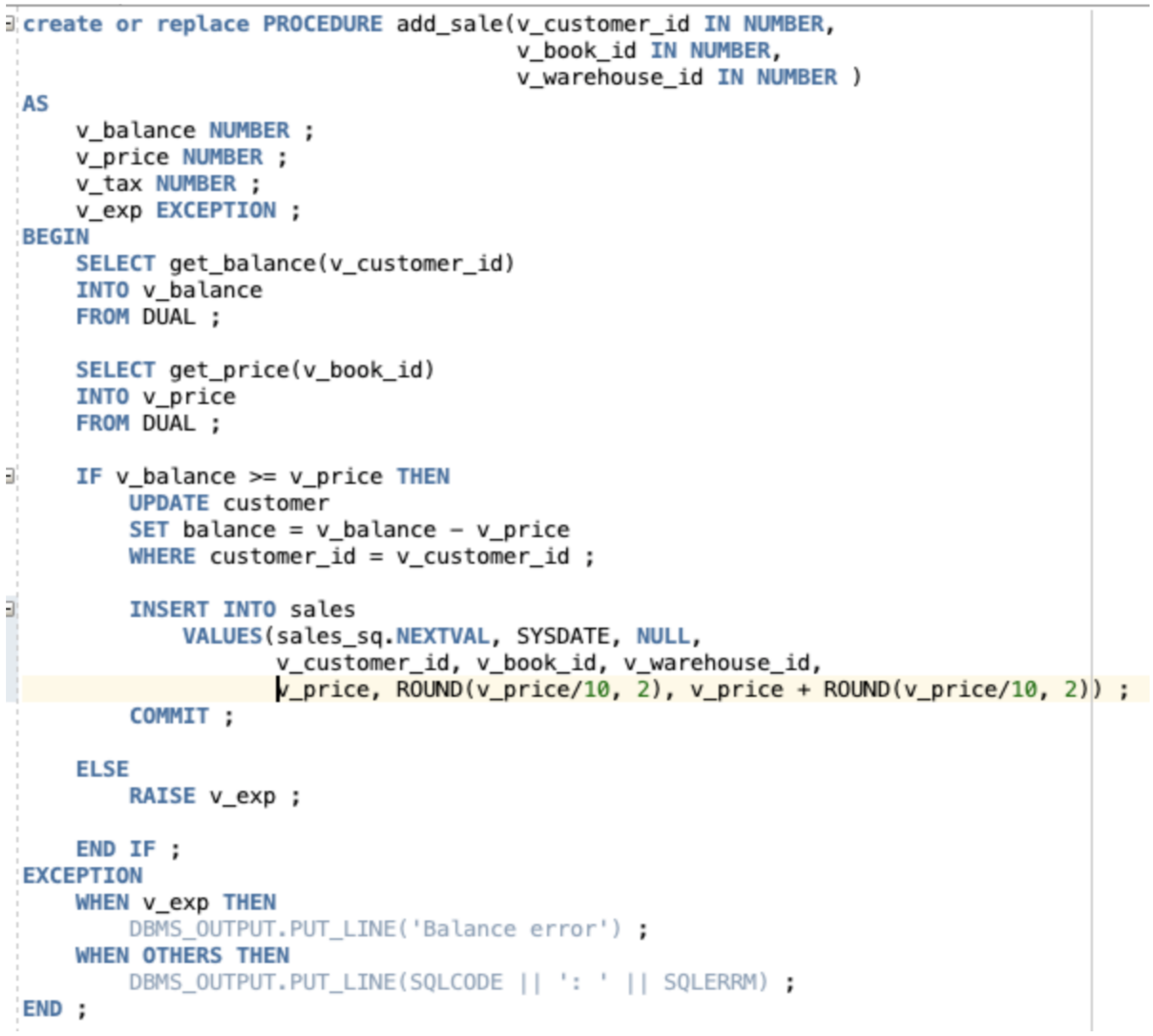
Procedure: add_sale

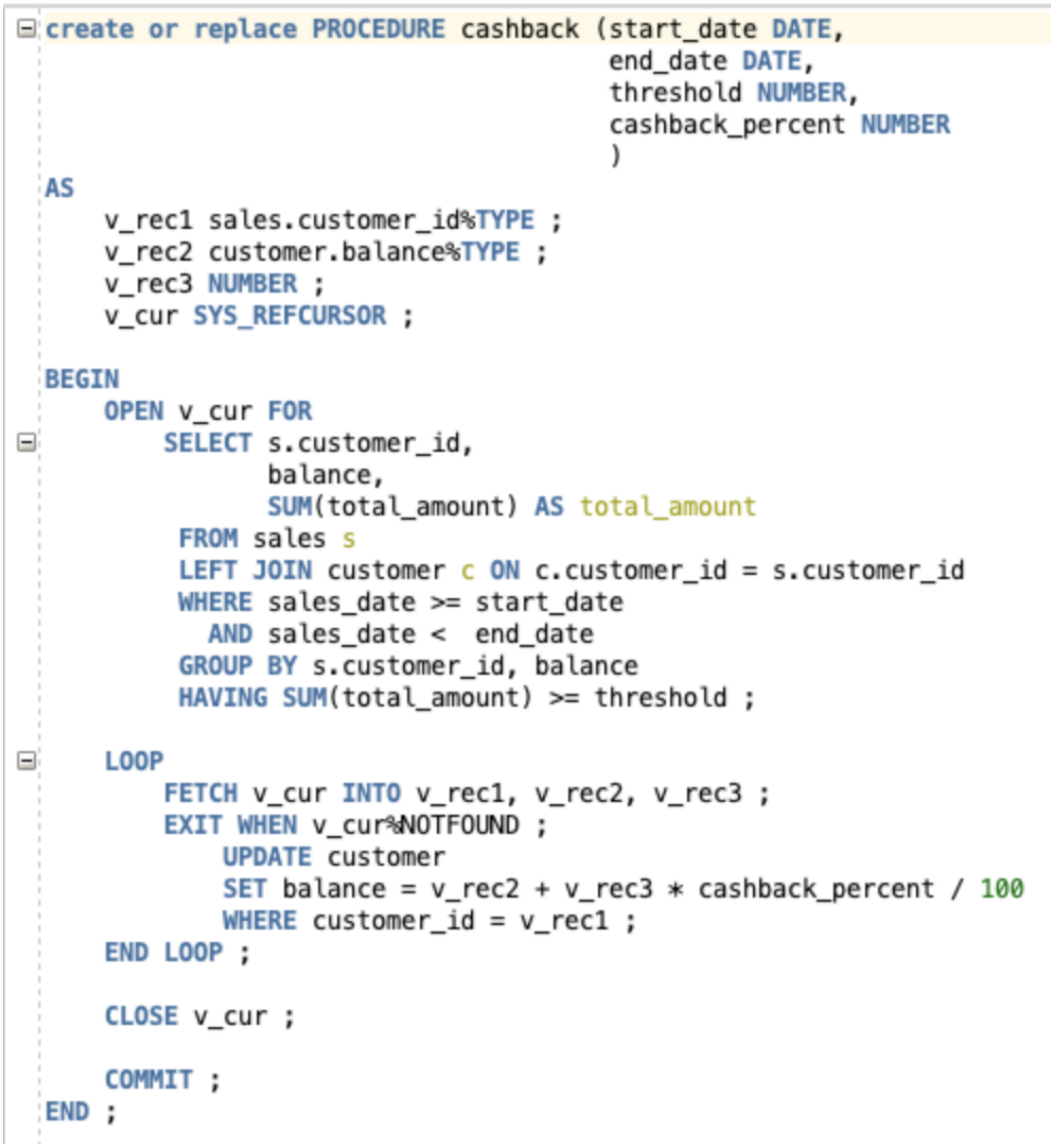
Goal: Create a procedure which adds new sale



```
create or replace PROCEDURE add_sale(v_customer_id IN NUMBER,
                                      v_book_id IN NUMBER,
                                      v_warehouse_id IN NUMBER )
AS
    v_balance NUMBER ;
    v_price NUMBER ;
    v_tax NUMBER ;
    v_exp EXCEPTION ;
BEGIN
    SELECT get_balance(v_customer_id)
    INTO v_balance
    FROM DUAL ;

    SELECT get_price(v_book_id)
    INTO v_price
    FROM DUAL ;

    IF v_balance >= v_price THEN
        UPDATE customer
        SET balance = v_balance - v_price
        WHERE customer_id = v_customer_id ;

        INSERT INTO sales
            VALUES(sales_sq.NEXTVAL, SYSDATE, NULL,
                   v_customer_id, v_book_id, v_warehouse_id,
                   v_price, ROUND(v_price/10, 2), v_price + ROUND(v_price/10, 2)) ;
        COMMIT ;

    ELSE
        RAISE v_exp ;

    END IF ;
EXCEPTION
    WHEN v_exp THEN
        DBMS_OUTPUT.PUT_LINE('Balance error') ;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLCODE || ': ' || SQLERRM) ;
END ;
```

Procedure: cashback

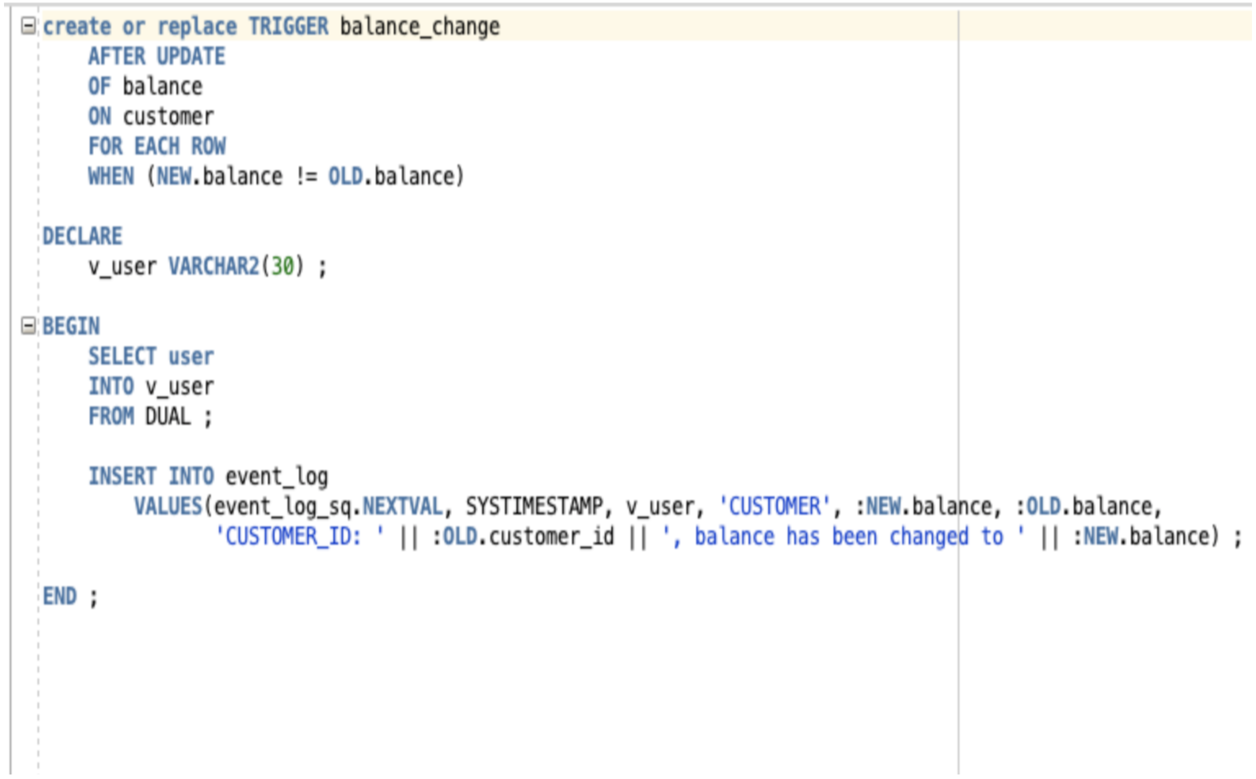
Goal: Provide cashback (cashback_percent) for customers spent more than certain amount (threshold) of money within certain period (start_date, end_date)



```
create or replace PROCEDURE cashback (start_date DATE,
                                      end_date DATE,
                                      threshold NUMBER,
                                      cashback_percent NUMBER
                                      )
AS
    v_rec1 sales.customer_id%TYPE ;
    v_rec2 customer.balance%TYPE ;
    v_rec3 NUMBER ;
    v_cur SYS_REFCURSOR ;

BEGIN
    OPEN v_cur FOR
        SELECT s.customer_id,
               balance,
               SUM(total_amount) AS total_amount
         FROM sales s
         LEFT JOIN customer c ON c.customer_id = s.customer_id
         WHERE sales_date >= start_date
           AND sales_date <  end_date
         GROUP BY s.customer_id, balance
         HAVING SUM(total_amount) >= threshold ;

    LOOP
        FETCH v_cur INTO v_rec1, v_rec2, v_rec3 ;
        EXIT WHEN v_cur%NOTFOUND ;
            UPDATE customer
            SET balance = v_rec2 + v_rec3 * cashback_percent / 100
            WHERE customer_id = v_rec1 ;
    END LOOP ;

    CLOSE v_cur ;

    COMMIT ;
END ;
```

Trigger: balance_change

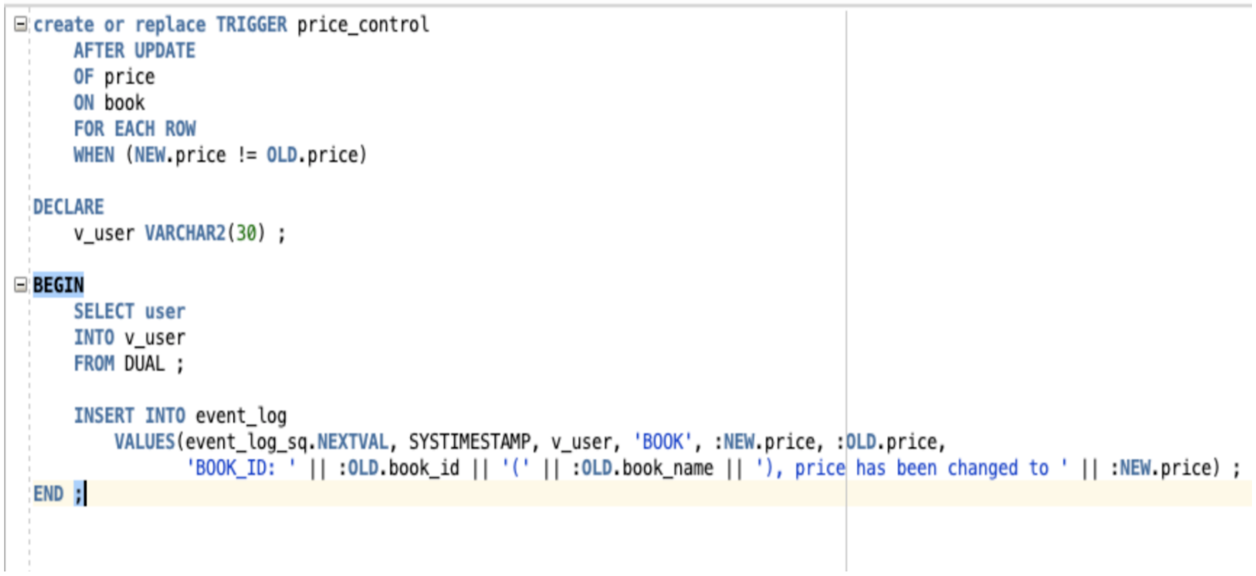
Goal: Ensure that if customer's balance is changed this operation will be logged



```
create or replace TRIGGER balance_change
    AFTER UPDATE
    OF balance
    ON customer
    FOR EACH ROW
    WHEN (NEW.balance != OLD.balance)

DECLARE
    v_user VARCHAR2(30) ;

BEGIN
    SELECT user
    INTO v_user
    FROM DUAL ;

    INSERT INTO event_log
        VALUES(event_log_sq.NEXTVAL, SYSTIMESTAMP, v_user, 'CUSTOMER', :NEW.balance, :OLD.balance,
               'CUSTOMER_ID: ' || :OLD.customer_id || ', balance has been changed to ' || :NEW.balance) ;

END ;
```

