CSED Practical Activity Report

submitted for

Database Management System

(UCS310)

Submitted by:

Aryan Bindra (102303269)

Shubham Goyal (102303272)

Harshit Katyal (102303276)

Bhavish Pushkarna (102303279)

BE Second Year Batch – 2C22

Submitted to:

Mrs. Geeta Kasana

Computer Science and Engineering Department

Thapar Institute of Engineering & Technology,

Patiala   Jan-May 2025

TABLE OF CONTENT

Acknowledgement

With immense appreciation, we extend our deepest gratitude to the constellation of individuals and entities who have illuminated the path of the SIMS project from its inception to fruition.

First and foremost, we offer our profound thanks to our exceptional lab teacher, **Mrs. Geeta Kasana**, for her invaluable contributions and unwavering support throughout the development of SIMS. Her mentorship, insightful guidance, and the wealth of knowledge she so generously shared were absolutely fundamental to the success of this project. Her dedication to our learning and her belief in our abilities have been truly inspiring.

Following this, we extend our sincere appreciation to all **Sports Complex Faculty** for their valuable contributions and support throughout the development of SIMS. Your insightful guidance and constructive feedback have been instrumental in shaping SIMS into the system it is today.

We are also deeply grateful to our dedicated **team members** for their collaborative spirit, tireless efforts, and individual skills. Your hard work and commitment were crucial in bringing the SIMS project to fruition, and your contributions are highly valued. Working together, we have achieved something significant, and your dedication has been truly commendable.

The successful completion of SIMS is a direct result of the collective dedication, support, and expertise of each and every one mentioned. Thank you for being an integral part of this achievement.

Introduction

The Sports Item Management System (SIMS) project aims to develop a robust database-driven application to streamline the management of sports equipment and student performance tracking within an educational institution. The system facilitates the issuing and returning of sports items by students, while maintaining accurate records for inventory tracking and fine management. Additionally, the platform includes a performance tracker to monitor student participation and performance, enabling authorities to shortlist students for competitions and advanced training programs. SIMS ensures efficient equipment utilization, reduces misuse, and supports informed decision-making in sports development activities.

Project Scope

The Sports Item Management System (SIMS) project addresses the inefficiencies of the traditional manual process of issuing and tracking sports equipment. By implementing a centralized DBMS, the system streamlines item issuance, return, and inventory management. It reduces errors, improves accountability, and includes a performance tracker to help identify and shortlist students for sports events based on their participation and activity.

Project Features

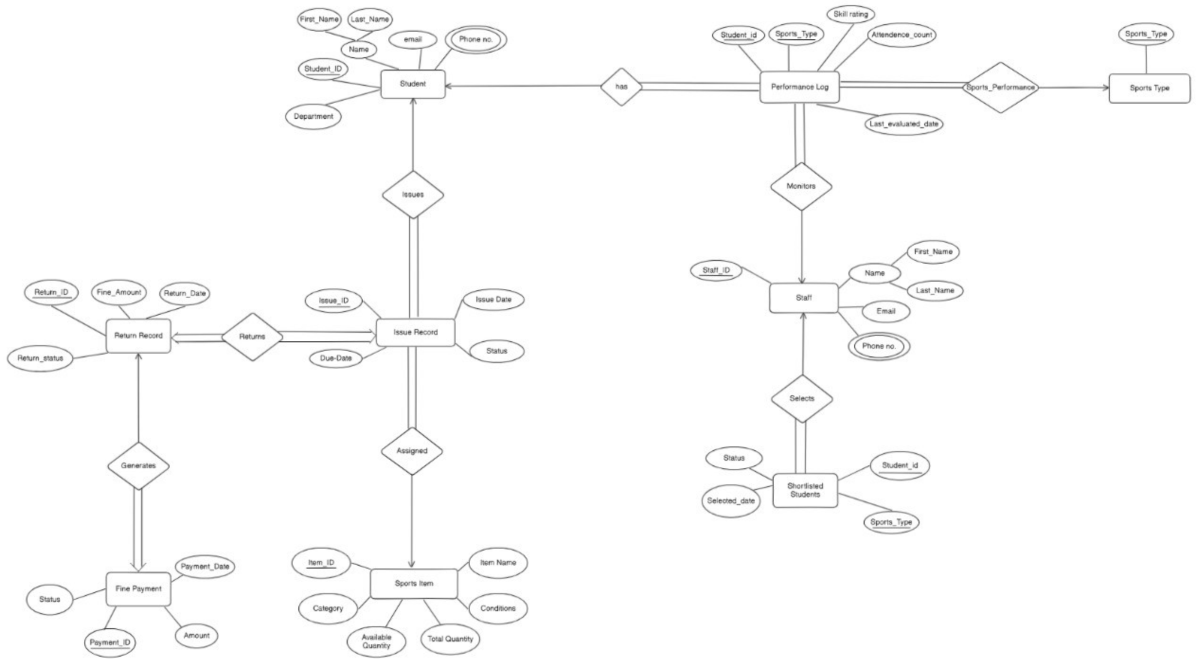
- **Issuance Management:** Enables issuance of sports items to students by logging the item type, student details, and date/time of issue for full accountability and tracking.
- **Return Tracking:** Facilitates the recording of returned items and automatically updates each item’s availability status, maintaining accurate inventory control.
- **Fine Management:** Applies an automatic fine of ₹10 per day for overdue returns, calculating penalties based on due dates and updating student accounts accordingly.
- **Performance Tracker:** Maintains a log of student participation and skill ratings, helping staff evaluate performance and efficiently shortlist candidates for teams or events.

Database Schema:

The database schema consists of the following tables:

- **Student**: Stores student details including student ID, name, department, and email.
- **Student_Phone**: Maintains one or more phone numbers associated with each student (linked to the Student table via Student_ID).
- **Sports_Item**: Contains data about each sports item such as item ID, name, category, condition, available quantity, and total quantity.
- **Issue_Record**: Records issuance of sports items with fields like issue ID, issue date, due date, and status. Linked to students and items.
- **Return_Record**: Stores return information including return ID, return date, and calculated fine amount. Connected to the related issue record.
- **Fine_Payment**: Captures fine payment details including payment ID, amount, and payment date. Generated when a late return occurs.
- **Performance_Log**: Tracks student participation and performance in various sports with fields like sports type, skill rating, attendance count, and last evaluated date. Linked to both students and staff.
- **Staff**: Contains staff information including staff ID, name, and email.
- **Staff_Phone**: Stores one or more contact numbers for each staff member (linked to the Staff table via Staff_ID).
- **Shortlisted_Students**: Holds data on shortlisted candidates including shortlisted ID, status, and selected date. Associated with both students and staff.
- **Sports Type**: To contain the sports items to be there in the performance log.

ER Diagram



Normalization

Normalization is the process of organizing database tables to minimize redundancy and dependency. It involves dividing large tables into smaller ones and defining relationships using foreign keys. The commonly used normal forms are **1NF**, **2NF**, and **3NF**.

Step 1: Identify the primary key

The primary key uniquely identifies each row in the table.

- Student: Student_ID
- Staff: Staff_ID
- Sports_Item: Item_ID
- Issue_Record: Issue_ID
- Return_Record: Return_ID
- Fine_Payment: Payment_ID
- Performance_Log: Student_ID,Sports_Type (Composite Key)
- Shortlisted_Students: Student_ID,Sports_Type (Composite Key)
- Student_Phone: Student_ID,Phone_No (Composite Key)
- Staff_Phone: Staff_ID,Phone_No (Composite Key)
- Sports_type: Sport_type

Step 2: Eliminate repeating groups

Repeating groups are columns that contain multiple values in a single row. Tables like Student, Staff, and Performance_Log may include multiple phone numbers. To eliminate repeating groups, we create separate **phone tables** for both Student and Staff.

Step 3: Create separate tables for related data

Here are the normalized tables:

1. Student

```
CREATE TABLE Student (

   student_id INT PRIMARY KEY,

   first_name VARCHAR(50),

   last_name VARCHAR(50),

   email VARCHAR(100),

   department VARCHAR(100) );
```

2. Student_Phone

```
CREATE TABLE Student_Phone (
    student_id INT,
    phone_no VARCHAR(20),
    PRIMARY KEY (student_id, phone_no),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

3. Staff

```
CREATE TABLE Staff (
    staff_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);
```

4. Staff_Phone

```
CREATE TABLE Staff_Phone (
    staff_id INT,
    phone_no VARCHAR(20),
    PRIMARY KEY (staff_id, phone_no),
    FOREIGN KEY (staff_id) REFERENCES Staff(staff_id)
);
```

5. Item

```
CREATE TABLE Item (
   item_id INT PRIMARY KEY,
   item_name VARCHAR(100),
   condition VARCHAR(50),
   category VARCHAR(50),
   total_qty INT,
   available_qty INT
);
```

---

6. Issue_Record

```
CREATE TABLE Issue_Record (
   issue_id INT PRIMARY KEY,
   issue_date DATE,
   due_date DATE,
   status VARCHAR(20),
   student_id INT,
   item_id INT,
   FOREIGN KEY (student_id) REFERENCES Student(student_id),
   FOREIGN KEY (item_id) REFERENCES Item(item_id)
);
```

---

7. Return_Record

```
CREATE TABLE Return_Record (
   return_id INT PRIMARY KEY,
   return_date DATE,
   fine_amount DECIMAL(10, 2),
   return_status VARCHAR(20),

   issue_id INT UNIQUE,

   FOREIGN KEY (issue_id) REFERENCES Issue_Record(issue_id)

);
```

---

8. Fine_Payment

```
CREATE TABLE Fine_Payment (

    payment_id INT PRIMARY KEY,

    payment_amount DECIMAL(10, 2),

    payment_date DATE,

    return_id INT UNIQUE,

    status VARCHAR2(20),

    FOREIGN KEY (return_id) REFERENCES Return_Record(return_id)

);
```

---

9. Sports_Type

```
CREATE TABLE Sports_Type (

    sport_type VARCHAR2(20) PRIMARY KEY

);
```

---

10. Shortlist

```
CREATE TABLE Shortlist (

    student_id    NUMBER(10),

    sports_type   VARCHAR2(30),

    staff_id      NUMBER(10),

    selected_date DATE,

    status        VARCHAR2(20),
    PRIMARY KEY (student_id, sports_type),

    FOREIGN KEY (student_id) REFERENCES Student(student_id),

    FOREIGN KEY (staff_id) REFERENCES Staff(staff_id),

    FOREIGN KEY (sports_type) REFERENCES Sports_Type(sport_type)

);
```

---

11. Performance_Log

```
CREATE TABLE Performance_Log (

    student_id INT,

    sports_type VARCHAR2(20),

    staff_id INT,

    skill_rating DECIMAL(3, 2),

    attendance_count INT,

    last_evaluated_date DATE,

    PRIMARY KEY (student_id, sports_type),

    FOREIGN KEY (student_id) REFERENCES Student(student_id),

    FOREIGN KEY (staff_id) REFERENCES Staff(staff_id),

    FOREIGN KEY (sports_type) REFERENCES Sports_Type(sport_type)

);
```

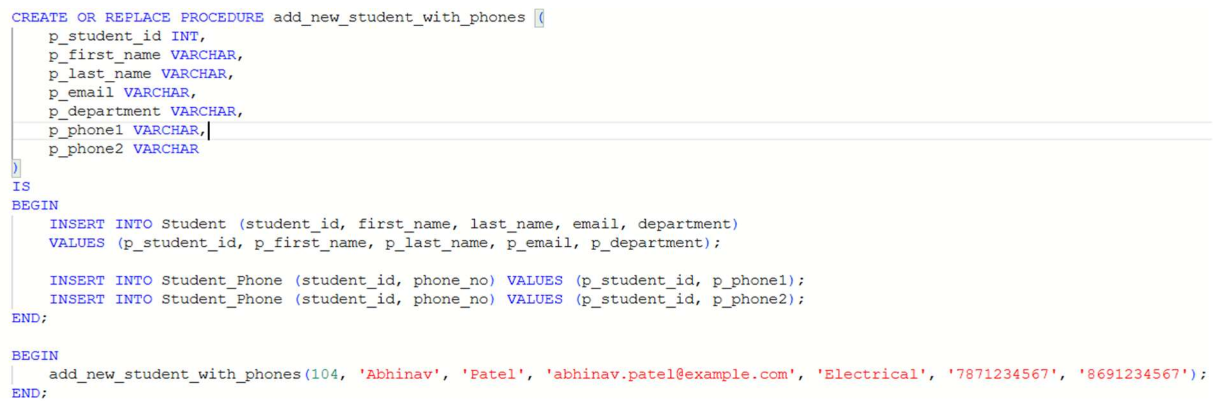
Step 4: Relationships and Foreign Keys

1. **Student — Phone Numbers (1:N)**
   - **Relation**: A Student can have multiple Phone Numbers.
   - **Implementation**: Use a separate Student_Phone entity/table with a foreign key Student_ID.
2. **Staff — Phone Numbers (1:N)**
   - **Relation**: A Staff member can have multiple Phone Numbers.
   - **Implementation**: Use a separate Staff_Phone entity/table with a foreign key Staff_ID.

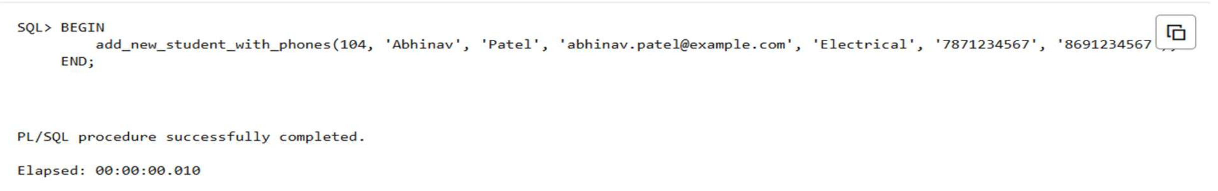
3. **Student — Issue Record (1:N)**
    - **Relation**: A Student can issue multiple Sports Items.
    - **Via**: Issue_Record entity linking Student and Sports_Item.
4. **Issue Record — Return Record (1:1)**
    - **Relation**: An Issue_Record may have at most one corresponding Return_Record.
5. **Return Record — Fine Payment (1:1 or 0:1)**
    - **Relation**: A Return_Record may generate one Fine_Payment.
6. **Staff — Monitors — Performance Log (1:N)**
    - **Relation**: A Staff member monitors multiple Performance_Log entries.
    - **Note**: Each Performance_Log belongs to exactly one Student.
7. **Staff — Selects — Shortlisted Students (1:N)**
    - **Relation**: A Staff member can select multiple Shortlisted_Students.
8. **Performance Log — Student (1:1 or M:1)**
    - **Relation**: A Performance_Log entry is maintained per Student.
    - **Note**: Each Student has at least one or exactly one log entry.
9. **Issue Record — Sports Item (M:1)**
    - **Relation**: Each Issue_Record is assigned to one Sports_Item, but a Sports_Item can be issued multiple times.
10. **Performance_Log — Sports Type (M:1)**
    - **Relation**: Each Sports type can be there in more than 1 performance log but a performance log can have only 1 sports type.

Procedures

Inserting data into Student and Student phone

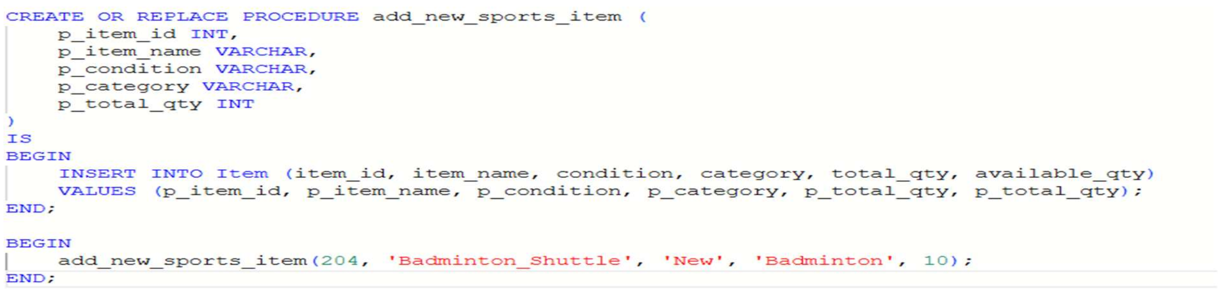


```
CREATE OR REPLACE PROCEDURE add_new_student_with_phones (
    p_student_id INT,
    p_first_name VARCHAR,
    p_last_name VARCHAR,
    p_email VARCHAR,
    p_department VARCHAR,
    p_phone1 VARCHAR,
    p_phone2 VARCHAR
)
IS
BEGIN
    INSERT INTO Student (student_id, first_name, last_name, email, department)
    VALUES (p_student_id, p_first_name, p_last_name, p_email, p_department);

    INSERT INTO Student_Phone (student_id, phone_no) VALUES (p_student_id, p_phone1);
    INSERT INTO Student_Phone (student_id, phone_no) VALUES (p_student_id, p_phone2);
END;

BEGIN
    add_new_student_with_phones(104, 'Abhinav', 'Patel', 'abhinav.patel@example.com', 'Electrical', '7871234567', '8691234567');
END;
```

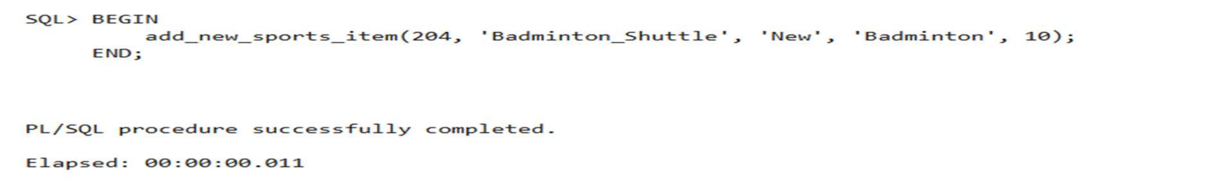


```
SQL> BEGIN
         add_new_student_with_phones(104, 'Abhinav', 'Patel', 'abhinav.patel@example.com', 'Electrical', '7871234567', '8691234567
     END;


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

Inserting data into Sports Item

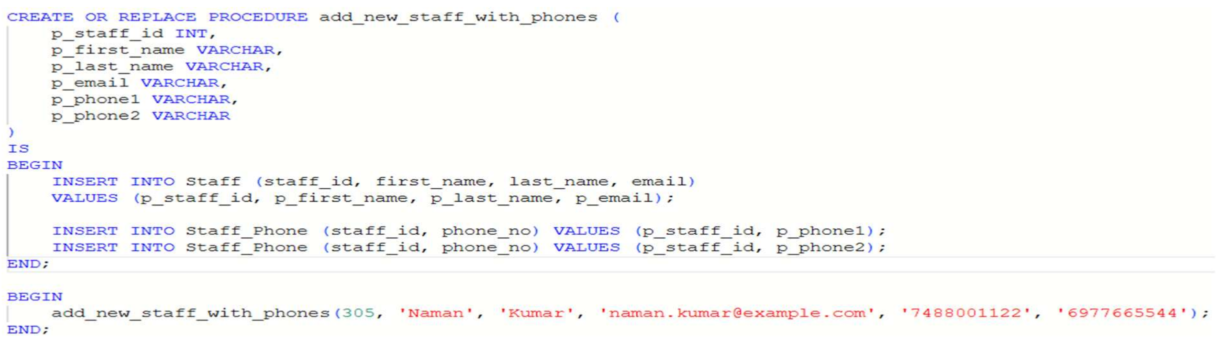


```
CREATE OR REPLACE PROCEDURE add_new_sports_item (
    p_item_id INT,
    p_item_name VARCHAR,
    p_condition VARCHAR,
    p_category VARCHAR,
    p_total_qty INT
)
IS
BEGIN
    INSERT INTO Item (item_id, item_name, condition, category, total_qty, available_qty)
    VALUES (p_item_id, p_item_name, p_condition, p_category, p_total_qty, p_total_qty);
END;

BEGIN
    add_new_sports_item(204, 'Badminton_Shuttle', 'New', 'Badminton', 10);
END;
```

