TEST SCRIPT FOR STORED PROCEDURE

OVERVIEW

This section presents the complete test script and execution evidence for all stored logic implemented in our database system, including stored procedures, triggers, indexes, and views. Each test demonstrates how the system enforces complex business rules through both successful operations and intentional failure cases. By executing controlled transactions—such as inserting conflicting ratings, overlapping streaming availability, invalid genre assignments, duplicate list items, and multi-winner nominations—we verify that our stored logic behaves correctly and consistently. Screenshots of real SQL output are included to provide transparent, verifiable proof that every constraint, validation rule, and automated process works as intended. Together, these tests confirm the reliability, robustness, and correctness of our procedural SQL implementation.

HEADER

TEST SCRIPT FOR STORED LOGIC (STORED PROCEDURES, TRIGGERS, VIEWS, INDEXES)

IMDb Clone Database

Purpose: Demonstrate successful execution + error handling (business rules)

- This script tests:

  • usp_add_user_rating

  • usp_add_review

  • usp_add_list_item

  • usp_show_monthly_stats

  • usp_calc_title_pop

  • usp_show_person_activity

  • All triggers (streaming overlap, versioning, adult genre, etc.)

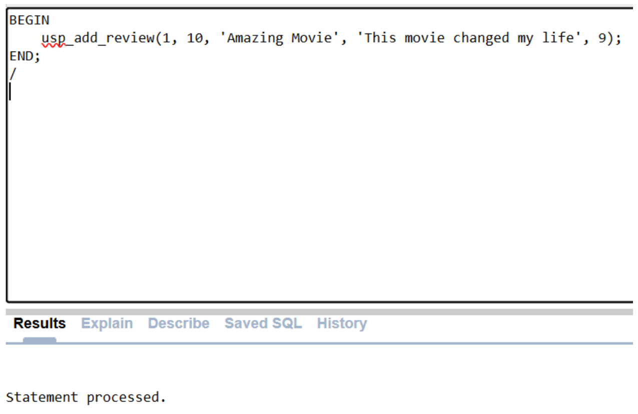
  • Views and Indexes

---

1. TESTING PROCEDURE: usp_add_review

1.1 SUCCESS CASE

CALLING:

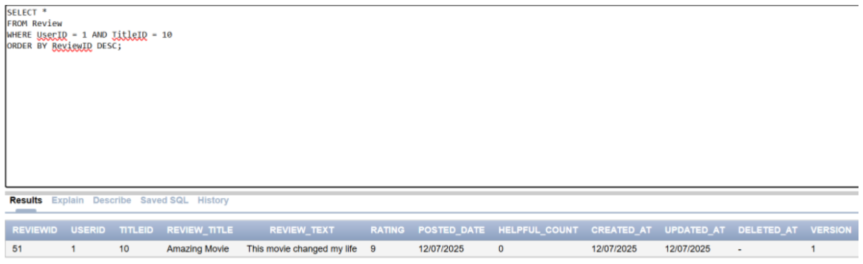


```
BEGIN
    usp_add_review(1, 10, 'Amazing Movie', 'This movie changed my life', 9);
END;
/
```

Results Explain Describe Saved SQL History

```
Statement processed.
```

RESULT:



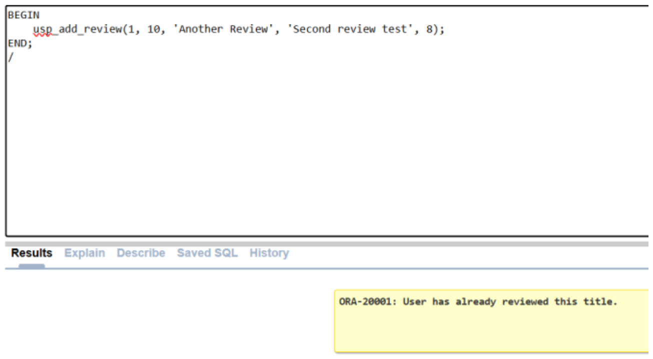
```
SELECT *
FROM Review
WHERE UserID = 1 AND TitleID = 10
ORDER BY ReviewID DESC;
```

Results Explain Describe Saved SQL History

REVIEWID	USERID	TITLEID	REVIEW_TITLE	REVIEW_TEXT	RATING	POSTED_DATE	HELPFUL_COUNT	CREATED_AT	UPDATED_AT	DELETED_AT	VERSION
51	1	10	Amazing Movie	This movie changed my life	9	12/07/2025	0	12/07/2025	12/07/2025	-	1

CONFIRMATION: HERE WE DO TESTING OUR PROCEDURE.