Trigger: price_control

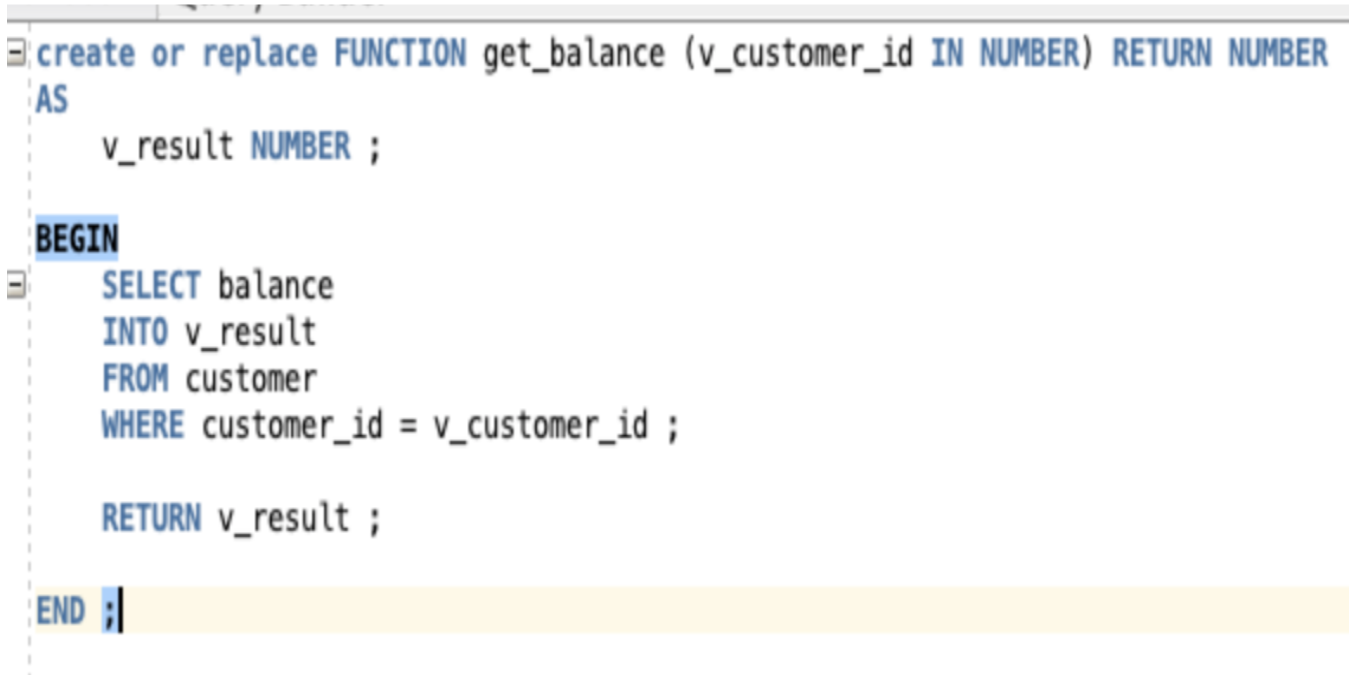
Goal: Ensure that if book price is changed this operation will be logged



```
create or replace TRIGGER price_control
    AFTER UPDATE
    OF price
    ON book
    FOR EACH ROW
    WHEN (NEW.price != OLD.price)

DECLARE
    v_user VARCHAR2(30) ;

BEGIN
    SELECT user
    INTO v_user
    FROM DUAL ;

    INSERT INTO event_log
        VALUES(event_log_sq.NEXTVAL, SYSTIMESTAMP, v_user, 'BOOK', :NEW.price, :OLD.price,
               'BOOK_ID: ' || :OLD.book_id || '(' || :OLD.book_name || '), price has been changed to ' || :NEW.price) ;
END ;
```

Function: get_balance

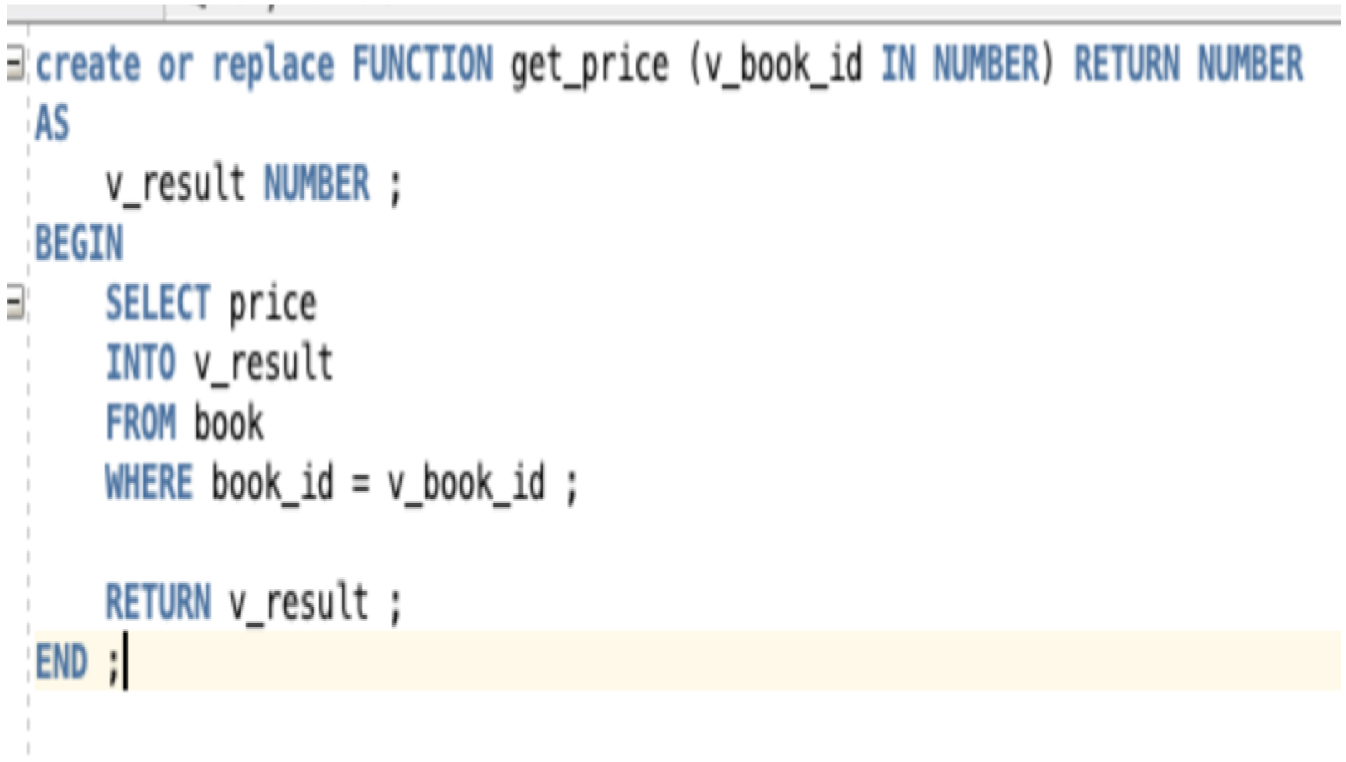
Goal: Create a function that extracts the current balance of a customer.



```
create or replace FUNCTION get_balance (v_customer_id IN NUMBER) RETURN NUMBER
AS
    v_result NUMBER ;

BEGIN
    SELECT balance
    INTO v_result
    FROM customer
    WHERE customer_id = v_customer_id ;

    RETURN v_result ;

END ;
```

Function: get_price

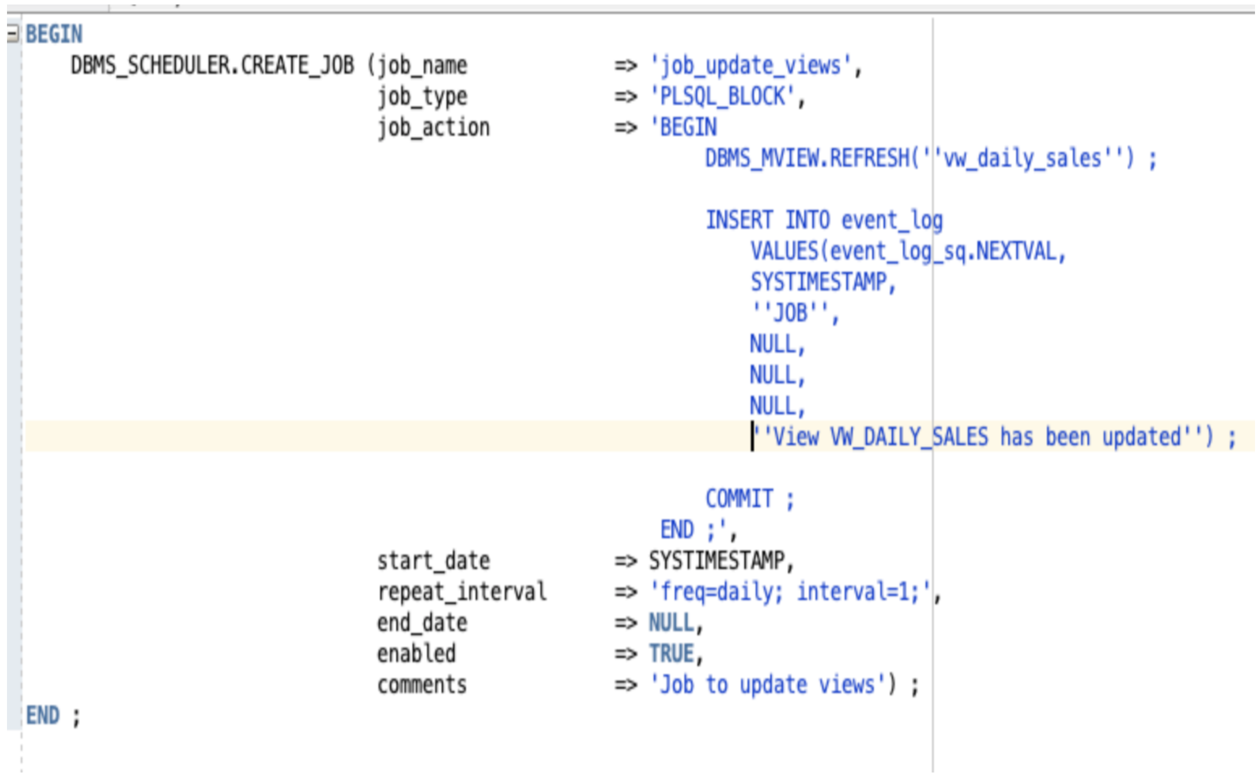
Goal: Create a function that extracts the price of a book



```
create or replace FUNCTION get_price (v_book_id IN NUMBER) RETURN NUMBER
AS
    v_result NUMBER ;
BEGIN
    SELECT price
    INTO v_result
    FROM book
    WHERE book_id = v_book_id ;

    RETURN v_result ;
END ;
```

Job: job_update_views

Goal: Create a job to update materialized views on daily basis



```
BEGIN
    DBMS_SCHEDULER.CREATE_JOB (job_name             => 'job_update_views',
                               job_type             => 'PLSQL_BLOCK',
                               job_action           => 'BEGIN
                                                            DBMS_MVIEW.REFRESH(''vw_daily_sales'') ;

                                                            INSERT INTO event_log
                                                                VALUES(event_log_sq.NEXTVAL,
                                                                SYSTIMESTAMP,
                                                                ''JOB'',
                                                                NULL,
                                                                NULL,
                                                                NULL,
                                                                ''View VW_DAILY_SALES has been updated'') ;

                                                            COMMIT ;
                                                        END ;',
                               start_date           => SYSTIMESTAMP,
                               repeat_interval      => 'freq=daily; interval=1;',
                               end_date             => NULL,
                               enabled              => TRUE,
                               comments             => 'Job to update views') ;
END ;
```