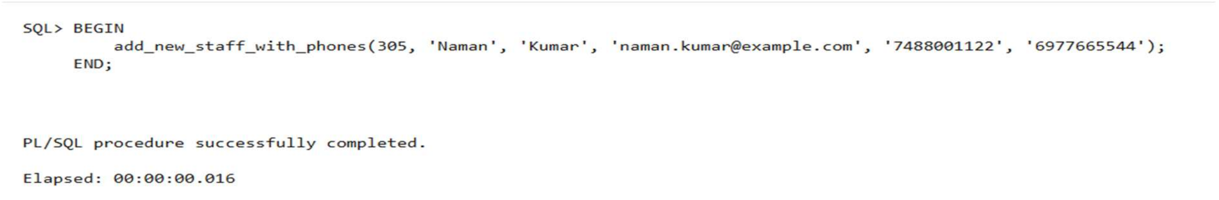


```
SQL> BEGIN
         add_new_sports_item(204, 'Badminton_Shuttle', 'New', 'Badminton', 10);
     END;


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.011
```

Inserting data into Staff and Staff Phone

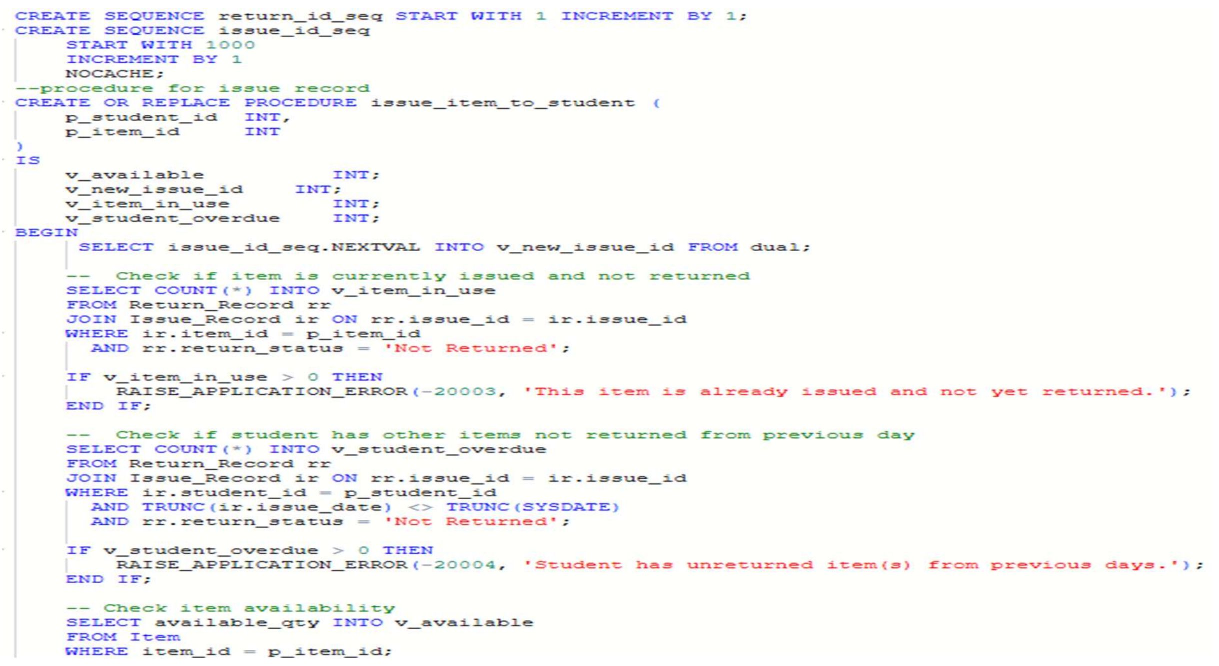


```
CREATE OR REPLACE PROCEDURE add_new_staff_with_phones (
    p_staff_id INT,
    p_first_name VARCHAR,
    p_last_name VARCHAR,
    p_email VARCHAR,
    p_phone1 VARCHAR,
    p_phone2 VARCHAR
)
IS
BEGIN
    INSERT INTO Staff (staff_id, first_name, last_name, email)
    VALUES (p_staff_id, p_first_name, p_last_name, p_email);

    INSERT INTO Staff_Phone (staff_id, phone_no) VALUES (p_staff_id, p_phone1);
    INSERT INTO Staff_Phone (staff_id, phone_no) VALUES (p_staff_id, p_phone2);
END;

BEGIN
    add_new_staff_with_phones(305, 'Naman', 'Kumar', 'naman.kumar@example.com', '7488001122', '6977665544');
END;
```

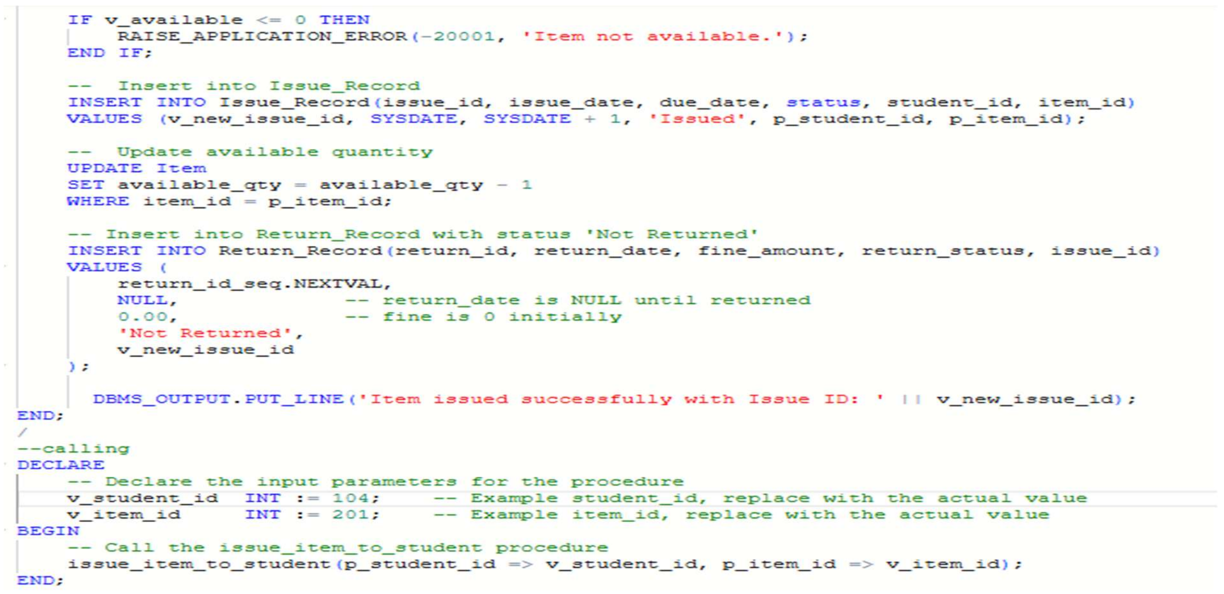


```
SQL> BEGIN
         add_new_staff_with_phones(305, 'Naman', 'Kumar', 'naman.kumar@example.com', '7488001122', '6977665544');
     END;


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.016
```

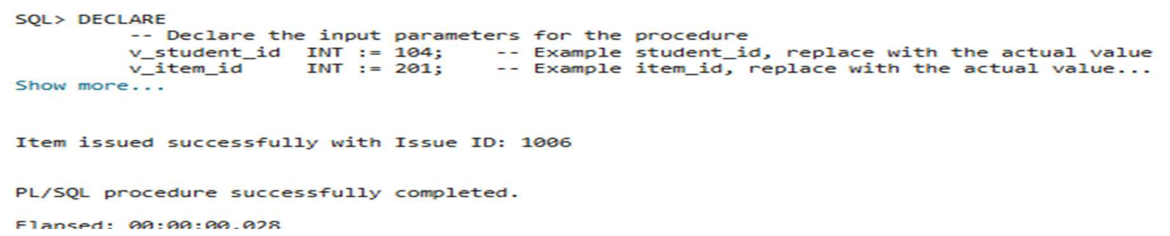
Inserting data into Issue Record



```
CREATE SEQUENCE return_id_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE issue_id_seq
    START WITH 1000
    INCREMENT BY 1
    NOCACHE;
--procedure for issue record
CREATE OR REPLACE PROCEDURE issue_item_to_student (
    p_student_id  INT,
    p_item_id     INT
)
IS
    v_available          INT;
    v_new_issue_id     INT;
    v_item_in_use        INT;
    v_student_overdue    INT;
BEGIN
     SELECT issue_id_seq.NEXTVAL INTO v_new_issue_id FROM dual;

    --  Check if item is currently issued and not returned
    SELECT COUNT(*) INTO v_item_in_use
    FROM Return_Record rr
    JOIN Issue_Record ir ON rr.issue_id = ir.issue_id
    WHERE ir.item_id = p_item_id
      AND rr.return_status = 'Not Returned';

    IF v_item_in_use > 0 THEN
        RAISE_APPLICATION_ERROR(-20003, 'This item is already issued and not yet returned.');
    END IF;

    --  Check if student has other items not returned from previous day
    SELECT COUNT(*) INTO v_student_overdue
    FROM Return_Record rr
    JOIN Issue_Record ir ON rr.issue_id = ir.issue_id
    WHERE ir.student_id = p_student_id
      AND TRUNC(ir.issue_date) <> TRUNC(SYSDATE)
      AND rr.return_status = 'Not Returned';

    IF v_student_overdue > 0 THEN
        RAISE_APPLICATION_ERROR(-20004, 'Student has unreturned item(s) from previous days.');
    END IF;

    -- Check item availability
    SELECT available_qty INTO v_available
    FROM Item
    WHERE item_id = p_item_id;
```

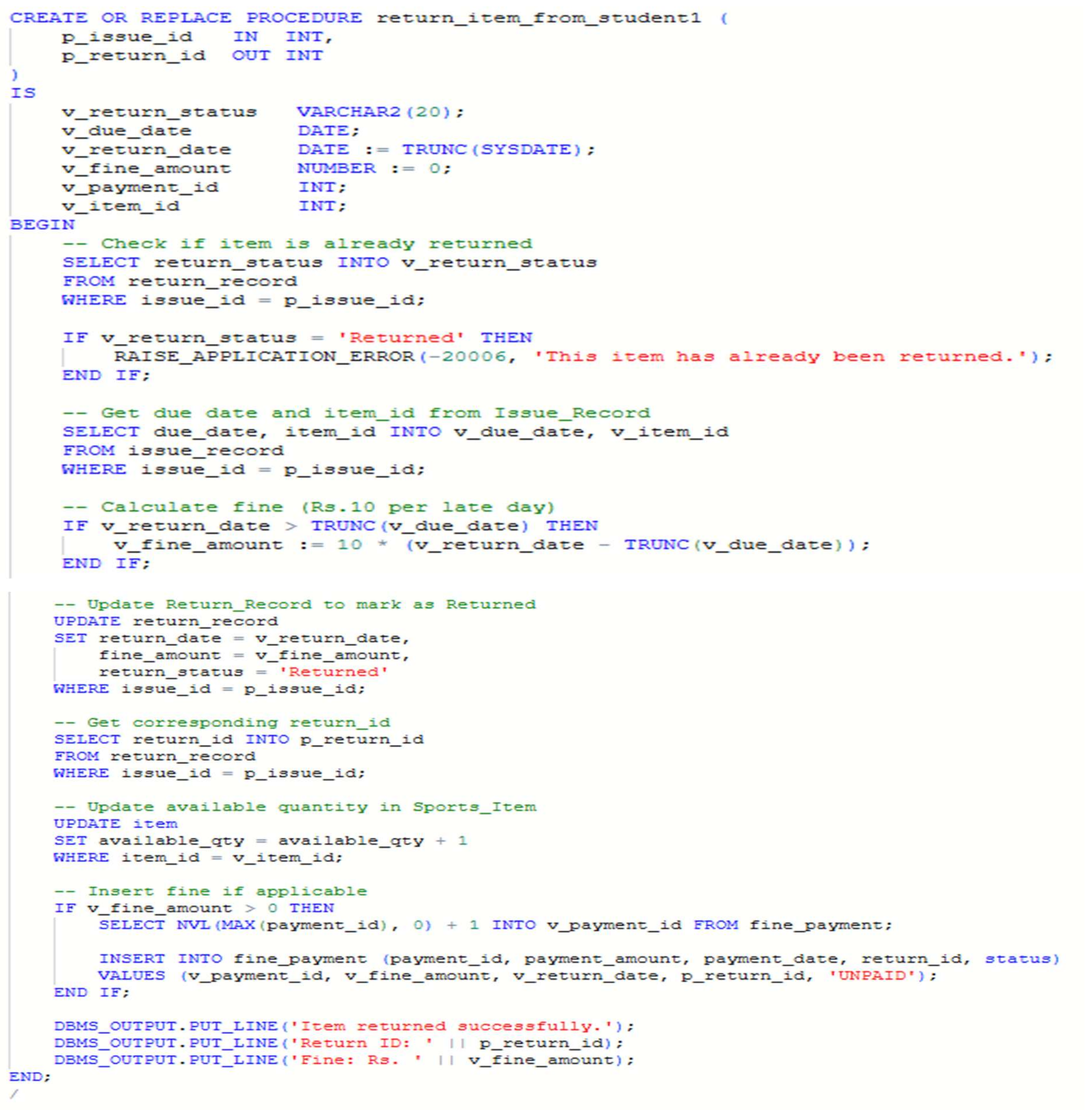


```
    IF v_available <= 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Item not available.');
    END IF;

    --  Insert into Issue_Record
    INSERT INTO Issue_Record(issue_id, issue_date, due_date, status, student_id, item_id)
    VALUES (v_new_issue_id, SYSDATE, SYSDATE + 1, 'Issued', p_student_id, p_item_id);

    --  Update available quantity
    UPDATE Item
    SET available_qty = available_qty - 1
    WHERE item_id = p_item_id;

    -- Insert into Return_Record with status 'Not Returned'
    INSERT INTO Return_Record(return_id, return_date, fine_amount, return_status, issue_id)
    VALUES (
        return_id_seq.NEXTVAL,
        NULL,              -- return_date is NULL until returned
        0.00,              -- fine is 0 initially
        'Not Returned',
        v_new_issue_id
    );

     DBMS_OUTPUT.PUT_LINE('Item issued successfully with Issue ID: ' || v_new_issue_id);
END;
/
--calling
DECLARE
    -- Declare the input parameters for the procedure
    v_student_id  INT := 104;     -- Example student_id, replace with the actual value
    v_item_id     INT := 201;     -- Example item_id, replace with the actual value
BEGIN
    -- Call the issue_item_to_student procedure
    issue_item_to_student(p_student_id => v_student_id, p_item_id => v_item_id);
END;
```

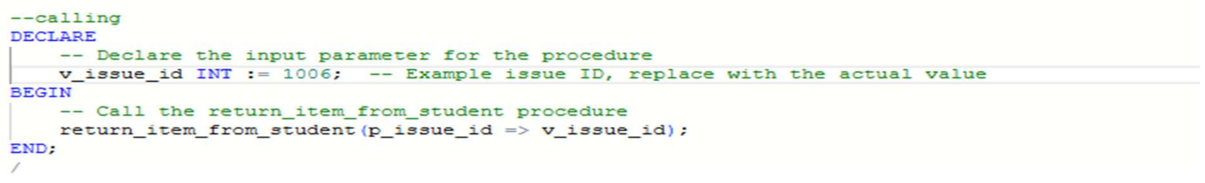


```
SQL> DECLARE
         -- Declare the input parameters for the procedure
         v_student_id  INT := 104;     -- Example student_id, replace with the actual value
         v_item_id     INT := 201;     -- Example item_id, replace with the actual value...
Show more...


Item issued successfully with Issue ID: 1006


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.028
```

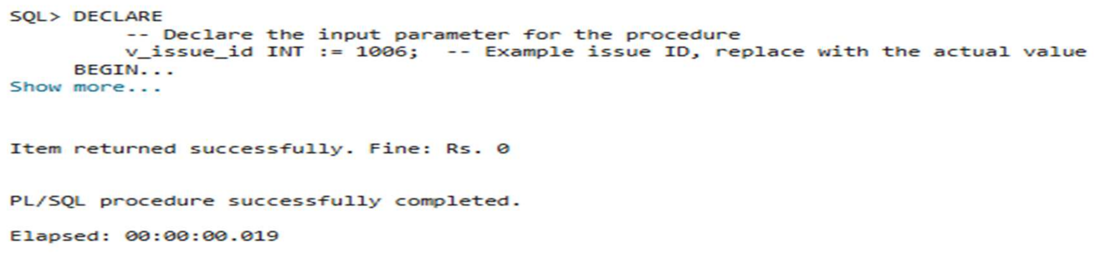
Inserting data into Return Record



```
CREATE OR REPLACE PROCEDURE return_item_from_student1 (
    p_issue_id   IN  INT,
    p_return_id  OUT INT
)
IS
    v_return_status   VARCHAR2(20);
    v_due_date        DATE;
    v_return_date     DATE := TRUNC(SYSDATE);
    v_fine_amount     NUMBER := 0;
    v_payment_id      INT;
    v_item_id         INT;
BEGIN
    -- Check if item is already returned
    SELECT return_status INTO v_return_status
    FROM return_record
    WHERE issue_id = p_issue_id;

    IF v_return_status = 'Returned' THEN
        RAISE_APPLICATION_ERROR(-20006, 'This item has already been returned.');
    END IF;

    -- Get due date and item_id from Issue_Record
    SELECT due_date, item_id INTO v_due_date, v_item_id
    FROM issue_record
    WHERE issue_id = p_issue_id;

    -- Calculate fine (Rs.10 per late day)
    IF v_return_date > TRUNC(v_due_date) THEN
        v_fine_amount := 10 * (v_return_date - TRUNC(v_due_date));
    END IF;

    -- Update Return_Record to mark as Returned
    UPDATE return_record
    SET return_date = v_return_date,
        fine_amount = v_fine_amount,
        return_status = 'Returned'
    WHERE issue_id = p_issue_id;

    -- Get corresponding return_id
    SELECT return_id INTO p_return_id
    FROM return_record
    WHERE issue_id = p_issue_id;

    -- Update available quantity in Sports_Item
    UPDATE item
    SET available_qty = available_qty + 1
    WHERE item_id = v_item_id;

    -- Insert fine if applicable
    IF v_fine_amount > 0 THEN
        SELECT NVL(MAX(payment_id), 0) + 1 INTO v_payment_id FROM fine_payment;

        INSERT INTO fine_payment (payment_id, payment_amount, payment_date, return_id, status)
        VALUES (v_payment_id, v_fine_amount, v_return_date, p_return_id, 'UNPAID');
    END IF;

    DBMS_OUTPUT.PUT_LINE('Item returned successfully.');
    DBMS_OUTPUT.PUT_LINE('Return ID: ' || p_return_id);
    DBMS_OUTPUT.PUT_LINE('Fine: Rs. ' || v_fine_amount);
END;
/
```

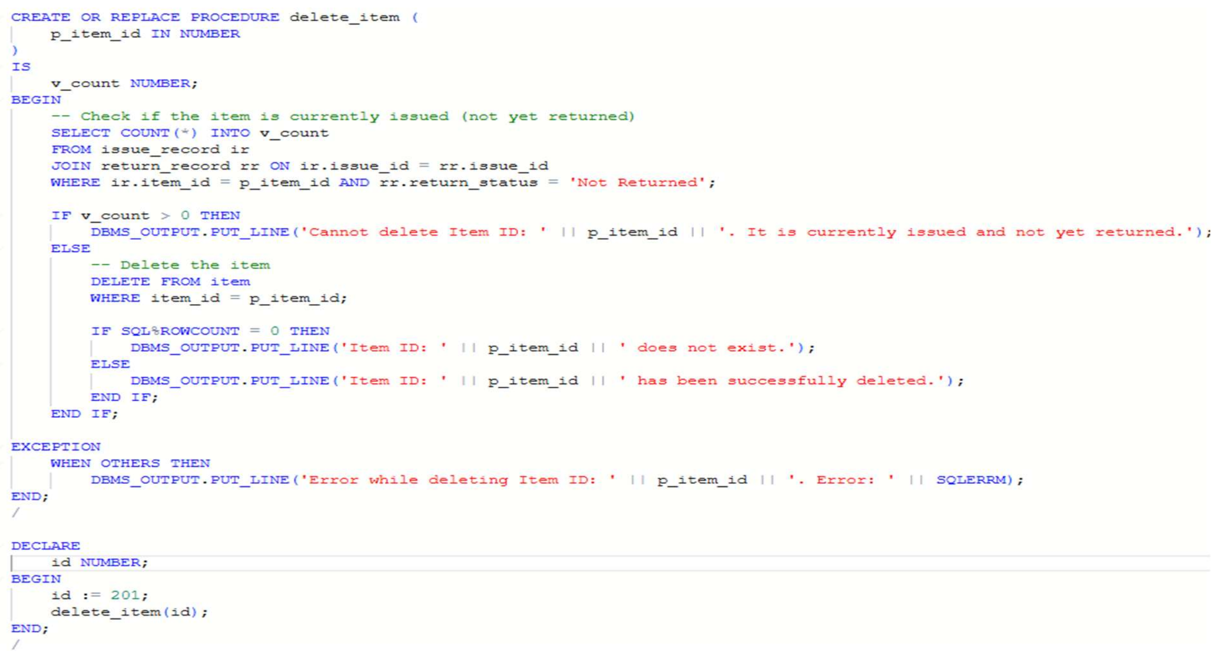


```
--calling
DECLARE
    -- Declare the input parameter for the procedure
    v_issue_id INT := 1006;  -- Example issue ID, replace with the actual value
BEGIN
    -- Call the return_item_from_student procedure
    return_item_from_student(p_issue_id => v_issue_id);
END;
/
```

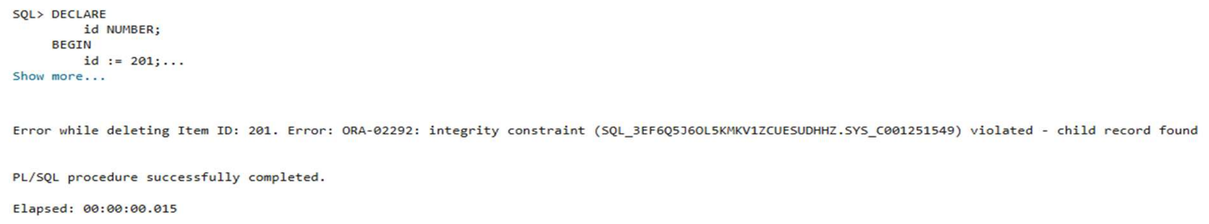


```
SQL> DECLARE
         -- Declare the input parameter for the procedure
         v_issue_id INT := 1006;  -- Example issue ID, replace with the actual value
     BEGIN...
Show more...


Item returned successfully. Fine: Rs. 0


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.019
```

Delete Sports Item

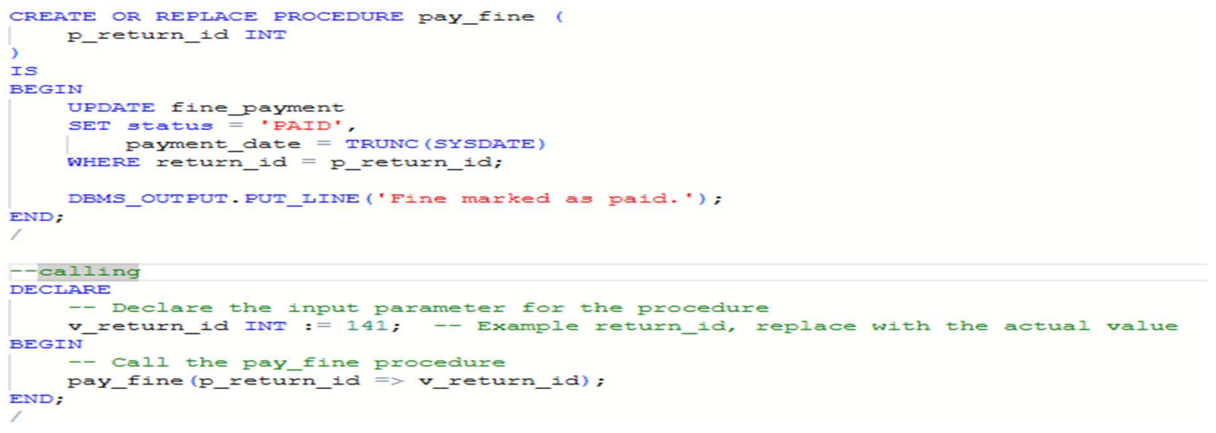


```
CREATE OR REPLACE PROCEDURE delete_item (
    p_item_id IN NUMBER
)
IS
    v_count NUMBER;
BEGIN
    -- Check if the item is currently issued (not yet returned)
    SELECT COUNT(*) INTO v_count
    FROM issue_record ir
    JOIN return_record rr ON ir.issue_id = rr.issue_id
    WHERE ir.item_id = p_item_id AND rr.return_status = 'Not Returned';

    IF v_count > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Cannot delete Item ID: ' || p_item_id || '. It is currently issued and not yet returned.');
    ELSE
        -- Delete the item
        DELETE FROM item
        WHERE item_id = p_item_id;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('Item ID: ' || p_item_id || ' does not exist.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Item ID: ' || p_item_id || ' has been successfully deleted.');
        END IF;
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error while deleting Item ID: ' || p_item_id || '. Error: ' || SQLERRM);
END;
/

DECLARE
    id NUMBER;
BEGIN
    id := 201;
    delete_item(id);
END;
/
```

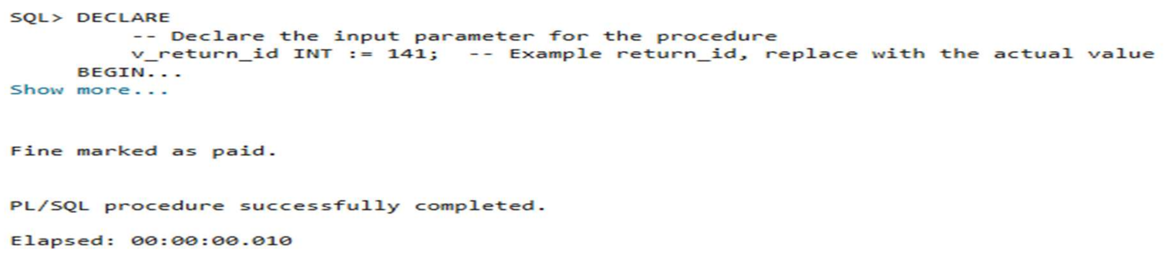


```
SQL> DECLARE
        id NUMBER;
     BEGIN
        id := 201;...
Show more...


Error while deleting Item ID: 201. Error: ORA-02292: integrity constraint (SQL_3EF6Q5J6OL5KMKV1ZCUESUDHHZ.SYS_C001251549) violated - child record found


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.015
```

Fine Payment

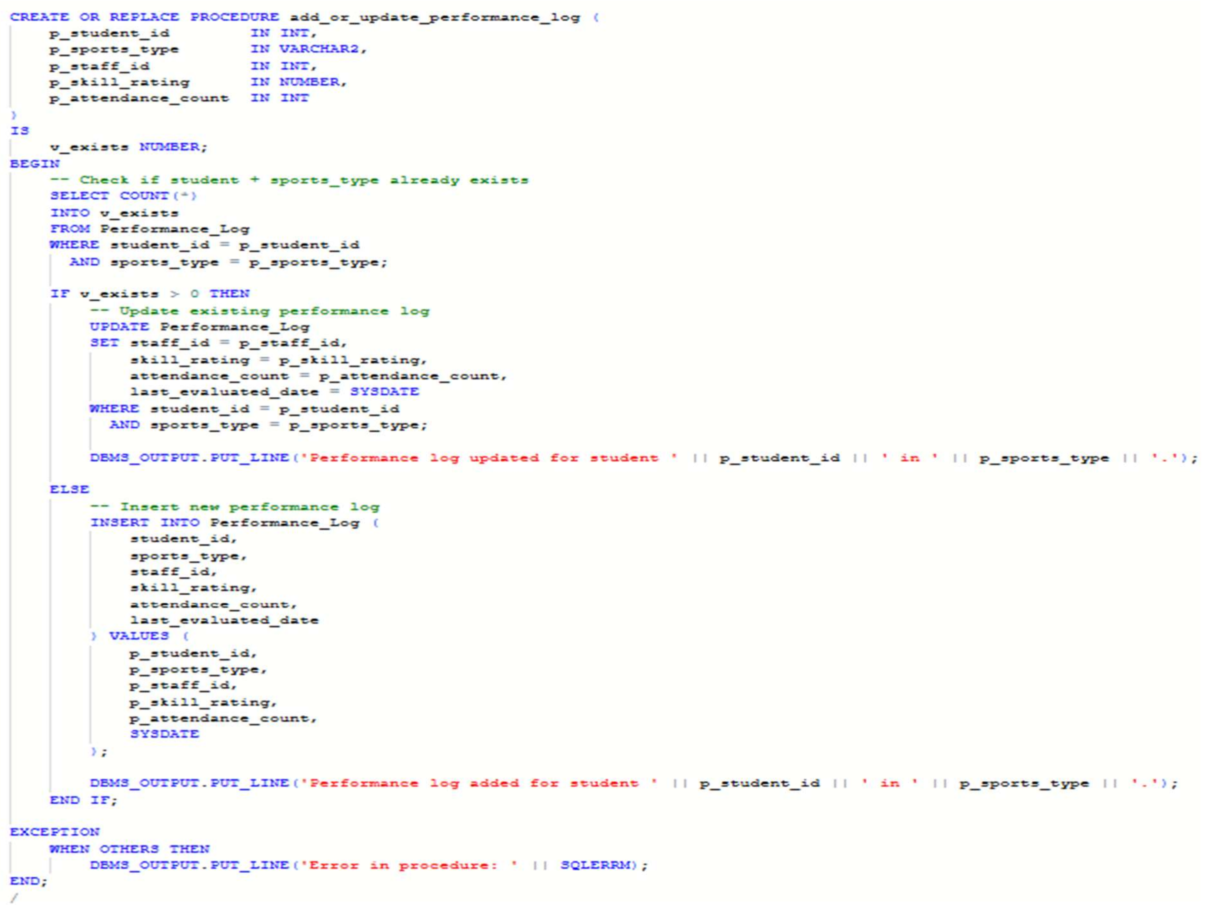


```
CREATE OR REPLACE PROCEDURE pay_fine (
    p_return_id INT
)
IS
BEGIN
    UPDATE fine_payment
    SET status = 'PAID',
        payment_date = TRUNC(SYSDATE)
    WHERE return_id = p_return_id;

    DBMS_OUTPUT.PUT_LINE('Fine marked as paid.');
END;
/

--calling
DECLARE
    -- Declare the input parameter for the procedure
    v_return_id INT := 141;  -- Example return_id, replace with the actual value
BEGIN
    -- Call the pay_fine procedure
    pay_fine(p_return_id => v_return_id);
END;
/
```

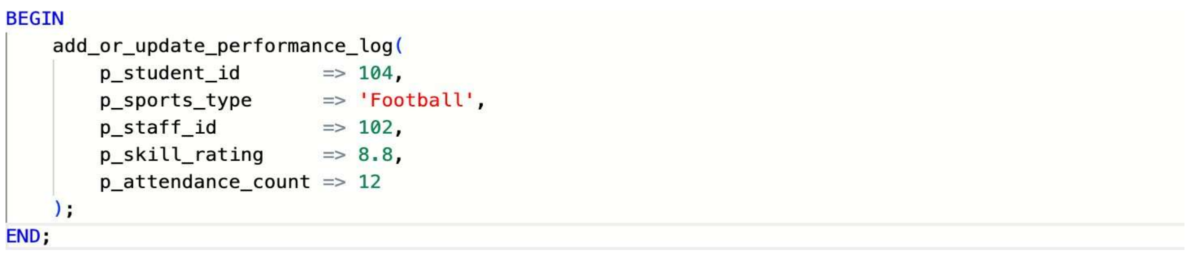


```
SQL> DECLARE
         -- Declare the input parameter for the procedure
         v_return_id INT := 141;  -- Example return_id, replace with the actual value
     BEGIN...
Show more...


Fine marked as paid.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

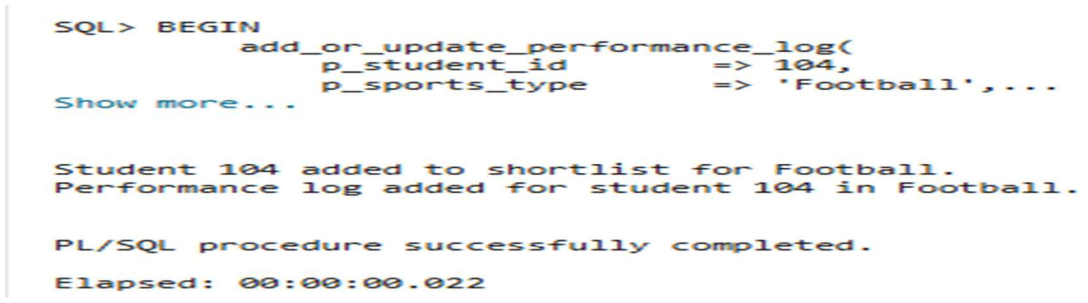
Insert into Performance Log



```
CREATE OR REPLACE PROCEDURE add_or_update_performance_log (
    p_student_id         IN INT,
    p_sports_type        IN VARCHAR2,
    p_staff_id           IN INT,
    p_skill_rating       IN NUMBER,
    p_attendance_count   IN INT
)
IS
    v_exists NUMBER;
BEGIN
    -- Check if student + sports_type already exists
    SELECT COUNT(*)
    INTO v_exists
    FROM Performance_Log
    WHERE student_id = p_student_id
      AND sports_type = p_sports_type;

    IF v_exists > 0 THEN
        -- Update existing performance log
        UPDATE Performance_Log
        SET staff_id = p_staff_id,
            skill_rating = p_skill_rating,
            attendance_count = p_attendance_count,
            last_evaluated_date = SYSDATE
        WHERE student_id = p_student_id
          AND sports_type = p_sports_type;

        DBMS_OUTPUT.PUT_LINE('Performance log updated for student ' || p_student_id || ' in ' || p_sports_type || '.');

    ELSE
        -- Insert new performance log
        INSERT INTO Performance_Log (
            student_id,
            sports_type,
            staff_id,
            skill_rating,
            attendance_count,
            last_evaluated_date
        ) VALUES (
            p_student_id,
            p_sports_type,
            p_staff_id,
            p_skill_rating,
            p_attendance_count,
            SYSDATE
        );

        DBMS_OUTPUT.PUT_LINE('Performance log added for student ' || p_student_id || ' in ' || p_sports_type || '.');
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error in procedure: ' || SQLERRM);
END;
/
```



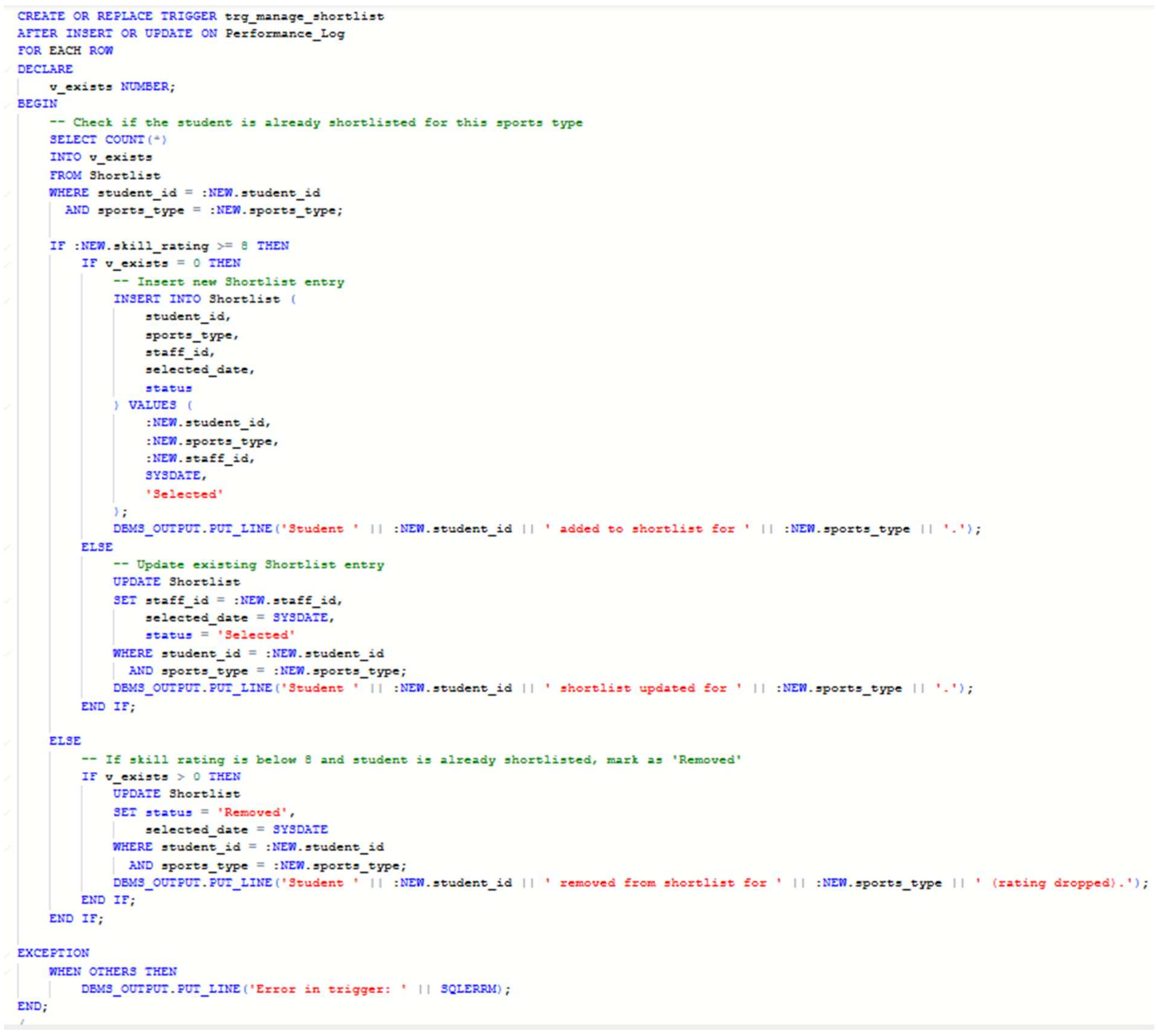
```
BEGIN
    add_or_update_performance_log(
        p_student_id       => 104,
        p_sports_type      => 'Football',
        p_staff_id         => 102,
        p_skill_rating     => 8.8,
        p_attendance_count => 12
    );
END;
```



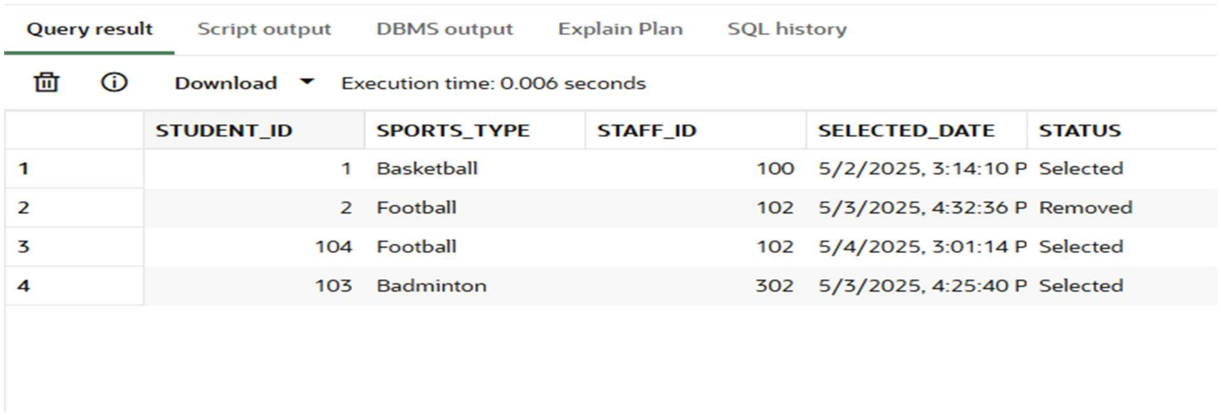
```
SQL> BEGIN
         add_or_update_performance_log(
             p_student_id         => 104,
             p_sports_type        => 'Football',...
Show more...


Student 104 added to shortlist for Football.
Performance log added for student 104 in Football.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.022
```

Triggers

Trigger to add to shortlist item after insert into performance log



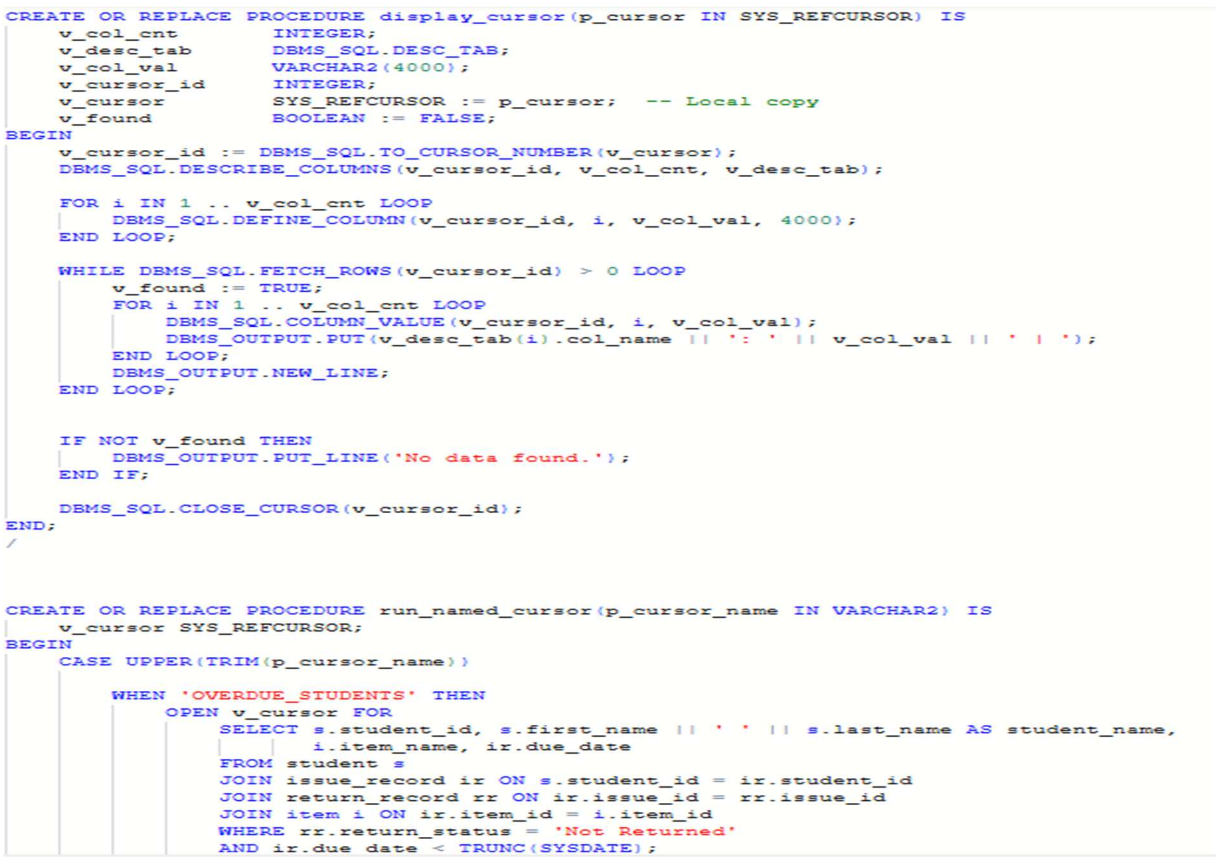
```
CREATE OR REPLACE TRIGGER trg_manage_shortlist
AFTER INSERT OR UPDATE ON Performance_Log
FOR EACH ROW
DECLARE
    v_exists NUMBER;
BEGIN
    -- Check if the student is already shortlisted for this sports type
    SELECT COUNT(*)
    INTO v_exists
    FROM Shortlist
    WHERE student_id = :NEW.student_id
      AND sports_type = :NEW.sports_type;

    IF :NEW.skill_rating >= 8 THEN
        IF v_exists = 0 THEN
            -- Insert new Shortlist entry
            INSERT INTO Shortlist (
                student_id,
                sports_type,
                staff_id,
                selected_date,
                status
            ) VALUES (
                :NEW.student_id,
                :NEW.sports_type,
                :NEW.staff_id,
                SYSDATE,
                'Selected'
            );
            DBMS_OUTPUT.PUT_LINE('Student ' || :NEW.student_id || ' added to shortlist for ' || :NEW.sports_type || '.');
        ELSE
            -- Update existing Shortlist entry
            UPDATE Shortlist
            SET staff_id = :NEW.staff_id,
                selected_date = SYSDATE,
                status = 'Selected'
            WHERE student_id = :NEW.student_id
              AND sports_type = :NEW.sports_type;
            DBMS_OUTPUT.PUT_LINE('Student ' || :NEW.student_id || ' shortlist updated for ' || :NEW.sports_type || '.');
        END IF;

    ELSE
        -- If skill rating is below 8 and student is already shortlisted, mark as 'Removed'
        IF v_exists > 0 THEN
            UPDATE Shortlist
            SET status = 'Removed',
                selected_date = SYSDATE
            WHERE student_id = :NEW.student_id
              AND sports_type = :NEW.sports_type;
            DBMS_OUTPUT.PUT_LINE('Student ' || :NEW.student_id || ' removed from shortlist for ' || :NEW.sports_type || ' (rating dropped).');
        END IF;
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error in trigger: ' || SQLERRM);
END;
/
```



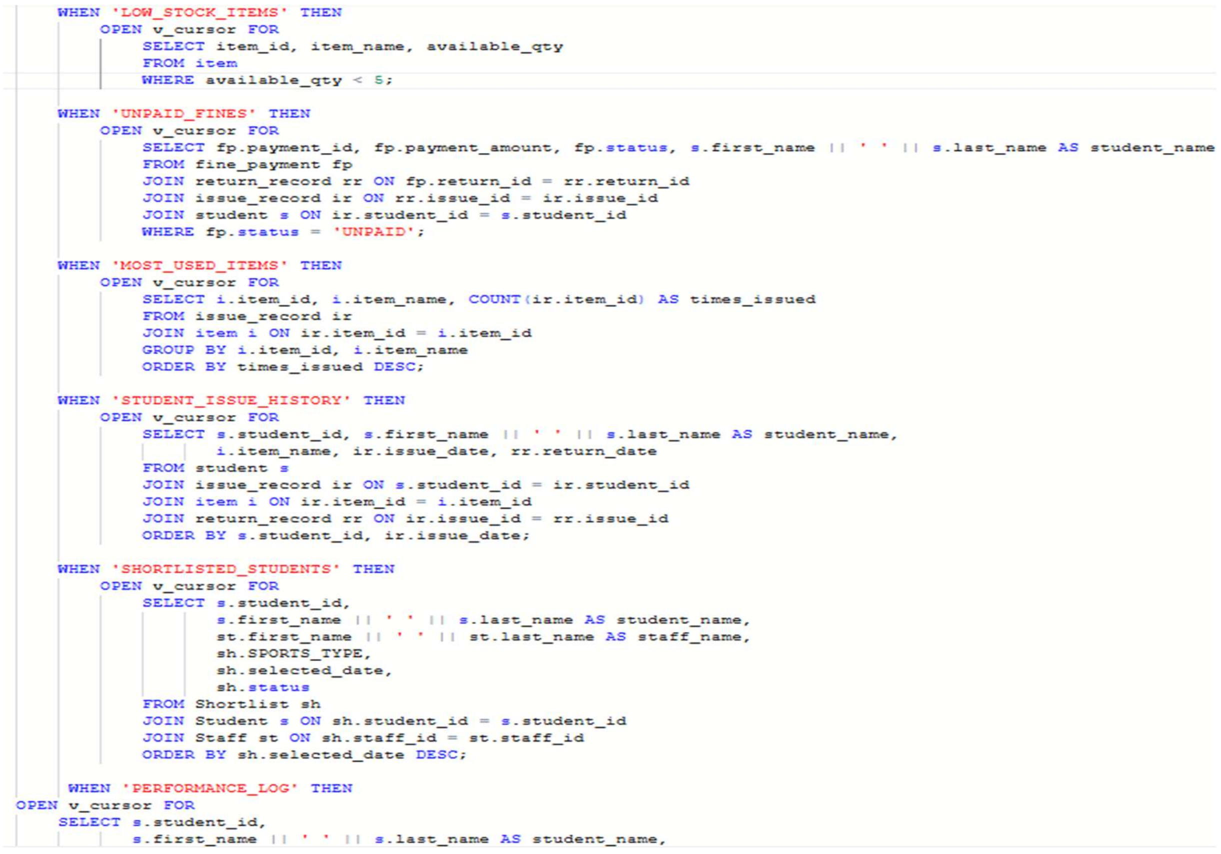
	STUDENT_ID	SPORTS_TYPE	STAFF_ID	SELECTED_DATE	STATUS
1	1	Basketball	100	5/2/2025, 3:14:10 P	Selected
2	2	Football	102	5/3/2025, 4:32:36 P	Removed
3	104	Football	102	5/4/2025, 3:01:14 P	Selected
4	103	Badminton	302	5/3/2025, 4:25:40 P	Selected

Issue id 104 was added to shortlist by the trigger as student skill rating was >8

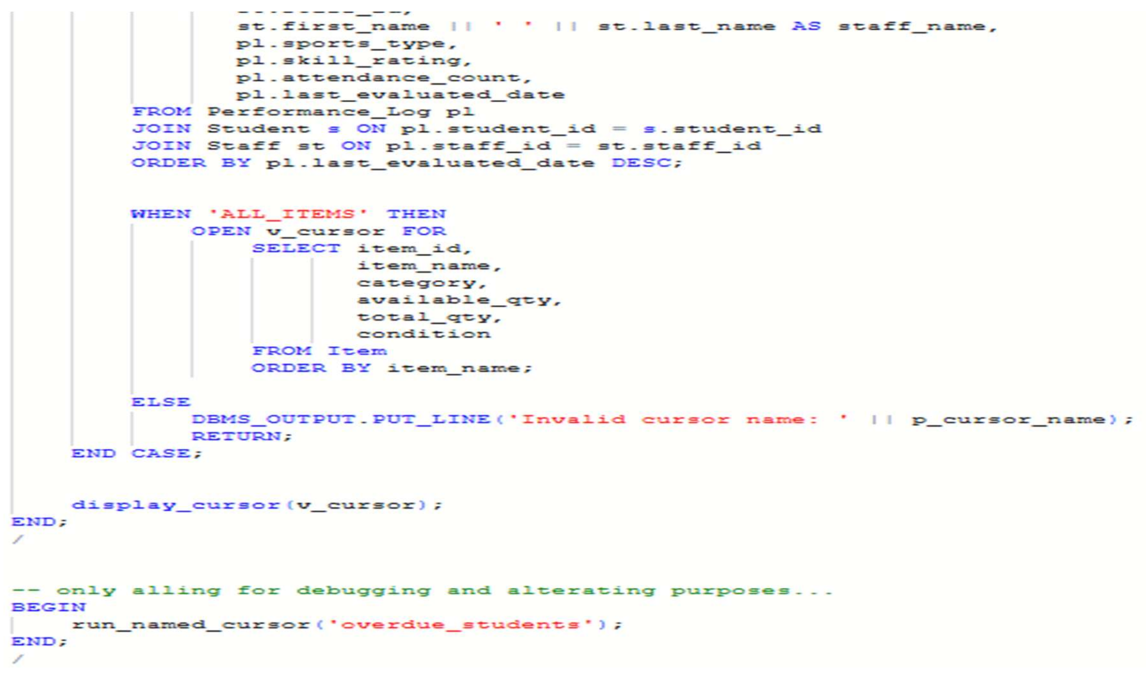
Cursors



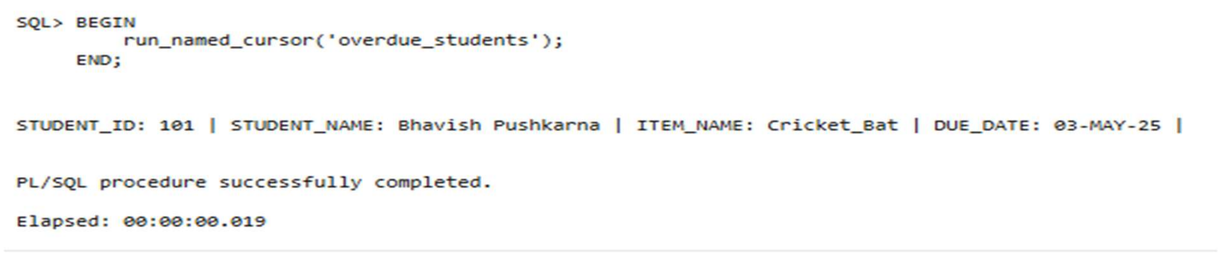
```
CREATE OR REPLACE PROCEDURE display_cursor(p_cursor IN SYS_REFCURSOR) IS
    v_col_cnt        INTEGER;
    v_desc_tab       DBMS_SQL.DESC_TAB;
    v_col_val        VARCHAR2(4000);
    v_cursor_id      INTEGER;
    v_cursor         SYS_REFCURSOR := p_cursor;  -- Local copy
    v_found          BOOLEAN := FALSE;
BEGIN
    v_cursor_id := DBMS_SQL.TO_CURSOR_NUMBER(v_cursor);
    DBMS_SQL.DESCRIBE_COLUMNS(v_cursor_id, v_col_cnt, v_desc_tab);

    FOR i IN 1 .. v_col_cnt LOOP
        DBMS_SQL.DEFINE_COLUMN(v_cursor_id, i, v_col_val, 4000);
    END LOOP;

    WHILE DBMS_SQL.FETCH_ROWS(v_cursor_id) > 0 LOOP
        v_found := TRUE;
        FOR i IN 1 .. v_col_cnt LOOP
            DBMS_SQL.COLUMN_VALUE(v_cursor_id, i, v_col_val);
            DBMS_OUTPUT.PUT(v_desc_tab(i).col_name || ': ' || v_col_val || ' | ');
        END LOOP;
        DBMS_OUTPUT.NEW_LINE;
    END LOOP;


    IF NOT v_found THEN
        DBMS_OUTPUT.PUT_LINE('No data found.');
    END IF;

    DBMS_SQL.CLOSE_CURSOR(v_cursor_id);
END;
/


CREATE OR REPLACE PROCEDURE run_named_cursor(p_cursor_name IN VARCHAR2) IS
    v_cursor SYS_REFCURSOR;
BEGIN
    CASE UPPER(TRIM(p_cursor_name))

        WHEN 'OVERDUE_STUDENTS' THEN
            OPEN v_cursor FOR
                SELECT s.student_id, s.first_name || ' ' || s.last_name AS student_name,
                       i.item_name, ir.due_date
                FROM student s
                JOIN issue_record ir ON s.student_id = ir.student_id
                JOIN return_record rr ON ir.issue_id = rr.issue_id
                JOIN item i ON ir.item_id = i.item_id
                WHERE rr.return_status = 'Not Returned'
                AND ir.due date < TRUNC(SYSDATE);
```



```
        WHEN 'LOW_STOCK_ITEMS' THEN
            OPEN v_cursor FOR
                SELECT item_id, item_name, available_qty
                FROM item
                WHERE available_qty < 5;

        WHEN 'UNPAID_FINES' THEN
            OPEN v_cursor FOR
                SELECT fp.payment_id, fp.payment_amount, fp.status, s.first_name || ' ' || s.last_name AS student_name
                FROM fine_payment fp
                JOIN return_record rr ON fp.return_id = rr.return_id
                JOIN issue_record ir ON rr.issue_id = ir.issue_id
                JOIN student s ON ir.student_id = s.student_id
                WHERE fp.status = 'UNPAID';

        WHEN 'MOST_USED_ITEMS' THEN
            OPEN v_cursor FOR
                SELECT i.item_id, i.item_name, COUNT(ir.item_id) AS times_issued
                FROM issue_record ir
                JOIN item i ON ir.item_id = i.item_id
                GROUP BY i.item_id, i.item_name
                ORDER BY times_issued DESC;

        WHEN 'STUDENT_ISSUE_HISTORY' THEN
            OPEN v_cursor FOR
                SELECT s.student_id, s.first_name || ' ' || s.last_name AS student_name,
                       i.item_name, ir.issue_date, rr.return_date
                FROM student s
                JOIN issue_record ir ON s.student_id = ir.student_id
                JOIN item i ON ir.item_id = i.item_id
                JOIN return_record rr ON ir.issue_id = rr.issue_id
                ORDER BY s.student_id, ir.issue_date;

        WHEN 'SHORTLISTED_STUDENTS' THEN
            OPEN v_cursor FOR
                SELECT s.student_id,
                       s.first_name || ' ' || s.last_name AS student_name,
                       st.first_name || ' ' || st.last_name AS staff_name,
                       sh.SPORTS_TYPE,
                       sh.selected_date,
                       sh.status
                FROM Shortlist sh
                JOIN Student s ON sh.student_id = s.student_id
                JOIN Staff st ON sh.staff_id = st.staff_id
                ORDER BY sh.selected_date DESC;

         WHEN 'PERFORMANCE_LOG' THEN
 OPEN v_cursor FOR
     SELECT s.student_id,
            s.first_name || ' ' || s.last_name AS student_name,
```



```
                st.first_name || ' ' || st.last_name AS staff_name,
                pl.sports_type,
                pl.skill_rating,
                pl.attendance_count,
                pl.last_evaluated_date
        FROM Performance_Log pl
        JOIN Student s ON pl.student_id = s.student_id
        JOIN Staff st ON pl.staff_id = st.staff_id
        ORDER BY pl.last_evaluated_date DESC;

        WHEN 'ALL_ITEMS' THEN
            OPEN v_cursor FOR
                SELECT item_id,
                       item_name,
                       category,
                       available_qty,
                       total_qty,
                       condition
                FROM Item
                ORDER BY item_name;

        ELSE
            DBMS_OUTPUT.PUT_LINE('Invalid cursor name: ' || p_cursor_name);
            RETURN;
    END CASE;

    display_cursor(v_cursor);
END;
/

-- only alling for debugging and alterating purposes...
BEGIN
    run_named_cursor('overdue_students');
END;
/
```



```
SQL> BEGIN
         run_named_cursor('overdue_students');
     END;


STUDENT_ID: 101 | STUDENT_NAME: Bhavish Pushkarna | ITEM_NAME: Cricket_Bat | DUE_DATE: 03-MAY-25 |


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.019
```

Conclusion

In conclusion, the Sports Item Management System provides a comprehensive and efficient solution for managing sports inventory and tracking student performance. By automating item issuance and returns, minimizing human errors, and enabling data-driven selection processes, the system not only improves operational workflow but also promotes greater student engagement in sports activities. Its scalable design ensures adaptability for future enhancements, making it a valuable tool for modern educational institutions.

Project Impact

The implementation of the Sports Item Management System is expected to significantly enhance the efficiency of sports resource management in educational institutions by digitizing the item issuance and return process, minimizing manual errors, and reducing dependency on paperwork. Additionally, the integrated performance tracking feature supports data-driven student selection for competitions, promoting fair evaluation and encouraging active participation in sports.

Future Enhancements

Future enhancements to the Sports Item Management System (SIMS) may include the development of a mobile application for easy item issuance, returns, and performance tracking, as well as integration with online payment gateways to automate fine collection for late returns or damages. Additionally, advanced analytics and dashboards can be implemented to monitor student performance and item usage trends for better decision-making in sports team selection.