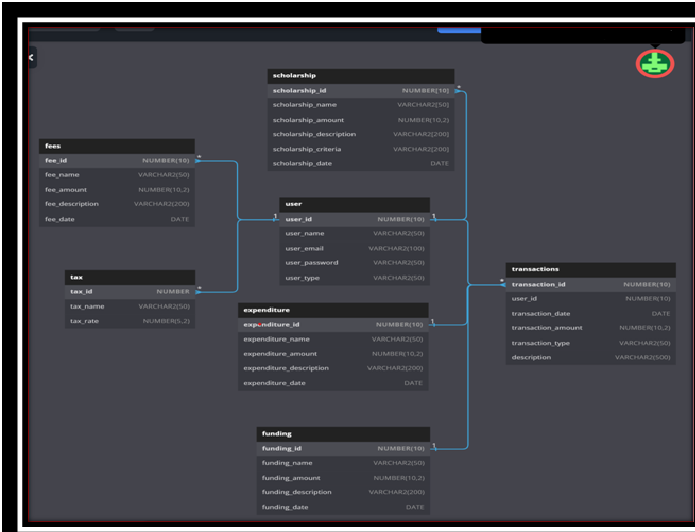
|  |
| --- |
| # The following is ER relationship types.   1. A Transaction record belongs to one   and only one of Expenditure,  Scholarship, Fees, or Tax records  (**one-to-one relationship**).   1. Each Transaction belongs to only one   Expenditure record, but an Expenditure record can have multiple Transactions. (**many-to-one relationship**).   1. A Scholarship record can have multiple   Transactions, but each Transaction belongs to only one Scholarship record (**one-to-many relationship**).   1. A User can have multiple Funding records,   but each Funding record belongs to  only one User  (**one-to-many relationship**). |



**DATABASE DESIGN:**

Tables are pre-normalised and don’t need normalisation:

* **Transactions**: Records of all incoming and outgoing financial transactions, including transaction type (e.g. received or sent), amount, date, and any related metadata (e.g. account number, user ID).
* **Tax**: Information about taxes applicable to the university, including tax rates, payment due dates, and any relevant legislation.
* **Users:** Information about individuals associated with the university, including their name, email address, account number, scholarship amounts, authorisation level, GPA, and any other relevant details.
* **Fees**: Details about fees paid by users, including the amount, date, and reason for the fee.
* **Funding**: Information about grants or other funding received by the university, including the source, amount, and intended use.
* **Expenditure**: Records of all expenses incurred by the university, including the amount, date, category (e.g. salaries, equipment, supplies), and any other relevant information.



Contains the following procedures and functions:

* **calculate\_expenditure**: takes start and end date as input and returns total expenditure for

that period.

* **insert\_transaction**: inserts a new transaction record into the transactions table.
* **University Financials:** Function thatreturns a summary of key financial metrics as a

formatted string for display or export. Calculates values based on data in the Transactions,

Tax, Users, Fees, Funding, and Expenditure tables.

* **add\_user:** This procedure takes several parameters to insert a user record into the users table.
* **delete\_user:** This procedure takes a user ID and deletes the user record from the users table.
* **Update User Info:** Allows administrators to update any of the fields associated with a specific

user record in the Users table using input prompts to determine which fields to update and

collect new values for those fields.

It is integrated with a triggerwhich fires after a new record is inserted into the transactions table and performs the following tasks:

1. **update\_expenditure\_info**- Determines, if the transaction type is "sent", inserts a new record into the expenditure table with the same transaction ID and other details.
2. **update\_relevant\_tables**- Updates the scholarship, tax, funding or fees tables based on the transaction description.
3. **update\_total\_transactions**- It updates the total transactions for the user associated with the new transaction record.

Finally, the code defines a sequence "transactions\_seq" that is used to generate new transactionIDs.

Indexes:

* **Transaction ID**: To quickly locate specific transactions by ID.
* **User ID**: To quickly retrieve all transactions associated with a particular user.
* **Transaction Type**: To quickly retrieve all transactions of a particular type (received, sent).
* **Date**: To allow quick retrieval of transactions within a specific date range.

IMPLEMENTATION:

**CREATE TABLE transactions (**

**transaction\_idNUMBER(15) PRIMARY KEY,**

**transaction\_date DATE,**

**transaction\_amountNUMBER(10,2),**

**transaction\_type VARCHAR2(20) DEFAULT 'RECEIVED',**

**transaction\_accountNUMBER(9) DEFAULT 102103388,**

**transaction\_usernumber(9),**

**description VARCHAR2(100)**

**);**

**CREATE TABLE users (**

**user\_idNUMBER(9) NOT NULL PRIMARY KEY,**

**user\_name VARCHAR2(50) ,**

**user\_email VARCHAR2(100) UNIQUE,**

**user\_account\_numberNUMBER(9),**

**user\_scholarship\_amount number DEFAULT NULL,**

**user\_authorisation\_type VARCHAR2(50),**

**user\_gpaNUMBER(3,2),**

**total\_transactionsNUMBER(10,2) DEFAULT 0**

**);**

**CREATE TABLE fees (**

**fee\_idNUMBER(15) NOT NULL PRIMARY KEY REFERENCES transactions(transaction\_id),**

**fee\_name VARCHAR2(50) ,**

**fee\_amountNUMBER(10,2) ,**

**fee\_description VARCHAR2(200),**

**fee\_dateDATE ,**

**user\_id\_for\_feeNUMBER(9) REFERENCES users(user\_id)**

**);**

**CREATE TABLE funding (**

**funding\_idNUMBER(15) NOT NULL PRIMARY KEY REFERENCES transactions(transaction\_id),**

**funding\_name VARCHAR2(50) ,**

**funding\_amountNUMBER(10,2) ,**

**funding\_description VARCHAR2(200),**

**funding\_dateDATE );**

**CREATE TABLE scholarship (**

**scholarship\_idNUMBER(2) NOT NULL PRIMARY KEY,**

**scholarship\_user\_idNUMBER(9) REFERENCES users(user\_id),**

**scholarship\_name VARCHAR2(50) ,**

**scholarship\_amountNUMBER(10,2) ,**

**scholarship\_description VARCHAR2(200),**

**scholarship\_criteria VARCHAR2(200)**

**);**

**CREATE TABLE expenditure (**

**expenditure\_idNUMBER(15) NOT NULL PRIMARY KEY REFERENCES transactions(transaction\_id),**

**expenditure\_name VARCHAR2(50),**

**expenditure\_amountNUMBER(10,2),**

**expenditure\_description VARCHAR2(200),**

**expenditure\_date DATE**

**);**

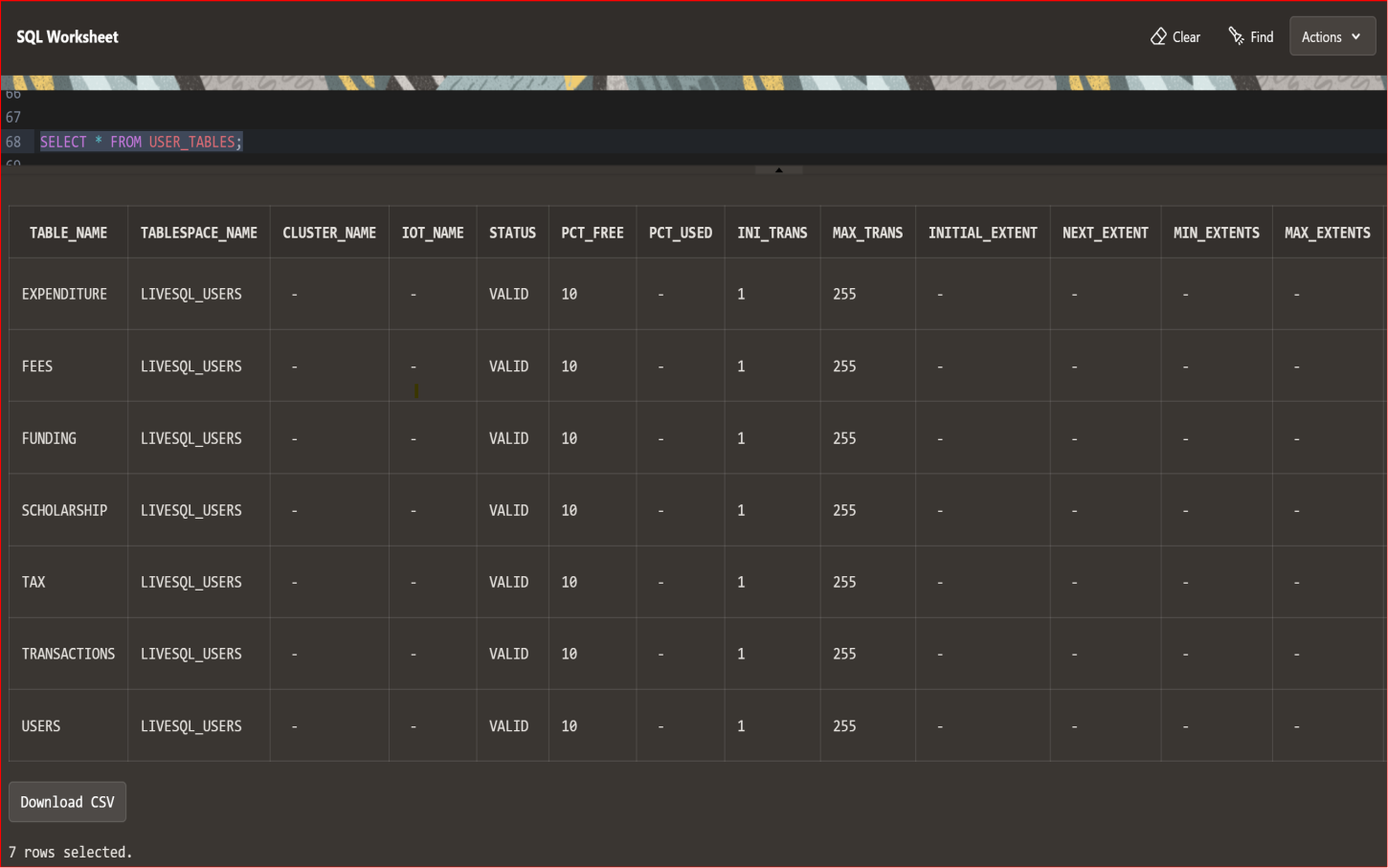
**CREATE TABLE tax (**

**tax\_idNUMBER(10) NOT NULL PRIMARY KEY,**

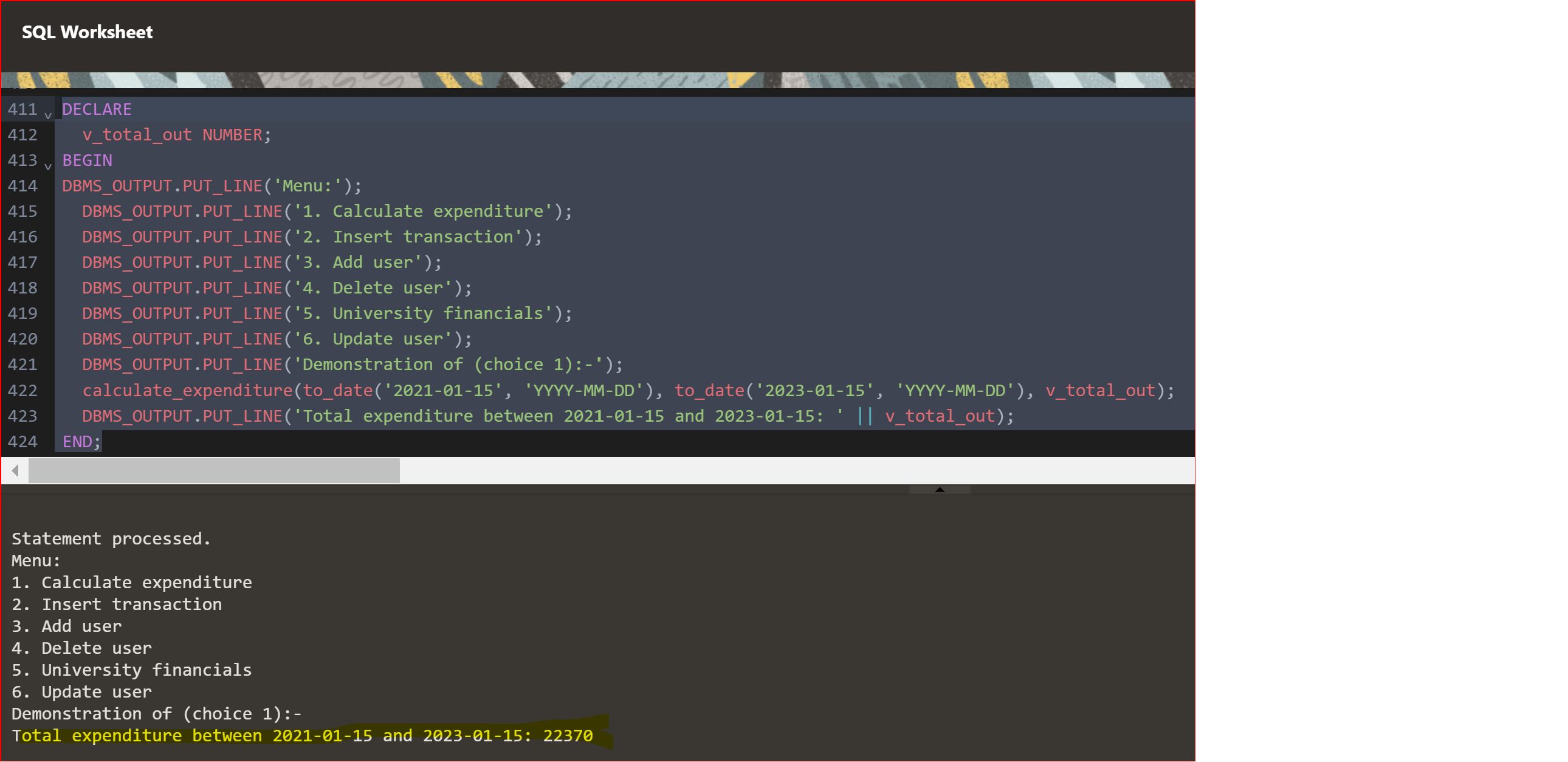
**tax\_name VARCHAR2(50),**

**tax\_amount NUMBER,**

**tax\_rateNUMBER(5,2)**

**);**

* **THE CALCULATE\_EXPENDITURE PROCEDURE**



* This stored procedure is VERY useful in calculating transaction based financial analysis or expenditurebreakdown, as it provides a quick and efficient way to calculate total expenditure within a given date range.

**CREATE OR REPLACE PROCEDURE calculate\_expenditure (**

**p\_start\_date IN DATE,**

**p\_end\_date IN DATE,**

**p\_total\_out OUT NUMBER) IS**

**v\_total\_outNUMBER(10,2) := 0;**

**v\_expenditure\_nameexpenditure.expenditure\_name%TYPE;**

**v\_expenditure\_amountexpenditure.expenditure\_amount%TYPE;**

**v\_expenditure\_dateexpenditure.expenditure\_date%TYPE;**

**CURSOR c\_expenditures IS**

**SELECT expenditure\_name, expenditure\_amount, expenditure\_date**

**FROM expenditure**

**WHERE expenditure\_date>= p\_start\_date AND expenditure\_date<= p\_end\_date;**

**BEGIN**

**FOR c\_expense IN c\_expenditures LOOP**

**v\_total\_out := v\_total\_out + c\_expense.expenditure\_amount;**

**END LOOP;**

**p\_total\_out := v\_total\_out;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('No expenditure found within specified date range.');**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);**

**END;**

**FEATURES**

* This is a PL/SQL procedure for calculating the total expenditure for a university during a specified date range. The procedure takes in two date parameters, p\_start\_date and p\_end\_date, which define the start and end dates of the expenditure period to be analyzed. The procedure then uses a cursor to retrieve all expenditures within the specified date range from a table called expenditure.
* For each expenditure, the procedure adds the expenditure amount to a running total, v\_total\_out. Once all expenditures have been added up, the final total expenditure is returned through the OUT parameter, p\_total\_out.
* This is a PL/SQL procedure called takes in two input parameters: "p\_start\_date" and "p\_end\_date," both of which are of type DATE. The procedure also has one output parameter "p\_total\_out," which is the required calculation.
* **THE INSERT\_TRANSACTIONPROCEDURE AND TRANSACTION\_TRIGGERS**

**CREATE OR REPLACE PROCEDURE insert\_transaction (**

**p\_transaction\_date IN DATE,**

**p\_transaction\_amount IN NUMBER,**

**p\_transaction\_type IN VARCHAR2,**

**p\_transaction\_account IN NUMBER DEFAULT 102103388,**

**p\_transaction\_user IN NUMBER,**

**p\_description IN VARCHAR2 DEFAULT NULL) IS**

**v\_transaction\_idNUMBER(15);**

**v\_countNUMBER(1);**

**v\_transaction\_rowtransactions%ROWTYPE; -- Declare a variable of type transactions%ROWTYPE**

**BEGIN**

**-- Generate a transaction ID**

**LOOP**

**v\_transaction\_id := TRUNC(DBMS\_RANDOM.VALUE(100000000000000, 999999999999999));**

**SELECT COUNT(\*) INTO v\_count FROM transactions WHERE transaction\_id = v\_transaction\_id;**

**EXIT WHEN v\_count = 0;**

**END LOOP;**

**-- Insert the transaction record**

**INSERT INTO transactions (transaction\_id, transaction\_date, transaction\_amount, transaction\_type,**

**transaction\_account, transaction\_user, description)**

**VALUES (v\_transaction\_id, p\_transaction\_date, p\_transaction\_amount, p\_transaction\_type,**

**p\_transaction\_account, p\_transaction\_user, p\_description);**

**-- Get the inserted record**

**SELECT \* INTO v\_transaction\_row FROM transactions WHERE transaction\_id = v\_transaction\_id;**

**-- Display the inserted record**

**DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_transaction\_row.transaction\_id);**

**DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || v\_transaction\_row.transaction\_date);**

**DBMS\_OUTPUT.PUT\_LINE('Transaction Amount: ' || v\_transaction\_row.transaction\_amount);**

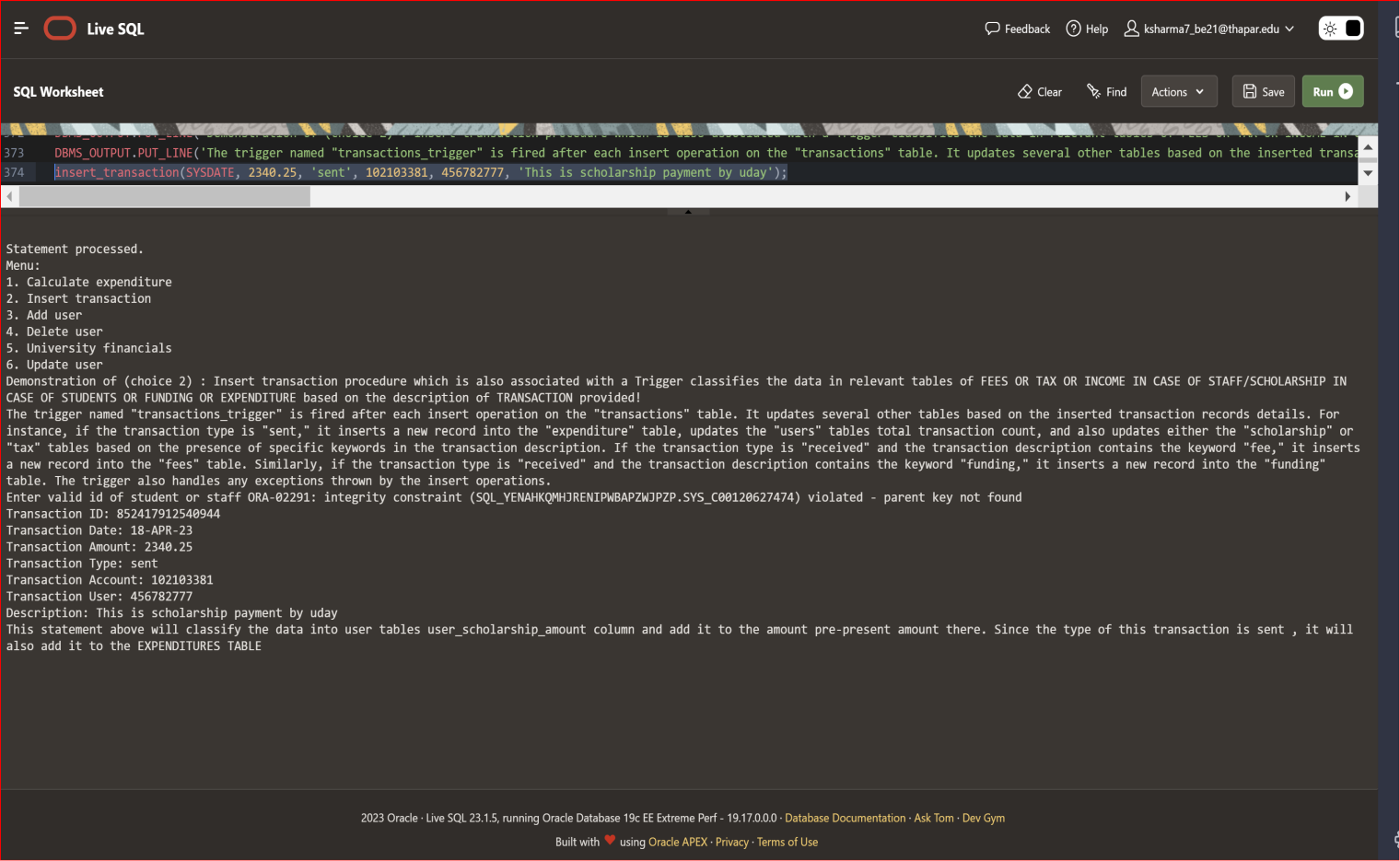
**DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || v\_transaction\_row.transaction\_type);**

**DBMS\_OUTPUT.PUT\_LINE('Transaction Account: ' || v\_transaction\_row.transaction\_account);**

**DBMS\_OUTPUT.PUT\_LINE('Transaction User: ' || v\_transaction\_row.transaction\_user);**

**DBMS\_OUTPUT.PUT\_LINE('Description: ' || v\_transaction\_row.description);**

**END;**



* Notice how the addition of transaction type as ‘sent’ has also added it to EXPENDITURE

TABLE aswell. This will help calculate and keep track of expenditures.

* The transaction\_description column will help to classify whether the transaction will be

added to ‘fees’, ‘funding’ table (tables in which the university receives money in form of

fees paid by students or funding received from government) or if the transaction will be

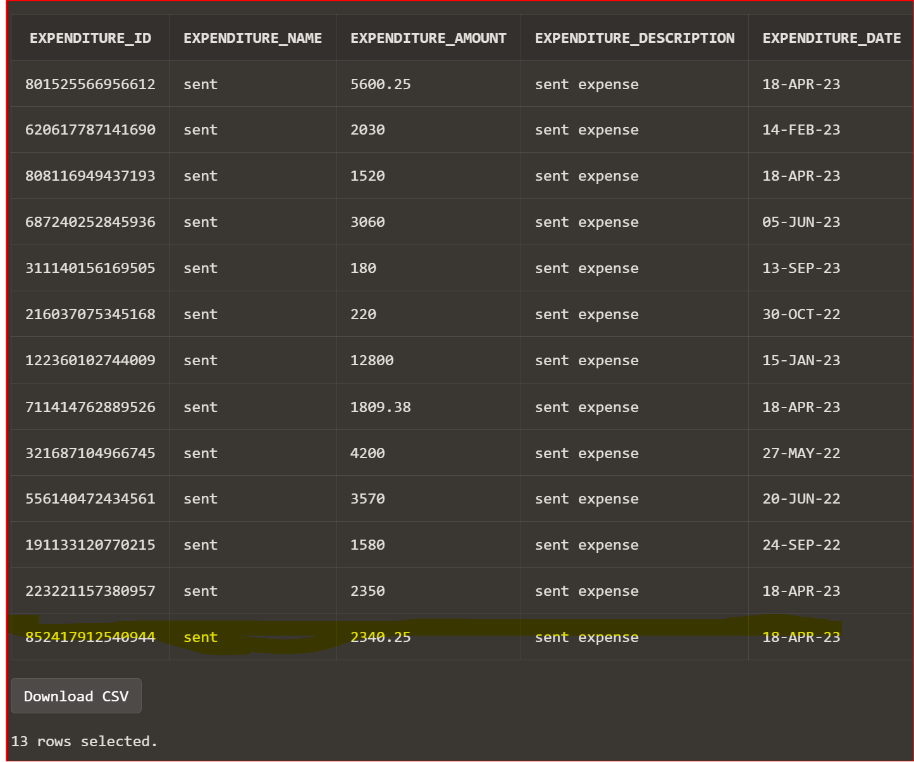
added to ‘tax’ or ‘scholarship/income’ table.

* This classification is not exhaustive and there will be some transactions which wont be

classified in any other tables except transaction.

* In our example we have updated transactions such that transaction\_type is ‘sent’,

hence it’ll be automatically TRIGGERED into EXECUTION TABLE.



**TRANSACTION\_TRIGGERS CODE DEFINITION**

**CREATE OR REPLACE TRIGGER transactions\_trigger**

**AFTER INSERT ON transactions**

**FOR EACH ROW**

**BEGIN**

**-- Update expenditure table**

**IF (LOWER(:NEW.transaction\_type) = 'sent') THEN**

**INSERT INTO expenditure (expenditure\_id, expenditure\_name, expenditure\_amount,**

**expenditure\_description, expenditure\_date)**

**VALUES (:NEW.transaction\_id, :NEW.transaction\_type, :NEW.transaction\_amount,**

**:NEW.transaction\_type || ' expense', :NEW.transaction\_date);**

**UPDATE users SET total\_transactions = (total\_transactions+1) WHERE user\_id =**

**:NEW.transaction\_user;**

**-- Update scholarship and tax tables**

**IF (LOWER(:NEW.description) LIKE '%scholarship%') THEN**

**UPDATE users SET user\_scholarship\_amount = NVL(user\_scholarship\_amount, 0) +**

**:NEW.transaction\_amount WHERE user\_id = :NEW.transaction\_user;**

**INSERT INTO scholarship (scholarship\_id, scholarship\_user\_id, scholarship\_name,**

**scholarship\_amount, scholarship\_description, scholarship\_criteria)**

**VALUES (99, :NEW.transaction\_user, 'Scholarship', :NEW.transaction\_amount, 'Scholarship**

**award', 'Bonus');**

**ELSIF (LOWER(:NEW.description) LIKE '%tax%') THEN**

**INSERT INTO tax (tax\_id, tax\_name, tax\_amount, tax\_rate)**

**VALUES (:NEW.transaction\_id, 'Transaction Tax', :NEW.transaction\_amount, 0.2);**

**END IF;**

**END IF;**

**-- Update fees table**

**IF (LOWER(:NEW.transaction\_type) = 'received' AND :NEW.description IS NOT NULL AND**

**LOWER(:NEW.description) LIKE '%fee%') THEN**

**INSERT INTO fees (fee\_id, fee\_name, fee\_amount, fee\_description, fee\_date, user\_id\_for\_fee)**

**VALUES (:NEW.transaction\_id, 'FEE paid', :NEW.transaction\_amount , :NEW.description,**

**:NEW.transaction\_date, :NEW.transaction\_user);**

**END IF;**

**-- Update funding table**

**IF (LOWER(:NEW.transaction\_type) = 'received' AND :NEW.description IS NOT NULL AND**

**LOWER(:NEW.description) LIKE '%funding%') THEN**

**INSERT INTO funding (funding\_id, funding\_name, funding\_amount, funding\_description,**

**funding\_date)**

**VALUES (:NEW.transaction\_id, 'Funding', :NEW.transaction\_amount, 'Funding received',**

**:NEW.transaction\_date);**

**END IF;**

**EXCEPTION**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE('Enter valid id of student or staff ' || SQLERRM);**

**END;**

**ADD USER PROCEDURE DEFINITION:**

**CREATE OR REPLACE PROCEDURE add\_user (**

**p\_user\_id IN NUMBER,**

**p\_user\_name IN VARCHAR2,**

**p\_user\_email IN VARCHAR2,**

**p\_user\_account\_number IN NUMBER,**

**p\_user\_scholarship\_amount IN NUMBER DEFAULT NULL,**

**p\_user\_authorisation\_type IN VARCHAR2 DEFAULT NULL,**

**p\_user\_gpa IN NUMBER DEFAULT NULL**

**) IS**

**BEGIN**

**INSERT INTO users (user\_id, user\_name, user\_email, user\_account\_number, user\_scholarship\_amount,**

**user\_authorisation\_type, user\_gpa)**

**VALUES (p\_user\_id, p\_user\_name, p\_user\_email, p\_user\_account\_number, p\_user\_scholarship\_amount,**

**p\_user\_authorisation\_type, p\_user\_gpa);**

**COMMIT;**

**DBMS\_OUTPUT.PUT\_LINE('User added successfully.');**

**EXCEPTION**

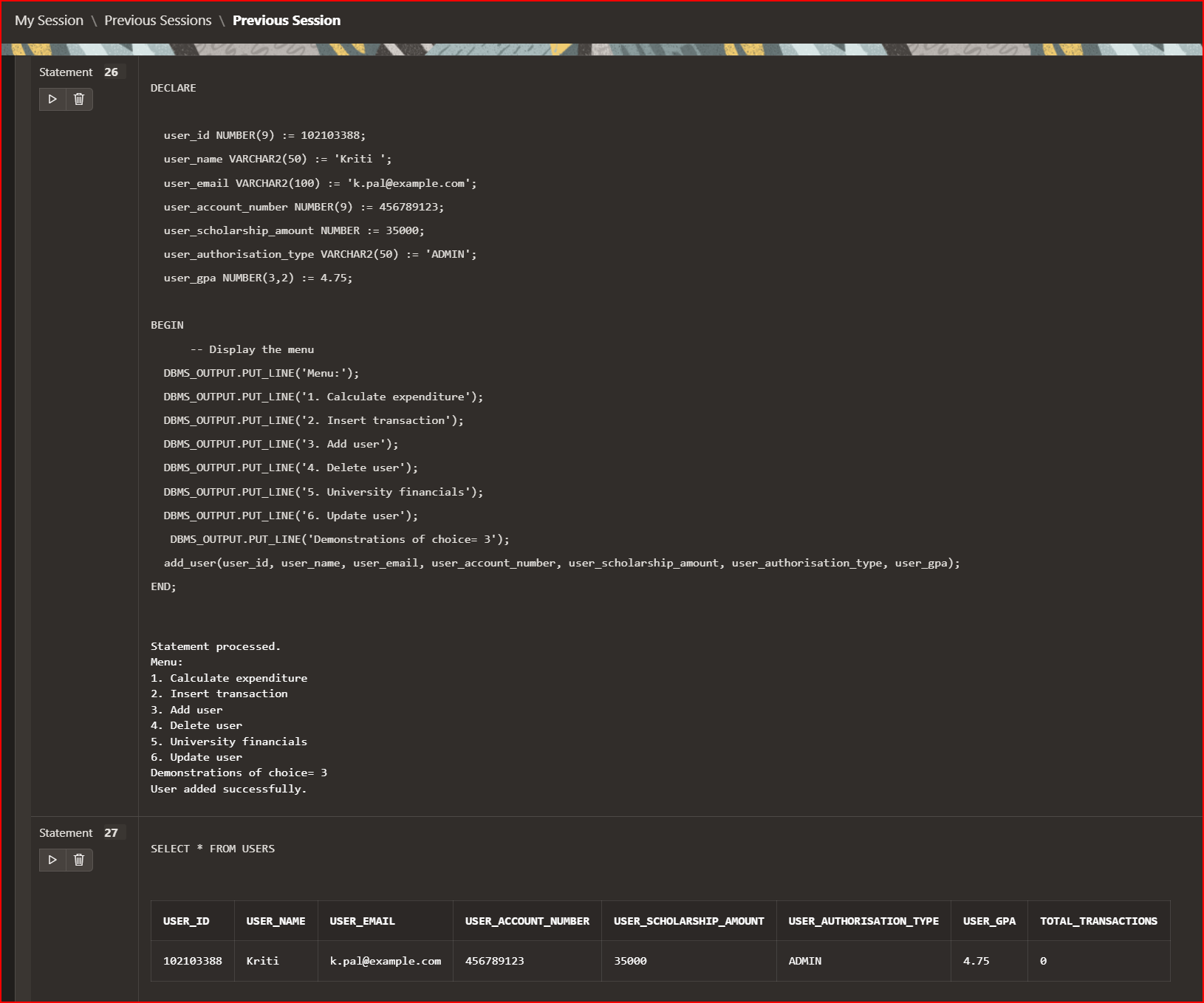
**WHEN DUP\_VAL\_ON\_INDEX THEN**

**DBMS\_OUTPUT.PUT\_LINE('User email already exists.');**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE('An error occurred while adding user.');**

**END;**



**FEATURES:**

* This procedure is useful for adding new users to a database table in a controlled manner and with error handling to ensure data integrity.
* This procedure is designed for use by a university's financial management system to add new users to the system.
* If the error is due to a duplicate id/email address, the procedure will output a message indicating that the user id/ email already exists.

**UPDATE USER PROCEDURE DEFINITION:**

**CREATE OR REPLACE PROCEDURE update\_user\_info(p\_user\_id IN NUMBER)**

**IS**

**v\_user\_nameusers.user\_name%TYPE;**

**v\_user\_emailusers.user\_email%TYPE;**

**v\_user\_account\_numberusers.user\_account\_number%TYPE;**

**v\_user\_scholarship\_amountusers.user\_scholarship\_amount%TYPE;**

**v\_user\_authorisation\_typeusers.user\_authorisation\_type%TYPE;**

**v\_user\_gpausers.user\_gpa%TYPE;**

**v\_total\_transactionsusers.total\_transactions%TYPE;**

**v\_choice VARCHAR2(1);**

**BEGIN**

**SELECT user\_name, user\_email, user\_account\_number, user\_scholarship\_amount,**

**user\_authorisation\_type, user\_gpa, total\_transactions**

**INTO v\_user\_name, v\_user\_email, v\_user\_account\_number,**

**v\_user\_scholarship\_amount, v\_user\_authorisation\_type, v\_user\_gpa,**

**v\_total\_transactions**

**FROM users**

**WHERE user\_id = p\_user\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Current User Details:');**

**DBMS\_OUTPUT.PUT\_LINE('User Name: ' || v\_user\_name);**

**DBMS\_OUTPUT.PUT\_LINE('User Email: ' || v\_user\_email);**

**DBMS\_OUTPUT.PUT\_LINE('User Account Number: ' || v\_user\_account\_number);**

**DBMS\_OUTPUT.PUT\_LINE('User Scholarship Amount: ' || v\_user\_scholarship\_amount);**

**DBMS\_OUTPUT.PUT\_LINE('User Authorisation Type: ' || v\_user\_authorisation\_type);**

**DBMS\_OUTPUT.PUT\_LINE('User GPA: ' || v\_user\_gpa);**

**DBMS\_OUTPUT.PUT\_LINE('Total Transactions: ' || v\_total\_transactions);**

**DBMS\_OUTPUT.PUT\_LINE('Choose which fields to update:');**

**DBMS\_OUTPUT.PUT\_LINE('1. User Name');**

**DBMS\_OUTPUT.PUT\_LINE('2. User Email');**

**DBMS\_OUTPUT.PUT\_LINE('3. User Account Number');**

**DBMS\_OUTPUT.PUT\_LINE('4. User Scholarship Amount');**

**DBMS\_OUTPUT.PUT\_LINE('5. User Authorisation Type');**

**DBMS\_OUTPUT.PUT\_LINE('6. User GPA');**

**DBMS\_OUTPUT.PUT\_LINE('7. Total Transactions');**

**DBMS\_OUTPUT.PUT\_LINE('Enter Choice (1-7):');**

**v\_choice := '&choice';**

**CASE v\_choice**

**WHEN '1' THEN**

**SELECT '&user\_name' INTO v\_user\_name FROM dual;**

**WHEN '2' THEN**

**SELECT '&user\_email' INTO v\_user\_email FROM dual;**

**WHEN '3' THEN**

**SELECT '&user\_account\_number' INTO v\_user\_account\_number FROM dual;**

**WHEN '4' THEN**

**SELECT '&user\_scholarship\_amount' INTO v\_user\_scholarship\_amount FROM dual;**

**WHEN '5' THEN**

**SELECT '&user\_authorisation\_type' INTO v\_user\_authorisation\_type FROM dual;**

**WHEN '6' THEN**

**SELECT '&user\_gpa' INTO v\_user\_gpa FROM dual;**

**WHEN '7' THEN**

**SELECT '&total\_transactions' INTO v\_total\_transactions FROM dual;**

**ELSE**

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

**END CASE;**

**UPDATE users**

**SET user\_name = v\_user\_name,**

**user\_email = v\_user\_email,**

**user\_account\_number = v\_user\_account\_number,**

**user\_scholarship\_amount = v\_user\_scholarship\_amount,**

**user\_authorisation\_type = v\_user\_authorisation\_type,**

**user\_gpa = v\_user\_gpa,**

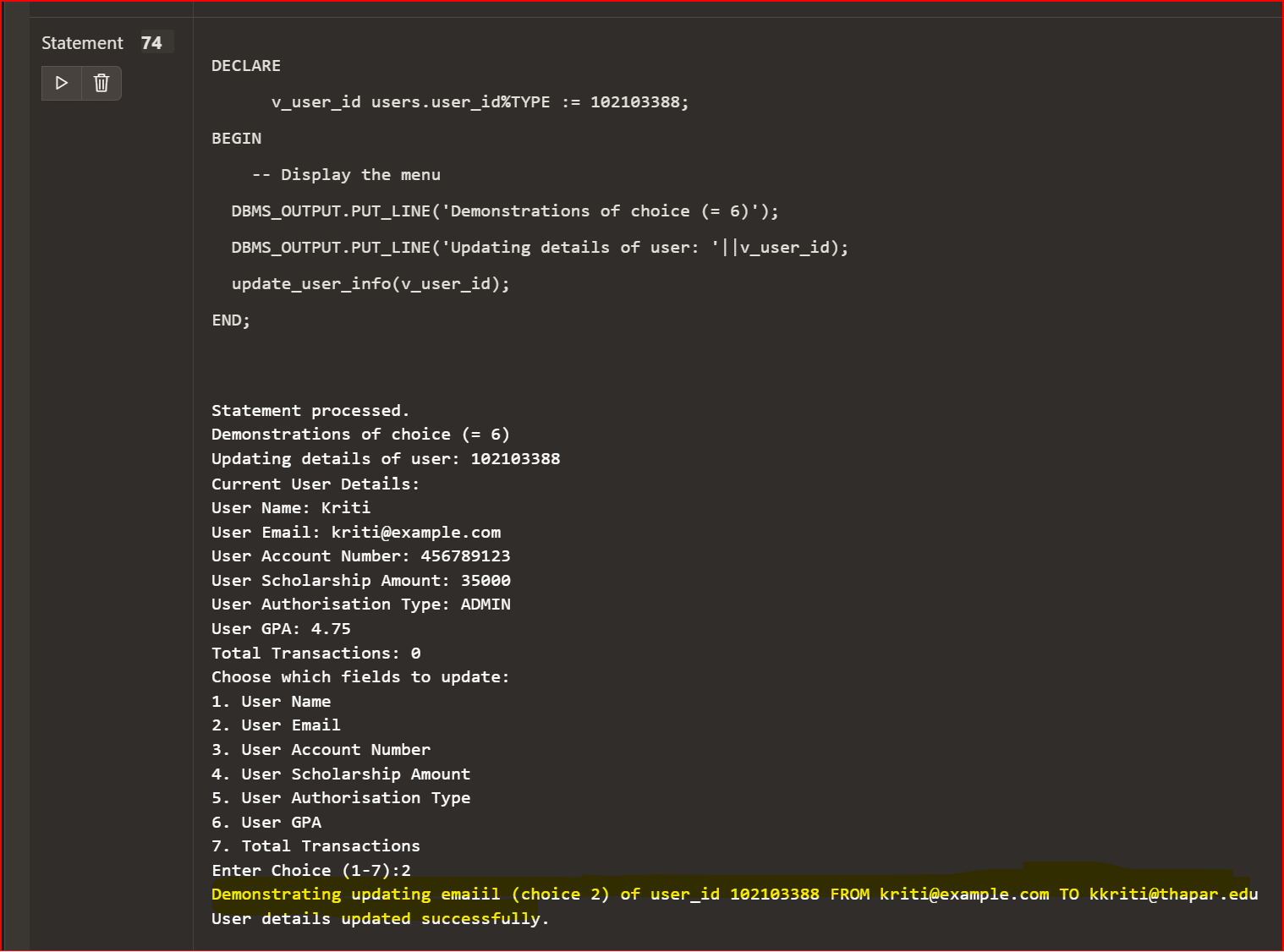
**total\_transactions = v\_total\_transactions**

**WHERE user\_id = p\_user\_id;**

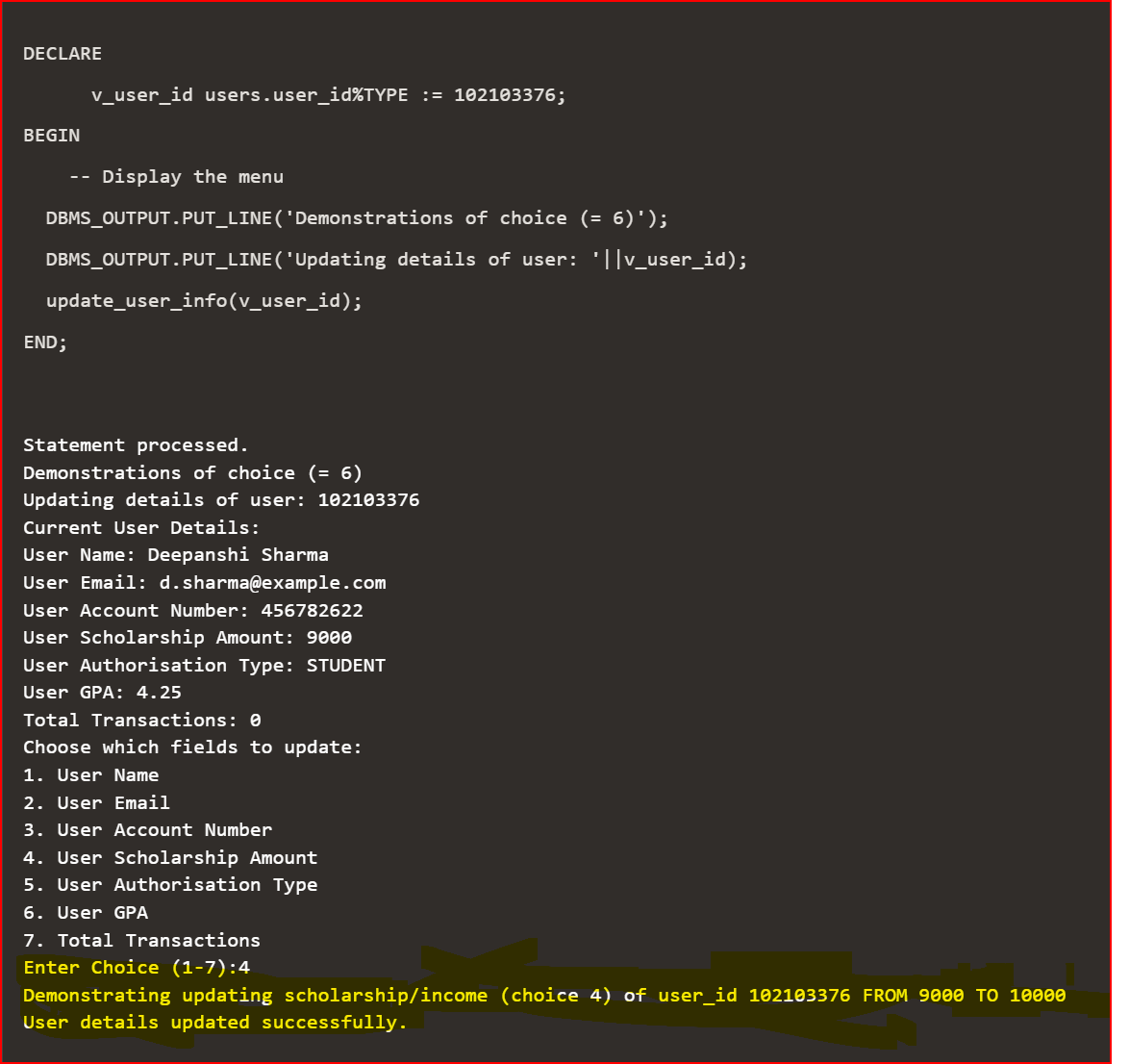
**DBMS\_OUTPUT.PUT\_LINE('User details updated successfully.');**

**END;**

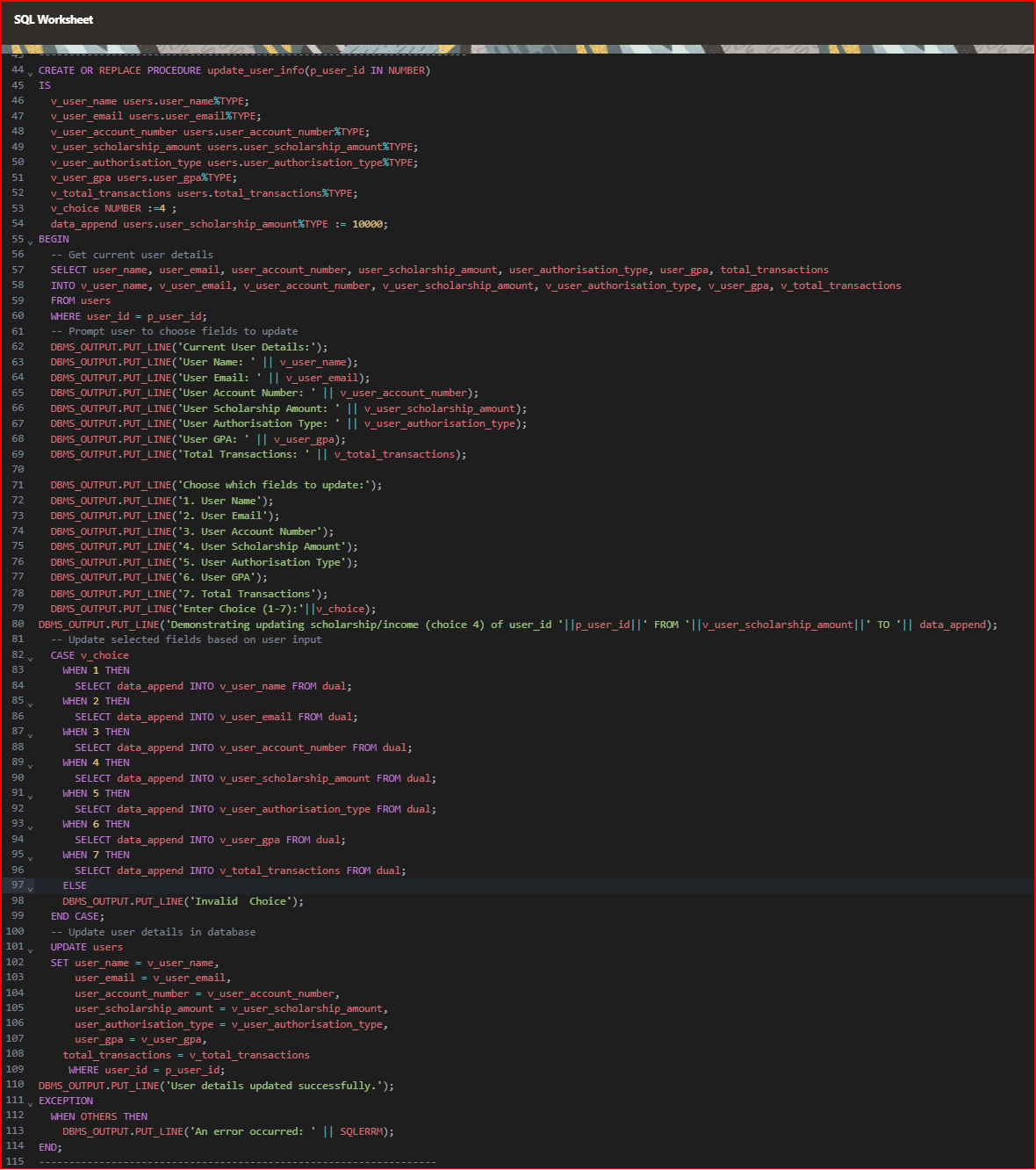
**FEATURES AND EXECUTION:**

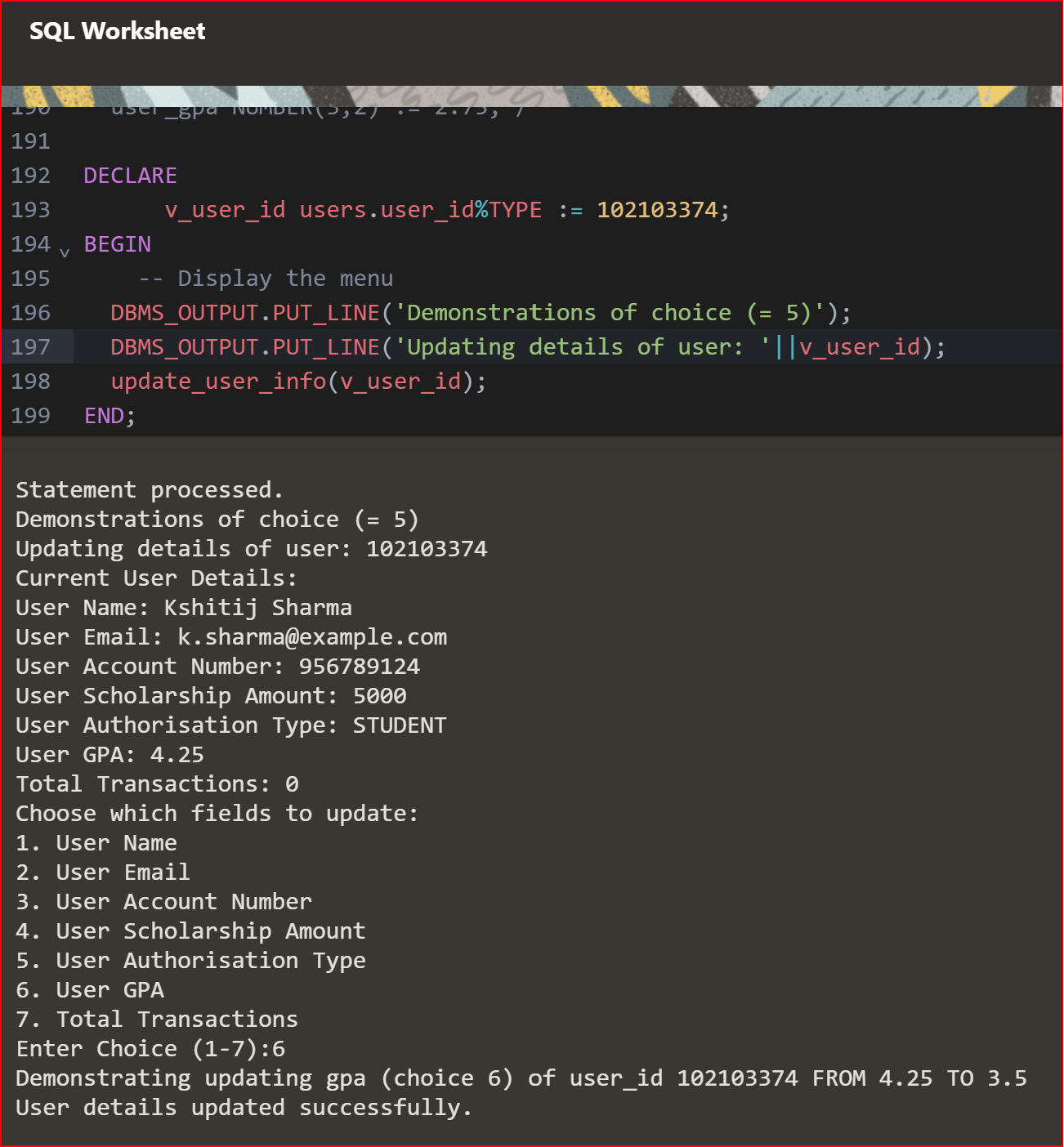


* **This is a PL/SQL stored procedure that updates user information in the users table of a University Financial Management System.**
* **The procedure takes a user ID as input and prompts the user to choose which fields to update. The available fields are user name, email, account number, scholarship amount, authorisation type, GPA, and total transactions.**
* **The procedure then updates the selected fields for the specified user ID.**



* **The procedure takes in one input parameter p\_user\_id which is the unique identifier for the user whose details need to be updated. It then fetches the current user details from the users table .**
* **After fetching the user details, the procedure displays them on the console. It then prompts the user to choose which fields they want to update by displaying a menu of options from 1 to 7.**
* **The procedure uses a CASE statement to determine which field the user wants to update based on their input choice. For each field, the procedure prompts the user to enter the new valueS.**

**UNIVERSITY FINANCIAL SUMMARY FUNCTION DEMONSTRARION:**



**CREATE OR REPLACE FUNCTION university\_financials RETURN VARCHAR2 AS**

**total\_received\_transactionsNUMBER(10,2);**

**total\_received\_amountNUMBER(10,2);**

**total\_sent\_transactionsNUMBER(10,2);**

**total\_sent\_amountNUMBER(10,2);**

**total\_tax\_amountNUMBER(10,2);**

**total\_user\_scholarshipsNUMBER(10,2);**

**total\_feesNUMBER(10,2);**

**total\_fundingNUMBER(10,2);**

**total\_expenditureNUMBER(10,2);**

**net\_scholarships\_givenNUMBER(10,2);**

**net\_expenditure\_after\_scholarshipsNUMBER(10,2);**

**net\_state\_of\_financialsNUMBER(10,2);**

**profit\_loss\_label VARCHAR2(20);**

**BEGIN**

**-- Calculate total received transactions and amount**

**SELECT COUNT(\*), SUM(transaction\_amount)**

**INTO total\_received\_transactions, total\_received\_amount**

**FROM transactions**

**WHERE transaction\_type = 'RECEIVED';**

**-- Calculate total sent transactions and amount**

**SELECT COUNT(\*), SUM(transaction\_amount)**

**INTO total\_sent\_transactions, total\_sent\_amount**

**FROM transactions**

**WHERE transaction\_type = 'SENT';**

**-- Calculate total tax amount**

**SELECT SUM(tax\_amount \* tax\_rate)**

**INTO total\_tax\_amount**

**FROM tax;**

**-- Calculate total user scholarships**

**SELECT SUM(user\_scholarship\_amount)**

**INTO total\_user\_scholarships**

**FROM users;**

**-- Calculate total fees**

**SELECT SUM(fee\_amount)**

**INTO total\_fees**

**FROM fees;**

**-- Calculate total funding**

**SELECT SUM(funding\_amount)**

**INTO total\_funding**

**FROM funding;**

**-- Calculate total expenditure**

**SELECT SUM(expenditure\_amount)**

**INTO total\_expenditure**

**FROM expenditure;**

**-- Calculate net state of financials**

**net\_state\_of\_financials := total\_received\_amount - total\_sent\_amount–**

**total\_tax\_amount ;**

**-- Determine if the university made a profit or a loss**

**IF net\_state\_of\_financials> 0 THEN**

**profit\_loss\_label := 'Profit';**

**ELSE**

**profit\_loss\_label := 'Loss';**

**END IF;**

**RETURN 'Financial aspects of the university:' || CHR(10) ||**

**'Total received transactions: '||total\_received\_transactions||CHR(10)||**

**'Total received amount: ' || total\_received\_amount || CHR(10) ||**

**'Total sent transactions: ' || total\_sent\_transactions || CHR(10) ||**

**'Total sent amount: ' || total\_sent\_amount || CHR(10) ||**

**'Total tax amount: ' || total\_tax\_amount || CHR(10) ||**

**'Total user scholarships: ' || total\_user\_scholarships || CHR(10) ||**

**'Total fees: ' || total\_fees || CHR(10) ||**

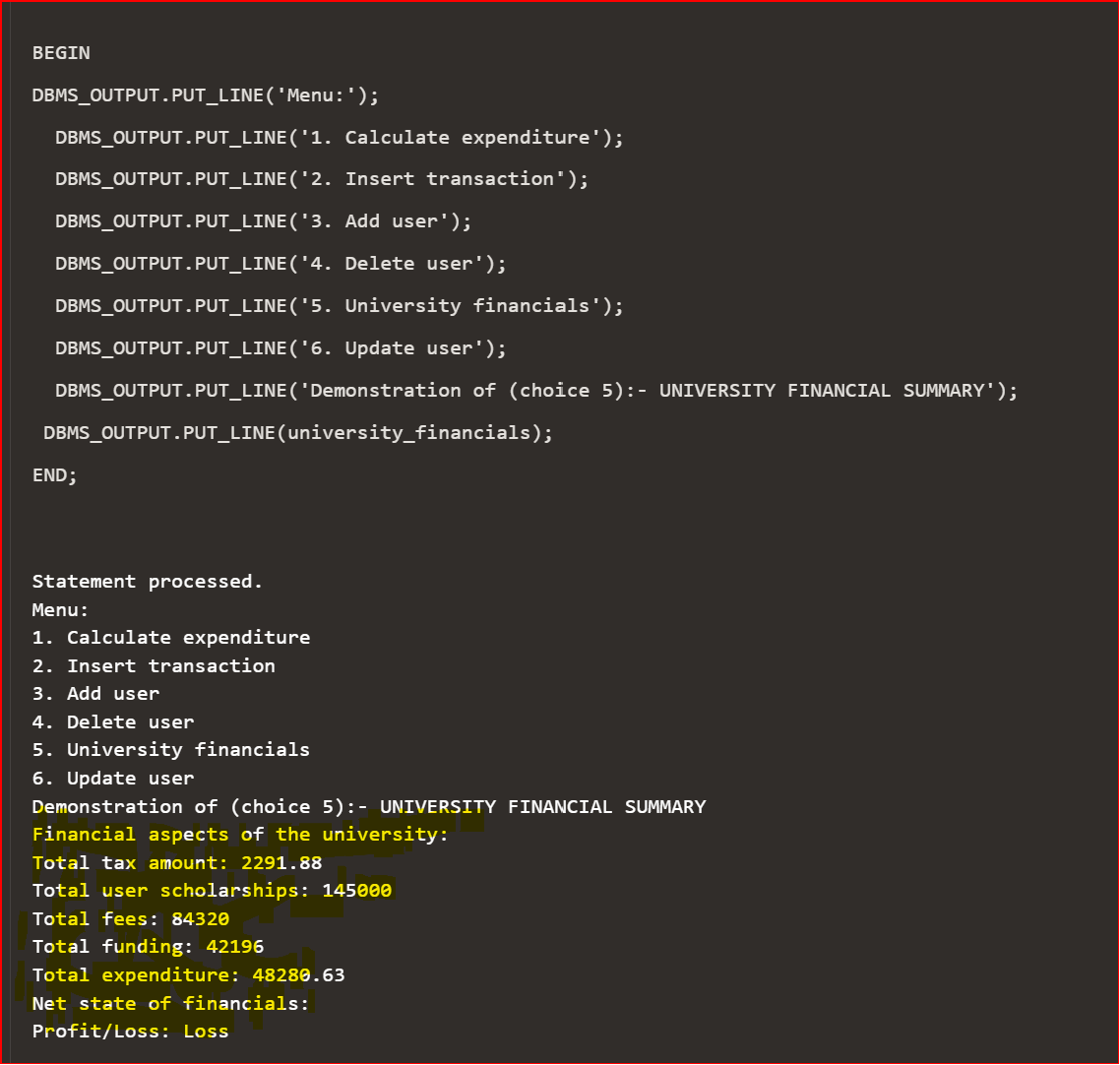
**'Total funding: ' || total\_funding || CHR(10) ||**

**'Total expenditure: ' || total\_expenditure || CHR(10) ||**

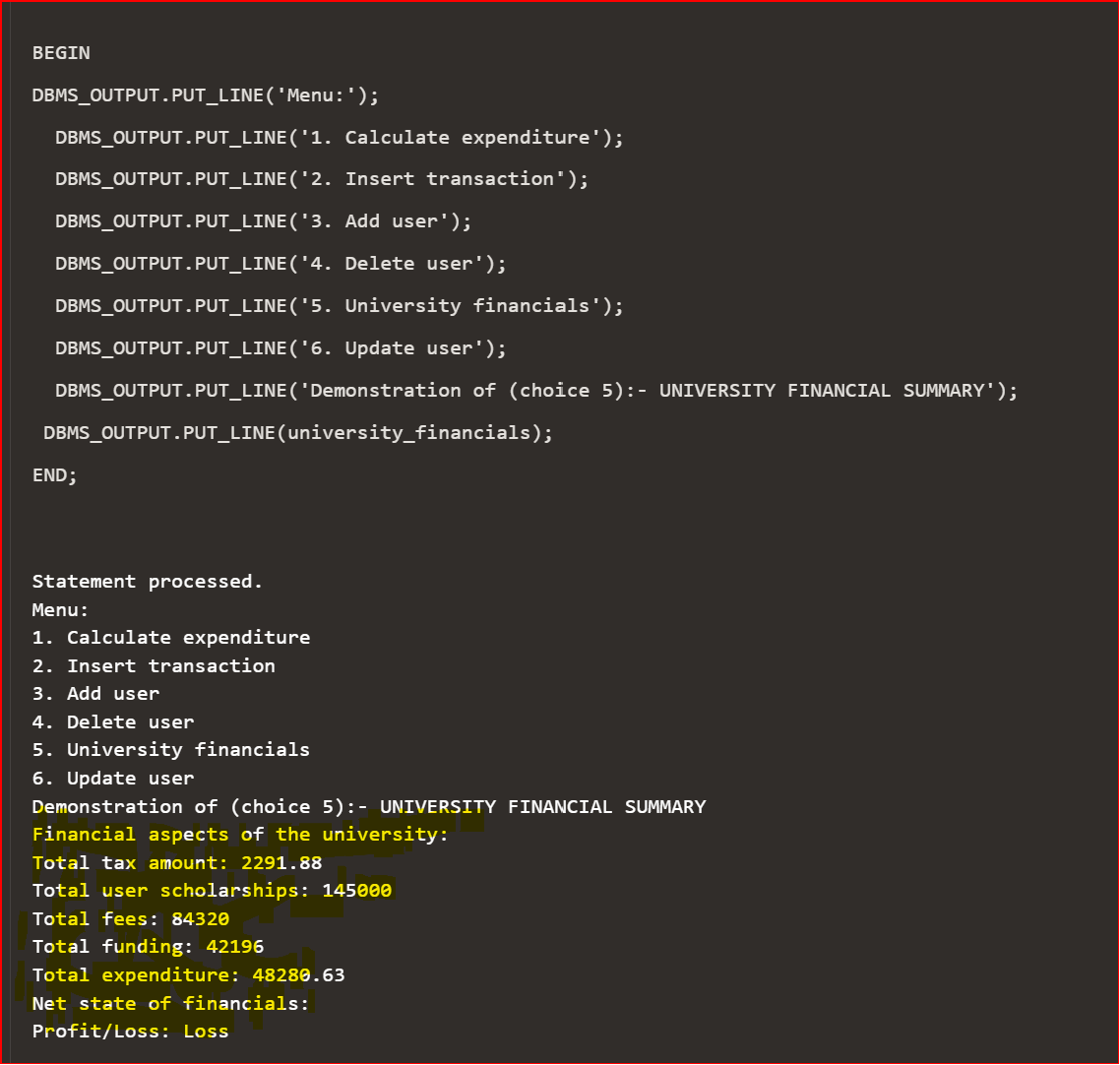
**'Net state of financials: ' || net\_state\_of\_financials || CHR(10) ||**

**'Profit/Loss: ' || profit\_loss\_label;**

**END;**







Overview of what the function does:

1. Declares a number of variables to hold various financial metrics.
2. Executes several SQL queries to calculate the values of these variables. For example, it calculates the total number and amount of transactions received and sent, the total tax amount, the total user scholarships, the total fees, the total funding, and the total expenditure.
3. Calculates the net state of financials by subtracting the total amount of sent transactions and the total tax amount from the total amount of received transactions.
4. Determines whether the university made a profit or a loss based on the net state of financials.
5. Constructs a string that includes all the calculated financial metrics and returns it.

This function can be useful for financial management purposes, as it provides a quick and easy way to get an overview of the university's financial situation.

**CHOICE ‘4’ DEMONSTRARION:**

**CREATE OR REPLACE PROCEDURE delete\_user(p\_user\_id IN NUMBER) IS**

**BEGIN**

**DELETE FROM users WHERE user\_id = p\_user\_id;**

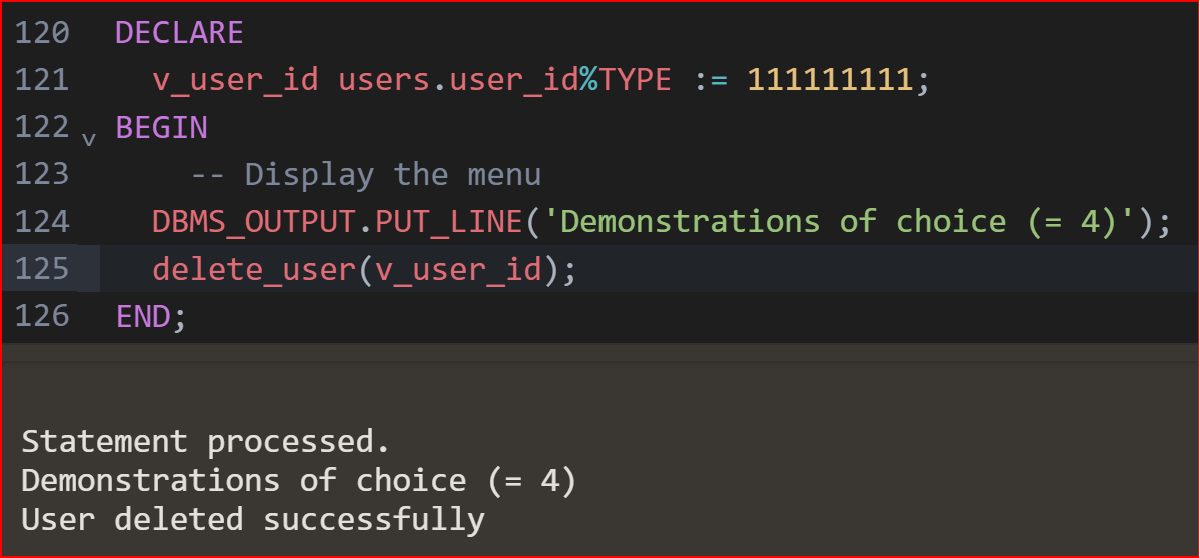
**DBMS\_OUTPUT.PUT\_LINE('User deleted successfully');**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('User with ID ' || p\_user\_id || ' not found');**

**END;**



**CODE FOR MENU:**

**DECLARE**

**choice NUMBER;**

**BEGIN**

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

**DBMS\_OUTPUT.PUT\_LINE('1. Calculate expenditure');**

**DBMS\_OUTPUT.PUT\_LINE('2. Insert transaction');**

**DBMS\_OUTPUT.PUT\_LINE('3. Add user');**

**DBMS\_OUTPUT.PUT\_LINE('4. Delete user');**

**DBMS\_OUTPUT.PUT\_LINE('5. University financials');**

**DBMS\_OUTPUT.PUT\_LINE('6. Update user');**

**choice := &choice;**

**CASE choice**

**WHEN 1 THEN**

**DECLARE**

**start\_date DATE;**

**end\_date DATE;**

**total\_outNUMBER(10,2);**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Enter start date (yyyy-mm-dd):');**

**start\_date := TO\_DATE('&start\_date', 'yyyy-mm-dd');**

**DBMS\_OUTPUT.PUT\_LINE('Enter end date (yyyy-mm-dd):');**

**end\_date := TO\_DATE('&end\_date', 'yyyy-mm-dd');**

**calculate\_expenditure(start\_date, end\_date, total\_out);**

**DBMS\_OUTPUT.PUT\_LINE('Total expenditure: ' || total\_out);**

**WHEN 2 THEN**

**DECLARE**

**transaction\_date DATE;**

**transaction\_amount NUMBER;**

**transaction\_type VARCHAR2(50);**

**transaction\_account VARCHAR2(50);**

**transaction\_user VARCHAR2(50);**

**description VARCHAR2(200);**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Enter transaction date (yyyy-mm-dd):');**

**transaction\_date := TO\_DATE('&transaction\_date', 'yyyy-mm-dd');**

**DBMS\_OUTPUT.PUT\_LINE('Enter transaction amount:');**

**transaction\_amount := &transaction\_amount;**

**DBMS\_OUTPUT.PUT\_LINE('Enter transaction type:');**

**transaction\_type := '&transaction\_type';**

**DBMS\_OUTPUT.PUT\_LINE('Enter transaction account:');**

**transaction\_account := '&transaction\_account';**

**DBMS\_OUTPUT.PUT\_LINE('Enter transaction user:');**

**transaction\_user := '&transaction\_user';**

**DBMS\_OUTPUT.PUT\_LINE('Enter transaction description:');**

**description := '&description';**

**insert\_transaction(transaction\_date, transaction\_amount, transaction\_type,**

**transaction\_account, transaction\_user, description);**

**END;**

**WHEN 3 THEN**

**DECLARE**

**user\_id NUMBER;**

**user\_name VARCHAR2(50);**

**user\_email VARCHAR2(100);**

**user\_account\_number NUMBER;**

**user\_scholarship\_amount NUMBER;**

**user\_authorisation\_type VARCHAR2(50);**

**user\_gpa NUMBER;**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Enter user ID:');**

**user\_id := &user\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Enter user name:');**

**user\_name := '&user\_name';**

**DBMS\_OUTPUT.PUT\_LINE('Enter user email:');**

**user\_email := '&user\_email';**

**DBMS\_OUTPUT.PUT\_LINE('Enter user account number:');**

**user\_account\_number := &user\_account\_number;**

**DBMS\_OUTPUT.PUT\_LINE('Enter user scholarship amount:');**

**user\_scholarship\_amount := &user\_scholarship\_amount;**

**DBMS\_OUTPUT.PUT\_LINE('Enter user authorisation type:');**

**user\_authorisation\_type := '&user\_authorisation\_type';**

**DBMS\_OUTPUT.PUT\_LINE('Enter user GPA:');**

**user\_gpa := &user\_gpa;**

**add\_user(user\_id, user\_name, user\_email, user\_account\_number,**

**user\_scholarship\_amount, user\_authorisation\_type, user\_gpa);**

**END;**

**WHEN 4 THEN**

**DECLARE**

**user\_idNUMBER(9);**

**DBMS\_OUTPUT.PUT\_LINE('Enter user ID:');**

**user\_id := &user\_id;**

**delete\_user(user\_id);**

**WHEN 5 THEN**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE(university\_financials());**

**END;**

**WHEN 6 THEN**

**DECLARE**

**v\_user\_id NUMBER;**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Enter User ID:');**

**v\_user\_id := '&user\_id';**

**update\_user\_info(v\_user\_id);**

**END;**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('Invalid input. Please enter a number between 1 and 6.');**

**END CASE;**

**END;**

THANKYOU!