BY AGAIN PUTTING THE SAME REVIEW THERE .



```
BEGIN
    usp_add_review(1, 10, 'Another Review', 'Second review test', 8);
END;
/
```

Results Explain Describe Saved SQL History

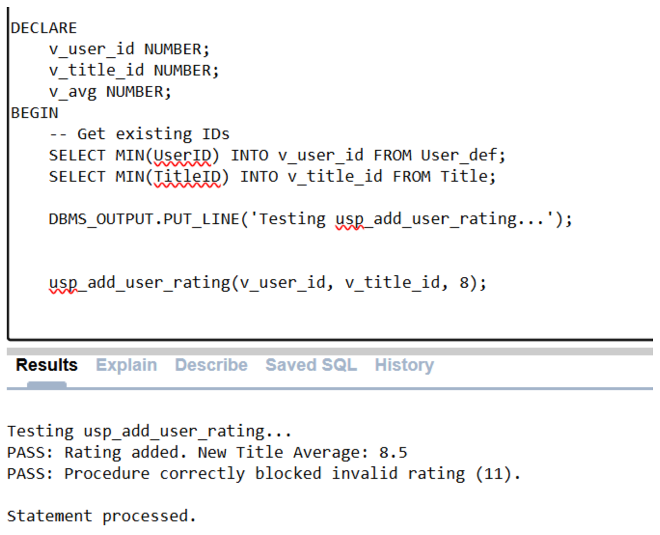
```
ORA-20001: User has already reviewed this title.
```

FROM NOW WE WILL SHOW ERROR HANDLING THROUGH DBMS.PUT_LINE

----

2. TESTING PROCEDURE: usp_add_user_rating

1.2 SUCCESS CASE

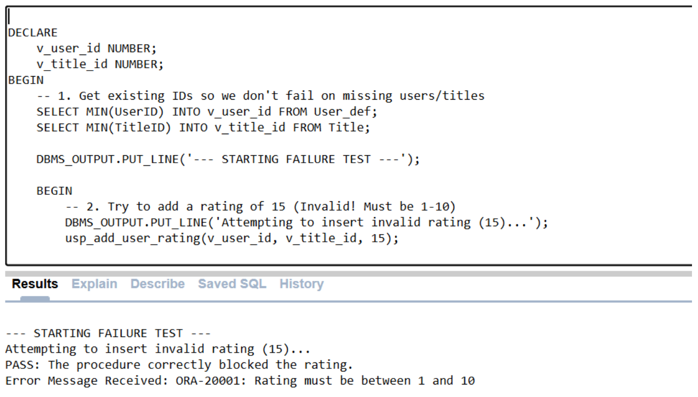


```
DECLARE
    v_user_id NUMBER;
    v_title_id NUMBER;
    v_avg NUMBER;
BEGIN
    -- Get existing IDs
    SELECT MIN(UserID) INTO v_user_id FROM User_def;
    SELECT MIN(TitleID) INTO v_title_id FROM Title;

    DBMS_OUTPUT.PUT_LINE('Testing usp_add_user_rating...');


    usp_add_user_rating(v_user_id, v_title_id, 8);
```

```
Testing usp_add_user_rating...
PASS: Rating added. New Title Average: 8.5
PASS: Procedure correctly blocked invalid rating (11).

Statement processed.
```

FAILURE CASE:

GIVES ERROR BECAUSE RATING IS OUT 0F 10 AND HE IS GIVING 15



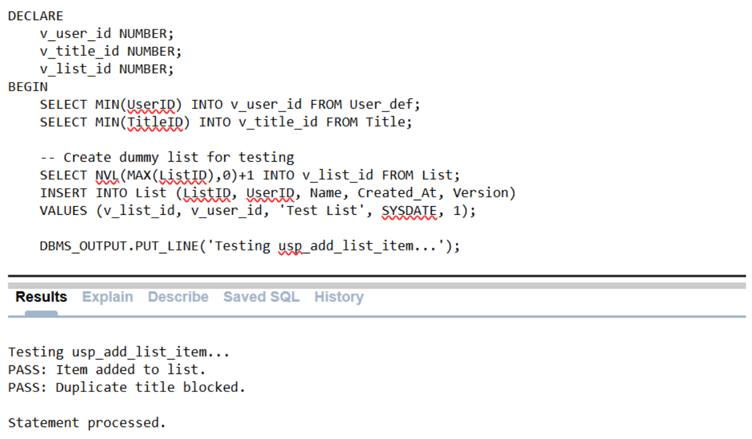
```
DECLARE
    v_user_id NUMBER;
    v_title_id NUMBER;
BEGIN
    -- 1. Get existing IDs so we don't fail on missing users/titles
    SELECT MIN(UserID) INTO v_user_id FROM User_def;
    SELECT MIN(TitleID) INTO v_title_id FROM Title;

    DBMS_OUTPUT.PUT_LINE('--- STARTING FAILURE TEST ---');

    BEGIN
        -- 2. Try to add a rating of 15 (Invalid! Must be 1-10)
        DBMS_OUTPUT.PUT_LINE('Attempting to insert invalid rating (15)...');
        usp_add_user_rating(v_user_id, v_title_id, 15);
```

```
--- STARTING FAILURE TEST ---
Attempting to insert invalid rating (15)...
PASS: The procedure correctly blocked the rating.
Error Message Received: ORA-20001: Rating must be between 1 and 10
```

1.3 **TESTING PROCEDURE:** usp_add_list_item

BOTH SUCCESS AND FAILURE CASE

AS FIRST ADDING A FAVOURITE ITEM IN THE LIST AND THEN AGAIN ADDING IT GIVES THE ERROR



```
DECLARE
    v_user_id NUMBER;
    v_title_id NUMBER;
    v_list_id NUMBER;
BEGIN
    SELECT MIN(UserID) INTO v_user_id FROM User_def;
    SELECT MIN(TitleID) INTO v_title_id FROM Title;

    -- Create dummy list for testing
    SELECT NVL(MAX(ListID),0)+1 INTO v_list_id FROM List;
    INSERT INTO List (ListID, UserID, Name, Created_At, Version)
    VALUES (v_list_id, v_user_id, 'Test List', SYSDATE, 1);

    DBMS_OUTPUT.PUT_LINE('Testing usp_add_list_item...');
```

```
Testing usp_add_list_item...
PASS: Item added to list.
PASS: Duplicate title blocked.

Statement processed.
```

PART 2 OF PROCEDURE

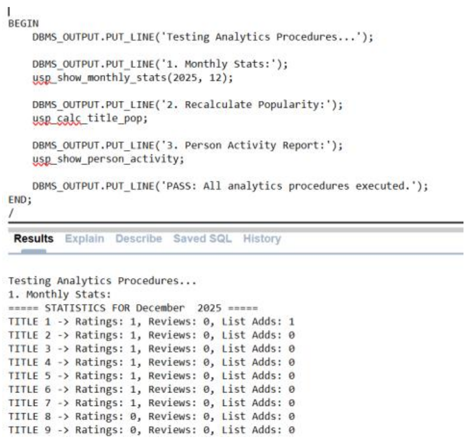
STATISTICS AND ANALYTICAL PROCEDURES

4) usp_generate_monthly_title_statistics

5) usp_recalculate_title_popularity()

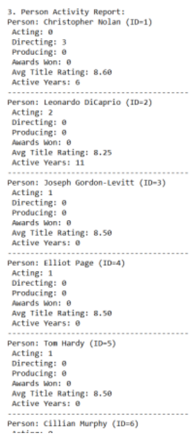
6) usp_generate_person_activity_report

THESE SCREENSHOTS SHOWS THE SUCCESSFUL COMPLETION OF THESE PROCEDURES.
IT IS EQUALLY IMPOSSIBLE THAT THESE PROCEDURES CAN HAVE ERRORS BECAUSE THEY DON'T TAKES ANY ARGUMENTS FROM THE USER.
IT'S WORK IS TO BATCHLY UPDATING THE DATABASE.



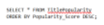
```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Testing Analytics Procedures...');

    DBMS_OUTPUT.PUT_LINE('1. Monthly Stats:');
    usp_show_monthly_stats(2025, 12);

    DBMS_OUTPUT.PUT_LINE('2. Recalculate Popularity:');
    usp_calc_title_pop;

    DBMS_OUTPUT.PUT_LINE('3. Person Activity Report:');
    usp_show_person_activity;

    DBMS_OUTPUT.PUT_LINE('PASS: All analytics procedures executed.');
END;
/
```

```
Testing Analytics Procedures...
1. Monthly Stats:
===== STATISTICS FOR December  2025 =====
TITLE 1 -> Ratings: 1, Reviews: 0, List Adds: 1
TITLE 2 -> Ratings: 1, Reviews: 0, List Adds: 0
TITLE 3 -> