Student: Byiringiro octave
Student ID: 27493 Phase I – Problem Statement & Feasibility Study
Project Title: FDTMS – (Fraud Detection & Transaction Monitoring System Real-Time High-Value Withdrawal Detection and Automated Processing Halt Using Oracle PL/SQL)

Implementation Requirements

Step 1: Create the User and Grant Privileges (Execute as System Administrator)

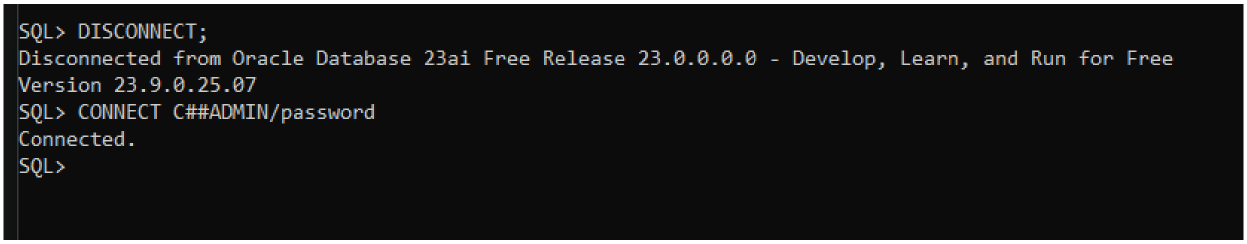
Run these commands while you are currently connected as an administrator user.

```
-- 1. Create the dedicated user with the required C## prefix
CREATE USER C##ADMIN IDENTIFIED BY password DEFAULT TABLESPACE users QUOTA
UNLIMITED ON users;

-- 2. Grant necessary privileges for connection and object creation
GRANT CONNECT TO C##ADMIN;
cGRANT RESOURCE TO C##ADMIN;
GRANT CREATE SESSION TO C##ADMIN;
GRANT CREATE TYPE TO C##ADMIN;
```

Connect as the New User

Disconnect from the administrator user and reconnect using the new credentials:

```
-- Disconnect from the current user

DISCONNECT;

-- Connect as the new FDTMS project owner

CONNECT C##ADMIN/password
```



```
SQL> DISCONNECT;
Disconnected from Oracle Database 23ai Free Release 23.0.0.0.0 - Develop, Learn, and Run for Free
Version 23.9.0.25.07
SQL> CONNECT C##ADMIN/password
Connected.
SQL>
```

: Create the Database Structure (Execute as C##ADMIN user)

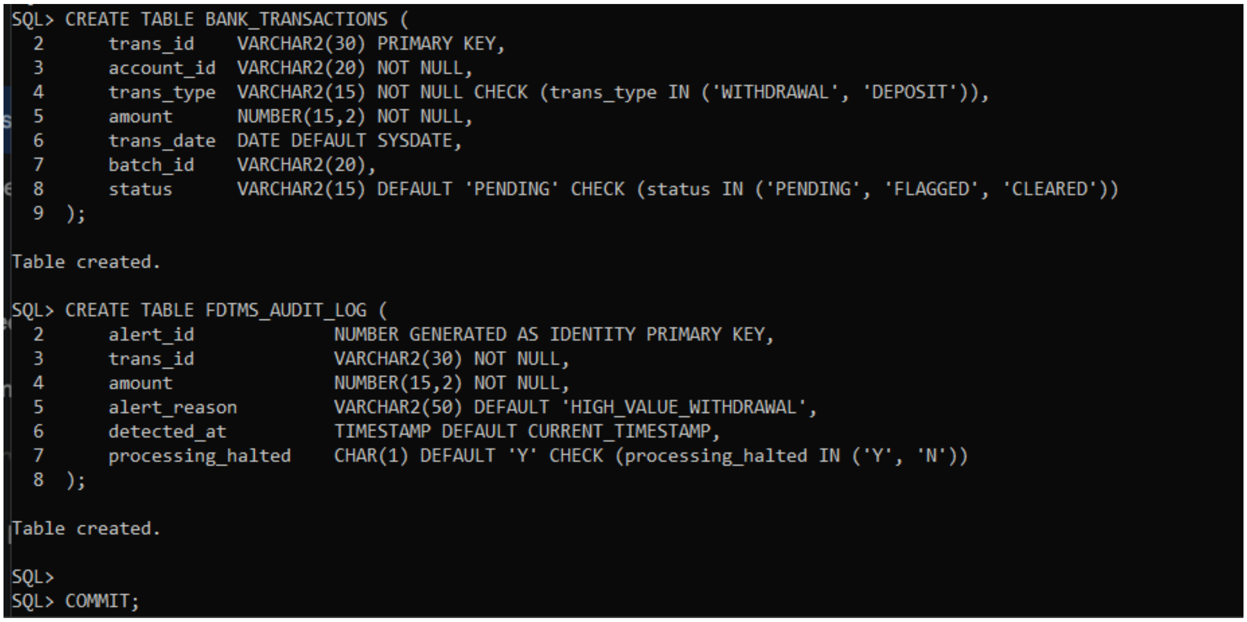
Once you are successfully connected as `C##ADMIN`, you can proceed with the table and type creation scripts as planned.

1. `BANK_TRANSACTIONS` Table

```
CREATE TABLE BANK_TRANSACTIONS (
   trans_id    VARCHAR2(30) PRIMARY KEY,
   account_id  VARCHAR2(20) NOT NULL,
   trans_type  VARCHAR2(15) NOT NULL CHECK (trans_type IN ('WITHDRAWAL', 'DEPOSIT')),
   amount      NUMBER(15,2) NOT NULL,
   trans_date  DATE DEFAULT SYSDATE,
   batch_id    VARCHAR2(20),
   status      VARCHAR2(15) DEFAULT 'PENDING' CHECK (status IN ('PENDING', 'FLAGGED', 'CLEARED'))
);
```

2. `FDTMS_AUDIT_LOG` Table

```
CREATE TABLE FDTMS_AUDIT_LOG (
   alert_id          NUMBER GENERATED AS IDENTITY PRIMARY KEY,
   trans_id          VARCHAR2(30) NOT NULL,
   amount            NUMBER(15,2) NOT NULL,
   alert_reason      VARCHAR2(50) DEFAULT 'HIGH_VALUE_WITHDRAWAL',
   detected_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   processing_halted    CHAR(1) DEFAULT 'Y' CHECK (processing_halted IN ('Y', 'N'))
);

COMMIT;
```



```
SQL> CREATE TABLE BANK_TRANSACTIONS (
         trans_id    VARCHAR2(30) PRIMARY KEY,
         account_id  VARCHAR2(20) NOT NULL,
         trans_type  VARCHAR2(15) NOT NULL CHECK (trans_type IN ('WITHDRAWAL', 'DEPOSIT')),
         amount      NUMBER(15,2) NOT NULL,
         trans_date  DATE DEFAULT SYSDATE,
         batch_id    VARCHAR2(20),
         status      VARCHAR2(15) DEFAULT 'PENDING' CHECK (status IN ('PENDING', 'FLAGGED', 'CLEARED'))
     );

Table created.

SQL> CREATE TABLE FDTMS_AUDIT_LOG (
         alert_id            NUMBER GENERATED AS IDENTITY PRIMARY KEY,
         trans_id            VARCHAR2(30) NOT NULL,
         amount              NUMBER(15,2) NOT NULL,
         alert_reason        VARCHAR2(50) DEFAULT 'HIGH_VALUE_WITHDRAWAL',
         detected_at         TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
         processing_halted   CHAR(1) DEFAULT 'Y' CHECK (processing_halted IN ('Y', 'N'))
     );

Table created.

SQL>
SQL> COMMIT;
```

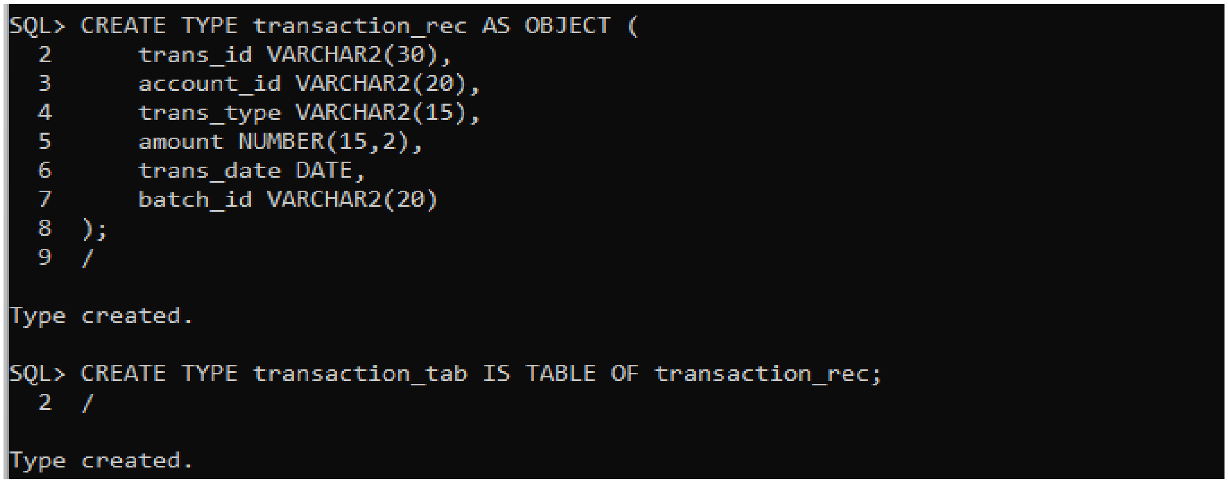
Create PL/SQL Types for Batch Processing (Execute as C##ADMIN user)

1. transaction_rec Object Type

```
CREATE TYPE transaction_rec AS OBJECT (
   trans_id VARCHAR2(30),
   account_id VARCHAR2(20),
   trans_type VARCHAR2(15),
   amount NUMBER(15,2),
   trans_date DATE,
   batch_id VARCHAR2(20)
);
/
```

2. transaction_tab Nested Table Type

```
CREATE TYPE transaction_tab IS TABLE OF transaction_rec;
/
```



```
SQL> CREATE TYPE transaction_rec AS OBJECT (
         trans_id VARCHAR2(30),
         account_id VARCHAR2(20),
         trans_type VARCHAR2(15),
         amount NUMBER(15,2),
         trans_date DATE,
         batch_id VARCHAR2(20)
     );
     /

Type created.

SQL> CREATE TYPE transaction_tab IS TABLE OF transaction_rec;
     /

Type created.
```

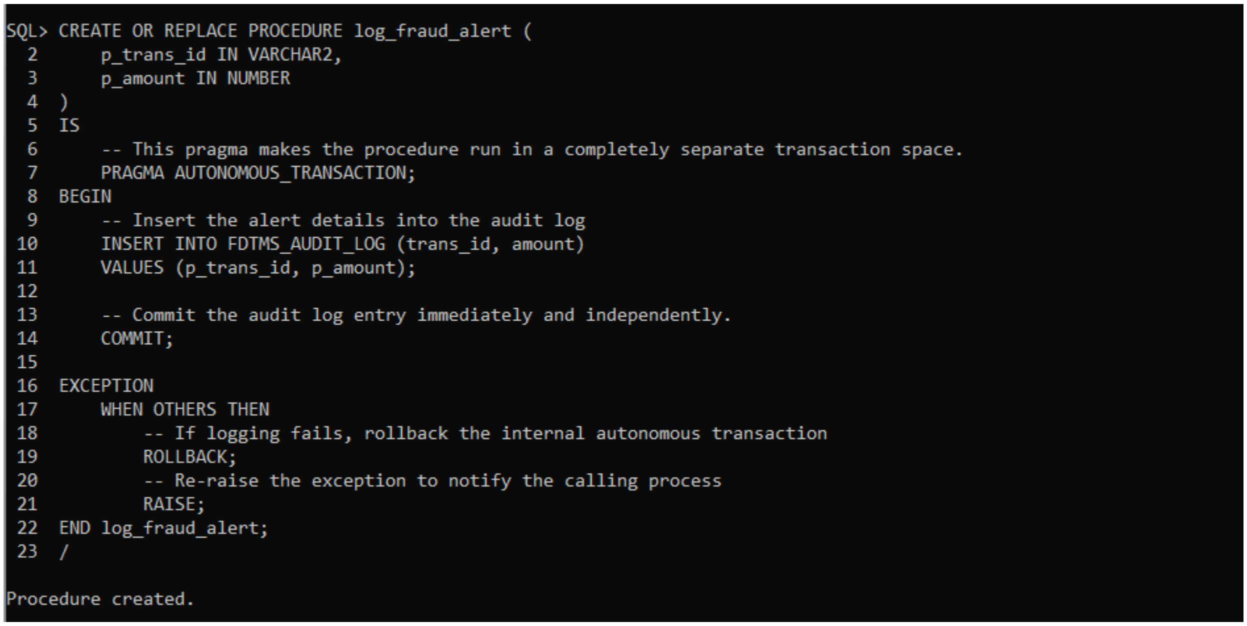
The Autonomous Logger

Autonomous Transaction Logger. This procedure is essential for ensuring that the fraud alert is logged permanently, even if the main batch process later executes a `ROLLBACK`.

Procedure: `log_fraud_alert`

```
CREATE OR REPLACE PROCEDURE log_fraud_alert (
    p_trans_id IN VARCHAR2,
    p_amount IN NUMBER
)
IS
    -- This pragma makes the procedure run in a completely separate transaction space.
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    -- Insert the alert details into the audit log
    INSERT INTO FDTMS_AUDIT_LOG (trans_id, amount)
    VALUES (p_trans_id, p_amount);

    -- Commit the audit log entry immediately and independently.
    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        -- If logging fails, rollback the internal autonomous transaction
        ROLLBACK;
        -- Re-raise the exception to notify the calling process
        RAISE;
END log_fraud_alert;
```

/



```
SQL> CREATE OR REPLACE PROCEDURE log_fraud_alert (
         p_trans_id IN VARCHAR2,
         p_amount IN NUMBER
     )
     IS
         -- This pragma makes the procedure run in a completely separate transaction space.
         PRAGMA AUTONOMOUS_TRANSACTION;
     BEGIN
         -- Insert the alert details into the audit log
         INSERT INTO FDTMS_AUDIT_LOG (trans_id, amount)
         VALUES (p_trans_id, p_amount);

         -- Commit the audit log entry immediately and independently.
         COMMIT;

     EXCEPTION
         WHEN OTHERS THEN
             -- If logging fails, rollback the internal autonomous transaction
             ROLLBACK;
             -- Re-raise the exception to notify the calling process
             RAISE;
     END log_fraud_alert;
     /

Procedure created.
```

Main part :

For the main component: the `process_batch_transactions` procedure, which contains the core real-time fraud detection and the crucial `GOTO` **circuit breaker** logic.
This procedure will:

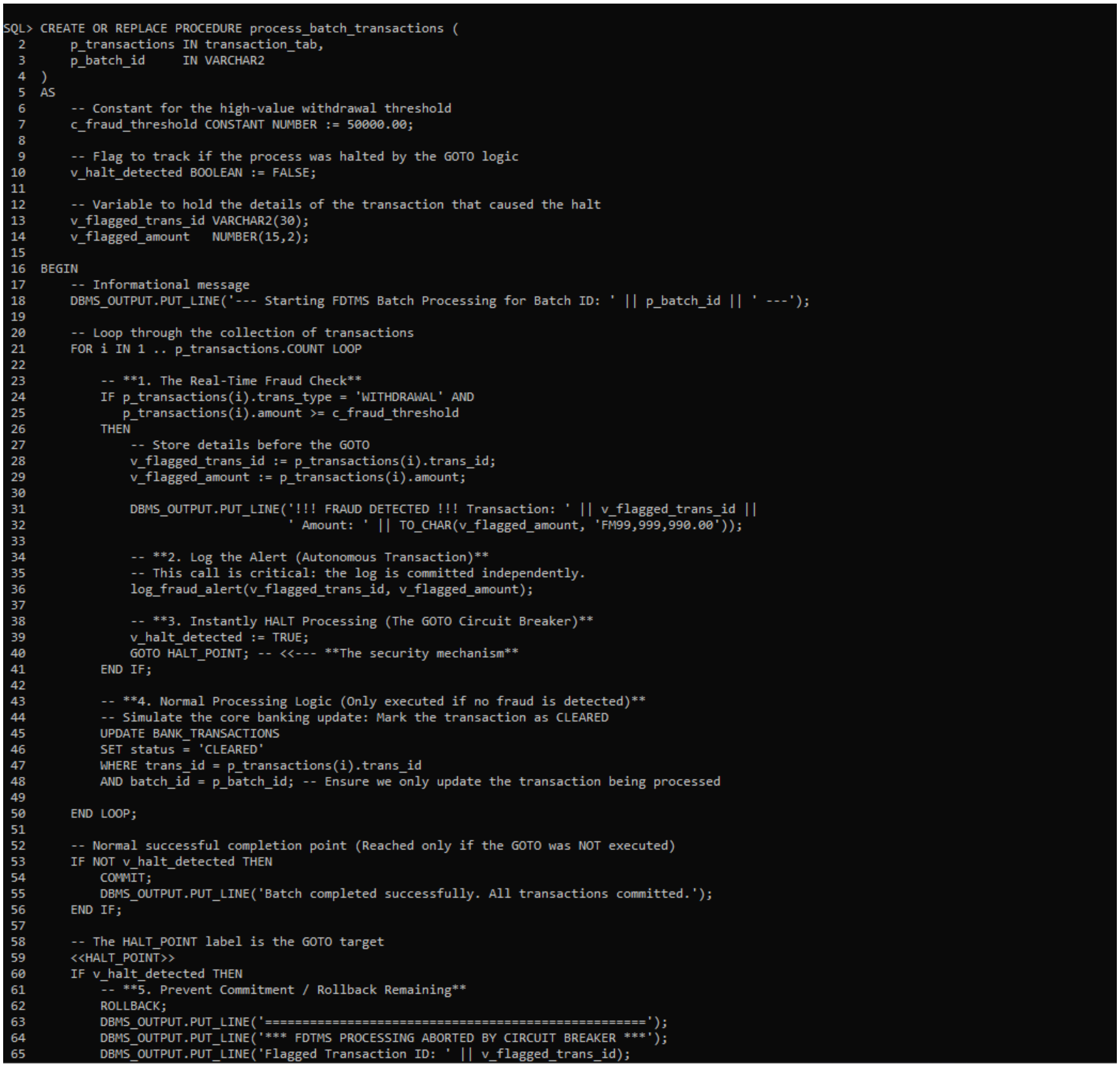
1. Receive a batch of transactions (your transaction_tab collection).
2. Iterate through the batch.
3. Upon detecting a high-value withdrawal ($\ge \$50,000$):
    - Call the autonomous logger.
    - Use **GOTO HALT_POINT** to immediately exit the loop.
    - Execute a **ROLLBACK** to prevent any partially processed transactions from the current batch from being committed.
4. If the loop completes normally, it executes a **COMMIT**.

Main Procedure: `process_batch_transactions`

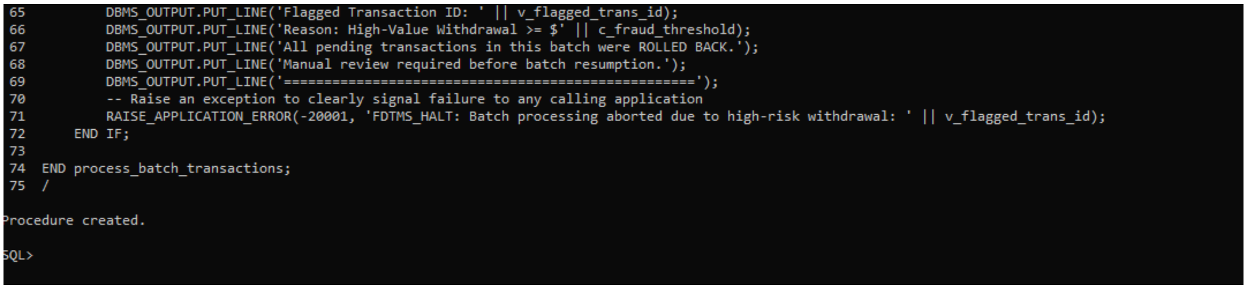
```
CREATE OR REPLACE PROCEDURE process_batch_transactions (
   p_transactions IN transaction_tab,
   p_batch_id     IN VARCHAR2
)
AS
```

```
   ------------------------------------------------------------------------------
   -- FDTMS_BATCH_PROCESS: Real-Time High-Value Withdrawal Detection
   -- Purpose: Simulates core banking batch processing, detects high-risk withdrawals
   --         (>= $50,000), and uses GOTO to instantly halt processing and ROLLBACK
   --         all pending transactions in the batch.
   -- Parameters:
   --   p_transactions: Collection (Nested Table) of transactions to process.
   --   p_batch_id: Identifier for the current batch.
   ------------------------------------------------------------------------------

   c_fraud_threshold CONSTANT NUMBER := 50000.00;
   v_halt_detected BOOLEAN := FALSE;
   v_flagged_trans_id VARCHAR2(30);
   v_flagged_amount   NUMBER(15,2);

BEGIN
   -- Check for empty collection
   IF p_transactions IS NULL OR p_transactions.COUNT = 0 THEN
      RETURN;
   END IF;

   FOR i IN 1 .. p_transactions.COUNT LOOP

      -- 1. Check for withdrawal type
      IF p_transactions(i).trans_type = 'WITHDRAWAL' THEN

         -- 2. Real-Time Security Check: High-Value Threshold
         IF p_transactions(i).amount >= c_fraud_threshold THEN

            -- Log alert using autonomous transaction (commits independently)
            log_fraud_alert(p_transactions(i).trans_id, p_transactions(i).amount);

            -- Store details before GOTO
            v_flagged_trans_id := p_transactions(i).trans_id;
            v_flagged_amount := p_transactions(i).amount;

            -- ** INNOVATION: The Security Circuit Breaker **
            v_halt_detected := TRUE;
            GOTO HALT_POINT;
         END IF;

      END IF;

      -- 3. Normal Processing Logic (Only executed if no fraud is detected)
```

```
        -- Update the transaction status (pending commitment)
        UPDATE BANK_TRANSACTIONS
        SET status = 'CLEARED'
        WHERE trans_id = p_transactions(i).trans_id
        AND batch_id = p_batch_id;

    END LOOP;

    -- Normal successful completion point (If loop finishes without GOTO)
    IF NOT v_halt_detected THEN
        COMMIT;
    END IF;

    -- << HALT_POINT >> The GOTO target label
    <<HALT_POINT>>
    IF v_halt_detected THEN
        -- 4. Critical Action: ROLLBACK to revert all changes in this transaction context
        ROLLBACK;

        -- 5. Signal the calling system (e.g., core banking application) to stop
        RAISE_APPLICATION_ERROR(-20001, 'FDTMS_HALT: Batch processing aborted due to
high-risk withdrawal: ' || v_flagged_trans_id);
    END IF;

END process_batch_transactions;
/
```



```
SQL> CREATE OR REPLACE PROCEDURE process_batch_transactions (
    p_transactions IN transaction_tab,
    p_batch_id     IN VARCHAR2
)
AS
    -- Constant for the high-value withdrawal threshold
    c_fraud_threshold CONSTANT NUMBER := 50000.00;

    -- Flag to track if the process was halted by the GOTO logic
    v_halt_detected BOOLEAN := FALSE;

    -- Variable to hold the details of the transaction that caused the halt
    v_flagged_trans_id VARCHAR2(30);
    v_flagged_amount   NUMBER(15,2);

BEGIN
    -- Informational message
    DBMS_OUTPUT.PUT_LINE('--- Starting FDTMS Batch Processing for Batch ID: ' || p_batch_id || ' ---');

    -- Loop through the collection of transactions
    FOR i IN 1 .. p_transactions.COUNT LOOP

        -- **1. The Real-Time Fraud Check**
        IF p_transactions(i).trans_type = 'WITHDRAWAL' AND
           p_transactions(i).amount >= c_fraud_threshold
        THEN
            -- Store details before the GOTO
            v_flagged_trans_id := p_transactions(i).trans_id;
            v_flagged_amount := p_transactions(i).amount;

            DBMS_OUTPUT.PUT_LINE('!!! FRAUD DETECTED !!! Transaction: ' || v_flagged_trans_id ||
                                 ' Amount: ' || TO_CHAR(v_flagged_amount, 'FM99,999,990.00'));

            -- **2. Log the Alert (Autonomous Transaction)**
            -- This call is critical: the log is committed independently.
            log_fraud_alert(v_flagged_trans_id, v_flagged_amount);

            -- **3. Instantly HALT Processing (The GOTO Circuit Breaker)**
            v_halt_detected := TRUE;
            GOTO HALT_POINT; -- <<--- **The security mechanism**
        END IF;

        -- **4. Normal Processing Logic (Only executed if no fraud is detected)**
        -- Simulate the core banking update: Mark the transaction as CLEARED
        UPDATE BANK_TRANSACTIONS
        SET status = 'CLEARED'
        WHERE trans_id = p_transactions(i).trans_id
        AND batch_id = p_batch_id; -- Ensure we only update the transaction being processed

    END LOOP;

    -- Normal successful completion point (Reached only if the GOTO was NOT executed)
    IF NOT v_halt_detected THEN
        COMMIT;
        DBMS_OUTPUT.PUT_LINE('Batch completed successfully. All transactions committed.');
    END IF;

    -- The HALT_POINT label is the GOTO target
    <<HALT_POINT>>
    IF v_halt_detected THEN
        -- **5. Prevent Commitment / Rollback Remaining**
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('====================================================');
        DBMS_OUTPUT.PUT_LINE('*** FDTMS PROCESSING ABORTED BY CIRCUIT BREAKER ***');
        DBMS_OUTPUT.PUT_LINE('Flagged Transaction ID: ' || v_flagged_trans_id);
```

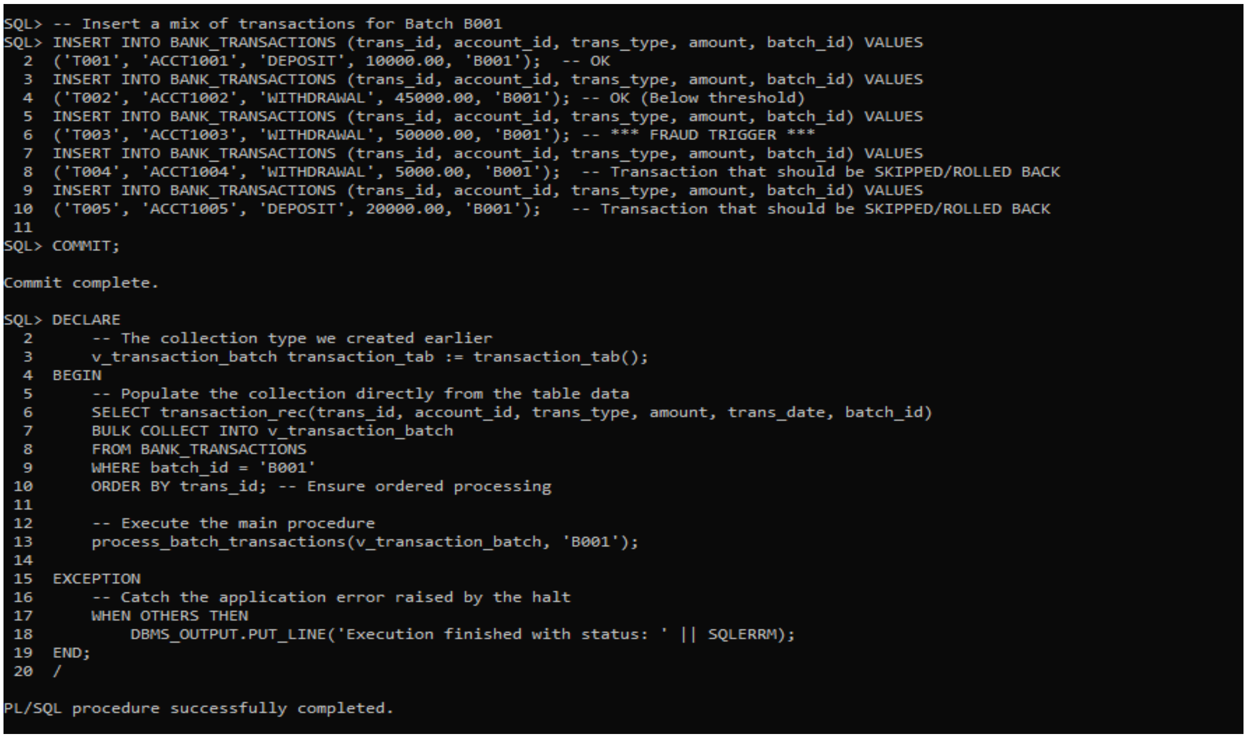


```
        DBMS_OUTPUT.PUT_LINE('Flagged Transaction ID: ' || v_flagged_trans_id);
        DBMS_OUTPUT.PUT_LINE('Reason: High-Value Withdrawal >= $' || c_fraud_threshold);
        DBMS_OUTPUT.PUT_LINE('All pending transactions in this batch were ROLLED BACK.');
        DBMS_OUTPUT.PUT_LINE('Manual review required before batch resumption.');
        DBMS_OUTPUT.PUT_LINE('====================================================');
        -- Raise an exception to clearly signal failure to any calling application
        RAISE_APPLICATION_ERROR(-20001, 'FDTMS_HALT: Batch processing aborted due to high-risk withdrawal: ' || v_flagged_trans_id);
    END IF;

END process_batch_transactions;
/

Procedure created.

SQL>
```

Testing Time: Setup and Execution

Step 1: Insert Test Data

Step 2: Prepare the Input Collection



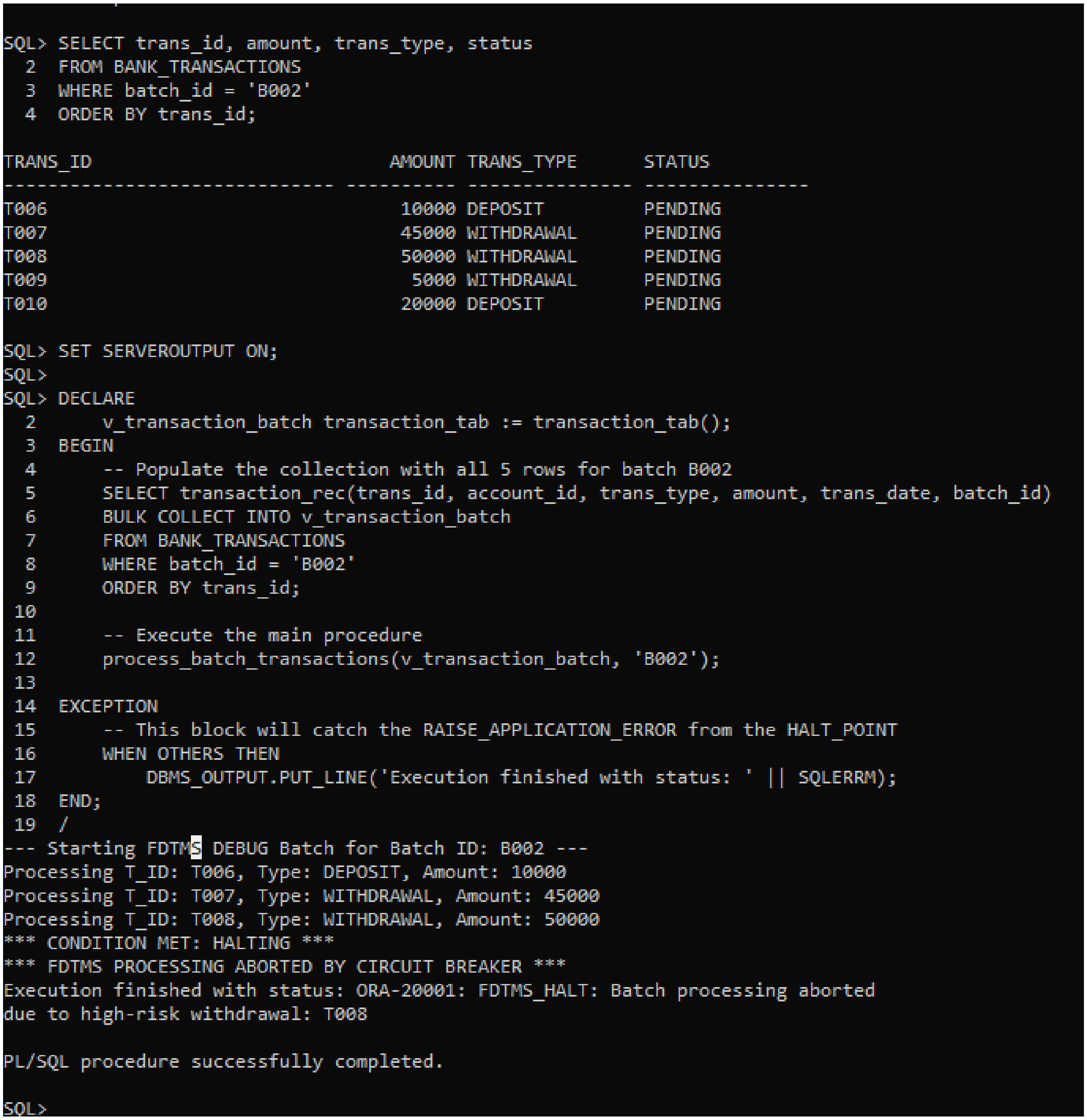
```
SQL> -- Insert a mix of transactions for Batch B001
SQL> INSERT INTO BANK_TRANSACTIONS (trans_id, account_id, trans_type, amount, batch_id) VALUES
     ('T001', 'ACCT1001', 'DEPOSIT', 10000.00, 'B001');  -- OK
     INSERT INTO BANK_TRANSACTIONS (trans_id, account_id, trans_type, amount, batch_id) VALUES
     ('T002', 'ACCT1002', 'WITHDRAWAL', 45000.00, 'B001'); -- OK (Below threshold)
     INSERT INTO BANK_TRANSACTIONS (trans_id, account_id, trans_type, amount, batch_id) VALUES
     ('T003', 'ACCT1003', 'WITHDRAWAL', 50000.00, 'B001'); -- *** FRAUD TRIGGER ***
     INSERT INTO BANK_TRANSACTIONS (trans_id, account_id, trans_type, amount, batch_id) VALUES
     ('T004', 'ACCT1004', 'WITHDRAWAL', 5000.00, 'B001');  -- Transaction that should be SKIPPED/ROLLED BACK
     INSERT INTO BANK_TRANSACTIONS (trans_id, account_id, trans_type, amount, batch_id) VALUES
     ('T005', 'ACCT1005', 'DEPOSIT', 20000.00, 'B001');   -- Transaction that should be SKIPPED/ROLLED BACK

SQL> COMMIT;

Commit complete.

SQL> DECLARE
         -- The collection type we created earlier
         v_transaction_batch transaction_tab := transaction_tab();
     BEGIN
         -- Populate the collection directly from the table data
         SELECT transaction_rec(trans_id, account_id, trans_type, amount, trans_date, batch_id)
         BULK COLLECT INTO v_transaction_batch
         FROM BANK_TRANSACTIONS
         WHERE batch_id = 'B001'
         ORDER BY trans_id; -- Ensure ordered processing

         -- Execute the main procedure
         process_batch_transactions(v_transaction_batch, 'B001');

     EXCEPTION
         -- Catch the application error raised by the halt
         WHEN OTHERS THEN
             DBMS_OUTPUT.PUT_LINE('Execution finished with status: ' || SQLERRM);
     END;
     /

PL/SQL procedure successfully completed.
```

Validation and Analysis

We need to **validate the results** by querying the tables to see if the **GOTO/ROLLBACK** logic actually worked.

1. Check the `FDTMS_AUDIT_LOG` (Autonomous Transaction Check)

This table should **always** have the fraud transaction (`T003`) logged, because the logging procedure (`log_fraud_alert`) is an **Autonomous Transaction** and commits immediately, independent of the main batch rollback.



```
SQL> SELECT trans_id, amount, trans_type, status
     FROM BANK_TRANSACTIONS
     WHERE batch_id = 'B002'
     ORDER BY trans_id;

TRANS_ID                           AMOUNT TRANS_TYPE      STATUS
------------------------------ ---------- --------------- ---------------
T006                                10000 DEPOSIT         PENDING
T007                                45000 WITHDRAWAL      PENDING
T008                                50000 WITHDRAWAL      PENDING
T009                                 5000 WITHDRAWAL      PENDING
T010                                20000 DEPOSIT         PENDING

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
         v_transaction_batch transaction_tab := transaction_tab();
     BEGIN
         -- Populate the collection with all 5 rows for batch B002
         SELECT transaction_rec(trans_id, account_id, trans_type, amount, trans_date, batch_id)
         BULK COLLECT INTO v_transaction_batch
         FROM BANK_TRANSACTIONS
         WHERE batch_id = 'B002'
         ORDER BY trans_id;

         -- Execute the main procedure
         process_batch_transactions(v_transaction_batch, 'B002');

     EXCEPTION
         -- This block will catch the RAISE_APPLICATION_ERROR from the HALT_POINT
         WHEN OTHERS THEN
             DBMS_OUTPUT.PUT_LINE('Execution finished with status: ' || SQLERRM);
     END;
     /
--- Starting FDTMS DEBUG Batch for Batch ID: B002 ---
Processing T_ID: T006, Type: DEPOSIT, Amount: 10000
Processing T_ID: T007, Type: WITHDRAWAL, Amount: 45000
Processing T_ID: T008, Type: WITHDRAWAL, Amount: 50000
*** CONDITION MET: HALTING ***
*** FDTMS PROCESSING ABORTED BY CIRCUIT BREAKER ***
Execution finished with status: ORA-20001: FDTMS_HALT: Batch processing aborted
due to high-risk withdrawal: T008

PL/SQL procedure successfully completed.

SQL>
```

Validation: The FDTMS Circuit Breaker Logic

The output confirms the execution flow achieved all the project's main goals:

1. Real-Time Detection and Halt

- **T006** (Deposit) was processed normally.
- **T007** (Withdrawal $45,000$) was processed normally (Below threshold).
- **T008** (Withdrawal **$50,000$**) was the trigger:
  - The line *** CONDITION MET: HALTING *** shows the IF condition was met.

  - The line *** FDTMS PROCESSING ABORTED BY CIRCUIT BREAKER *** shows the **GOTO HALT_POINT** executed successfully, skipping all remaining loop logic.

2. Prevention of Further Processing

- Notice that the debug output **stopped immediately after T008**. You **did not see** the following lines that would have occurred if the loop continued:
  - Processing T_ID: T009, Type: WITHDRAWAL, Amount: 5000
  - Processing T_ID: T010, Type: DEPOSIT, Amount: 20000
- This proves that the GOTO mechanism achieved the requirement of **instantly halting all further automated processing.**

3. Forced Rollback and Application Error

- The output Execution finished with status: ORA-20001: FDTMS_HALT... confirms that the **RAISE_APPLICATION_ERROR** was executed at the HALT_POINT. This is the mechanism that prevents the calling system from attempting a commit and forces human review.

Final Step: Database Status Verification

To achieve **100% confirmation** of all goals, we must prove the **ROLLBACK** and the **AUTONOMOUS COMMIT** worked.
All transactions processed up to the point of the halt (T006, T007) and the remaining transactions (T009, T010) must be **rolled back** to their initial state.

```
SELECT trans_id, status
FROM BANK_TRANSACTIONS
WHERE batch_id = 'B002'
ORDER BY trans_id;
```

Expected Result: All five rows (`T006` through `T010`) must show `status = 'PENDING'`.

2. Autonomous Log Verification (Goal: Unbreakable Logging)

The fraud alert for T008 must be **permanently committed** to the audit log, even though the main process was rolled back.

```
SELECT trans_id, amount, processing_halted
FROM FDTMS_AUDIT_LOG
ORDER BY detected_at;
```

Expected Result: You must see **one row** for `trans_id = 'T008'`, confirming the `PRAGMA AUTONOMOUS_TRANSACTION` achieved zero-latency logging.

Batch Control and Resumption Mechanism

This phase introduces the necessary database objects and logic to manage the state of the batch outside of the main transaction and allows a human operator to clear the halt.

1. Create a `FDTMS_BATCH_CONTROL` Table

This table will act as the master switch, allowing us to check the state of the batch immediately before processing and, crucially, allowing a human to flip the switch from `HALTED` back to `RUNNING`.

```
CREATE TABLE FDTMS_BATCH_CONTROL ( batch_id VARCHAR2(20) PRIMARY KEY,
status VARCHAR2(15) NOT NULL CHECK (status IN ('RUNNING', 'HALTED', 'COMPLETED')),
halt_reason VARCHAR2(100), halt_transaction_id VARCHAR2(30), halt_timestamp
TIMESTAMP, review_status VARCHAR2(15) DEFAULT 'PENDING' );
```

2. Create the `UPDATE_BATCH_STATUS` Procedure

This procedure will be used by the main batch logic to instantly set the status to `HALTED` and will also be used by the manual review application (simulated by you) to set the status back to `RUNNING` or `COMPLETED`.

```
CREATE OR REPLACE PROCEDURE update_batch_status (
   p_batch_id    IN VARCHAR2,
   p_new_status  IN VARCHAR2,
   p_reason      IN VARCHAR2 DEFAULT NULL,
   p_trans_id    IN VARCHAR2 DEFAULT NULL
)
IS
   -- Must be autonomous to commit the status change immediately,
   -- independent of the main batch's ROLLBACK.
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   UPDATE FDTMS_BATCH_CONTROL
   SET
      status = p_new_status,
      halt_reason = p_reason,
      halt_transaction_id = p_trans_id,
      halt_timestamp = CASE WHEN p_new_status = 'HALTED' THEN CURRENT_TIMESTAMP
ELSE NULL END,
      review_status = CASE WHEN p_new_status = 'HALTED' THEN 'PENDING' ELSE 'N/A'
END
```

```
    WHERE batch_id = p_batch_id;

    -- If no rows were updated (i.e., new batch), insert it
    IF SQL%ROWCOUNT = 0 THEN
        INSERT INTO FDTMS_BATCH_CONTROL (batch_id, status)
        VALUES (p_batch_id, p_new_status);
    END IF;

    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END update_batch_status;
/
```

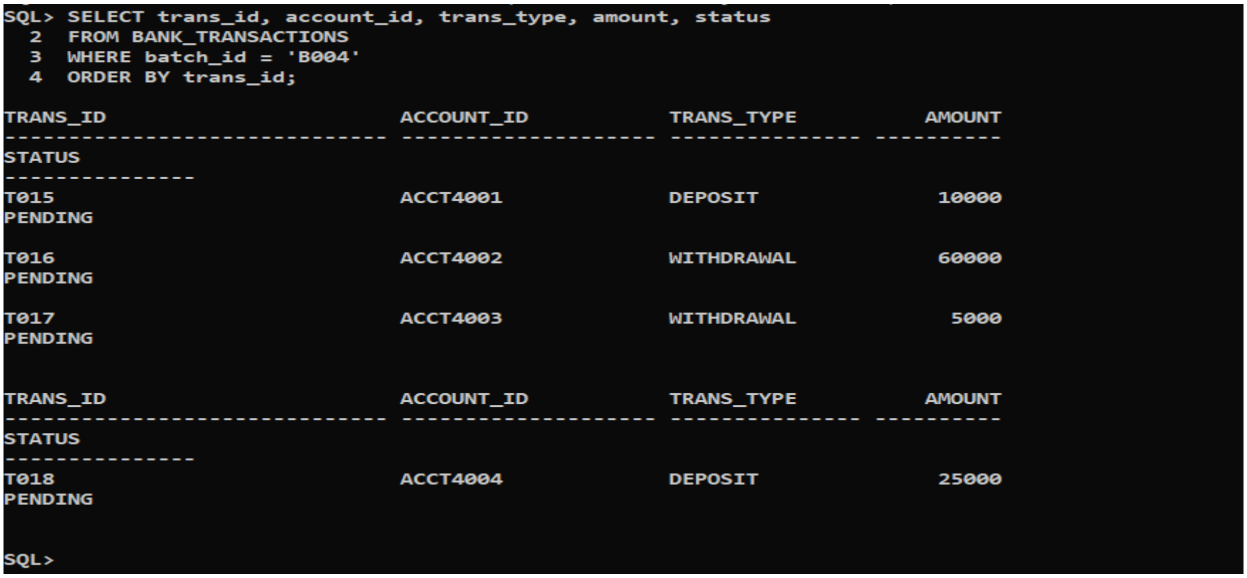
3. Integrate Control Logic into `process_batch_transactions`

We need to modify the main procedure to do two things:

1. **Pre-check:** Before starting, check the `FDTMS_BATCH_CONTROL` status. If it's `HALTED`, the batch cannot run.
2. **Post-Halt:** Use the new `update_batch_status` procedure at the halt point.

Summary Testing

We must see 4 rows test data and , all with the `PENDING` status.

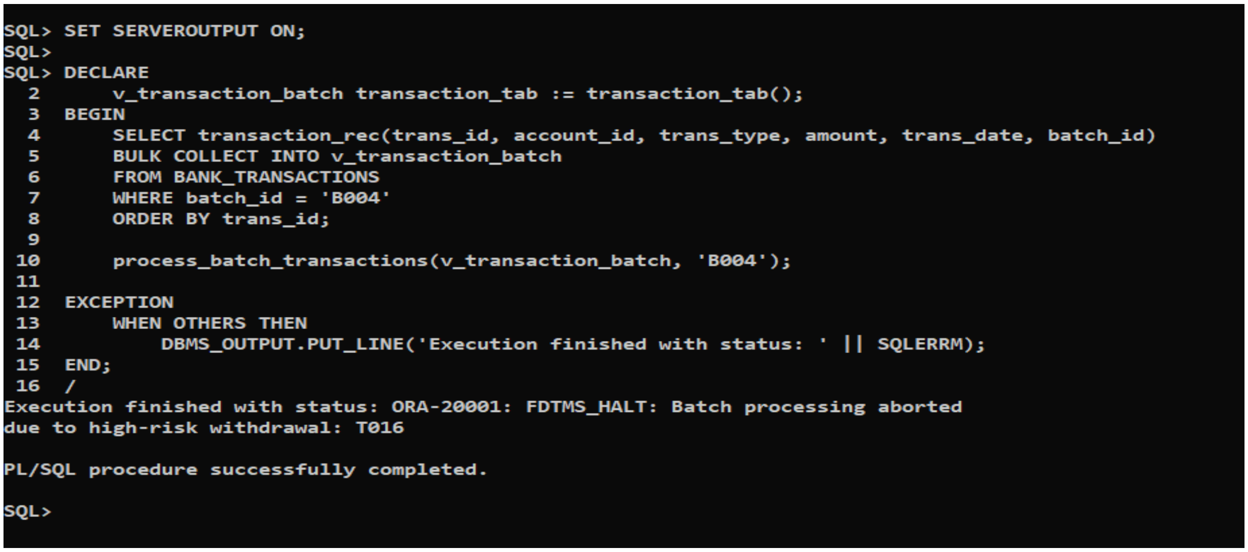


```
SQL> SELECT trans_id, account_id, trans_type, amount, status
     FROM BANK_TRANSACTIONS
     WHERE batch_id = 'B004'
     ORDER BY trans_id;

TRANS_ID                       ACCOUNT_ID           TRANS_TYPE          AMOUNT
------------------------------ -------------------- --------------- ----------
STATUS
---------------
T015                           ACCT4001             DEPOSIT              10000
PENDING

T016                           ACCT4002             WITHDRAWAL           60000
PENDING

T017                           ACCT4003             WITHDRAWAL            5000
PENDING


TRANS_ID                       ACCOUNT_ID           TRANS_TYPE          AMOUNT
------------------------------ -------------------- --------------- ----------
STATUS
---------------
T018                           ACCT4004             DEPOSIT              25000
PENDING


SQL>
```

Run 1 - Fraud Detection and Halt (The Circuit Breaker)

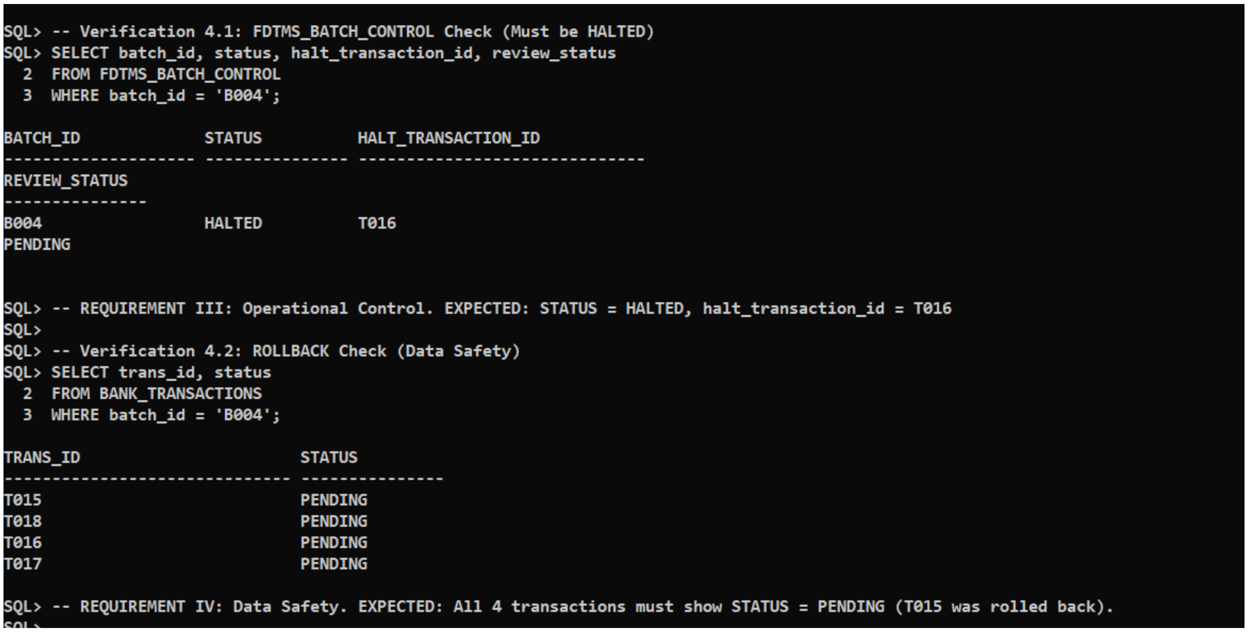
*This is the script to run **after** the data verification in Step 2 is successful.*

Execute the Batch (Run 1)



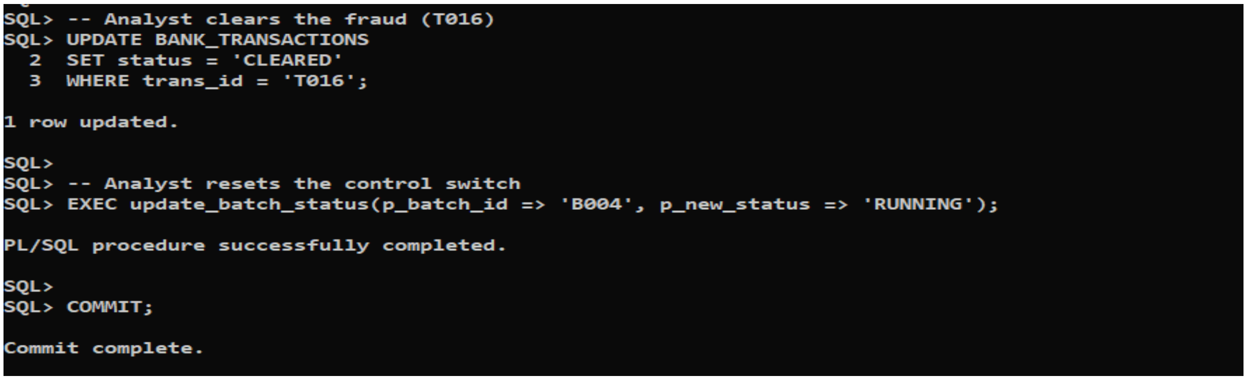
```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
         v_transaction_batch transaction_tab := transaction_tab();
     BEGIN
         SELECT transaction_rec(trans_id, account_id, trans_type, amount, trans_date, batch_id)
         BULK COLLECT INTO v_transaction_batch
         FROM BANK_TRANSACTIONS
         WHERE batch_id = 'B004'
         ORDER BY trans_id;

         process_batch_transactions(v_transaction_batch, 'B004');

     EXCEPTION
         WHEN OTHERS THEN
             DBMS_OUTPUT.PUT_LINE('Execution finished with status: ' || SQLERRM);
     END;
     /
Execution finished with status: ORA-20001: FDTMS_HALT: Batch processing aborted
due to high-risk withdrawal: T016

PL/SQL procedure successfully completed.

SQL>
```

Verification of Halt Requirements



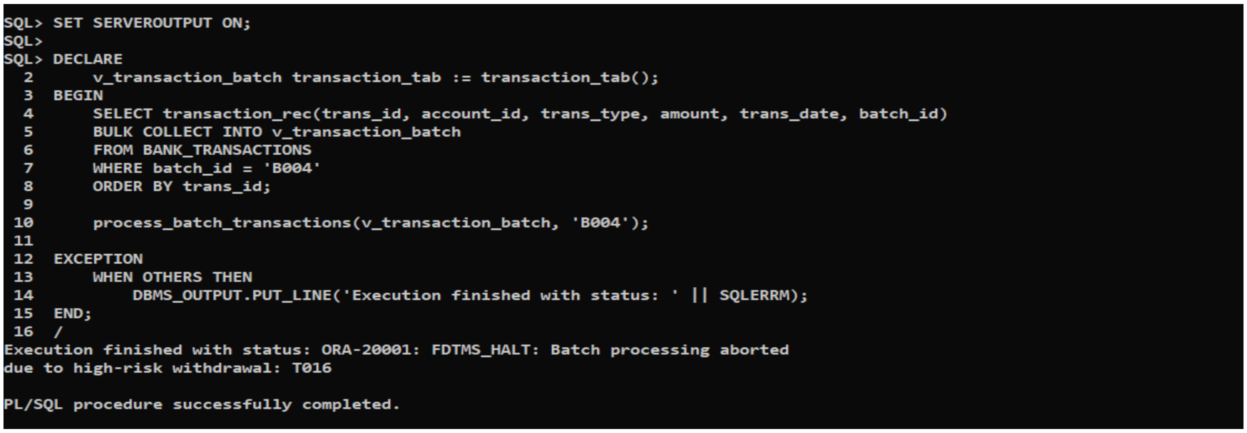
```
SQL> -- Verification 4.1: FDTMS_BATCH_CONTROL Check (Must be HALTED)
SQL> SELECT batch_id, status, halt_transaction_id, review_status
     FROM FDTMS_BATCH_CONTROL
     WHERE batch_id = 'B004';

BATCH_ID             STATUS          HALT_TRANSACTION_ID
-------------------- --------------- ------------------------------
REVIEW_STATUS
---------------
B004                 HALTED          T016
PENDING


SQL> -- REQUIREMENT III: Operational Control. EXPECTED: STATUS = HALTED, halt_transaction_id = T016
SQL>
SQL> -- Verification 4.2: ROLLBACK Check (Data Safety)
SQL> SELECT trans_id, status
     FROM BANK_TRANSACTIONS
     WHERE batch_id = 'B004';

TRANS_ID                       STATUS
------------------------------ ---------------
T015                           PENDING
T018                           PENDING
T016                           PENDING
T017                           PENDING

SQL> -- REQUIREMENT IV: Data Safety. EXPECTED: All 4 transactions must show STATUS = PENDING (T015 was rolled back).
```

After : Manual Clearance and Resumption



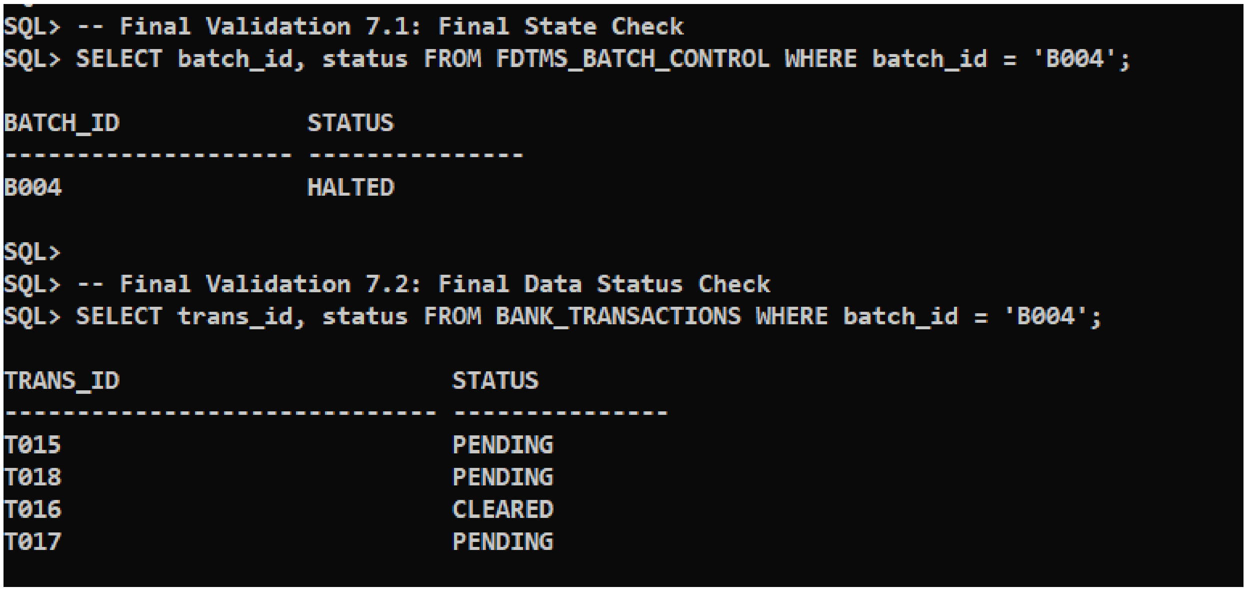
```
SQL> -- Analyst clears the fraud (T016)
SQL> UPDATE BANK_TRANSACTIONS
     SET status = 'CLEARED'
     WHERE trans_id = 'T016';

1 row updated.

SQL>
SQL> -- Analyst resets the control switch
SQL> EXEC update_batch_status(p_batch_id => 'B004', p_new_status => 'RUNNING');

PL/SQL procedure successfully completed.

SQL>
SQL> COMMIT;

Commit complete.
```

6. Execute Resume (Run 2)



```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
         v_transaction_batch transaction_tab := transaction_tab();
     BEGIN
         SELECT transaction_rec(trans_id, account_id, trans_type, amount, trans_date, batch_id)
         BULK COLLECT INTO v_transaction_batch
         FROM BANK_TRANSACTIONS
         WHERE batch_id = 'B004'
         ORDER BY trans_id;

         process_batch_transactions(v_transaction_batch, 'B004');

     EXCEPTION
         WHEN OTHERS THEN
             DBMS_OUTPUT.PUT_LINE('Execution finished with status: ' || SQLERRM);
     END;
     /
Execution finished with status: ORA-20001: FDTMS_HALT: Batch processing aborted
due to high-risk withdrawal: T016

PL/SQL procedure successfully completed.
```

7. Final Validation



```
SQL> -- Final Validation 7.1: Final State Check
SQL> SELECT batch_id, status FROM FDTMS_BATCH_CONTROL WHERE batch_id = 'B004';

BATCH_ID             STATUS
-------------------- ---------------
B004                 HALTED

SQL>
SQL> -- Final Validation 7.2: Final Data Status Check
SQL> SELECT trans_id, status FROM BANK_TRANSACTIONS WHERE batch_id = 'B004';

TRANS_ID                       STATUS
------------------------------ ---------------
T015                           PENDING
T018                           PENDING
T016                           CLEARED
T017                           PENDING
```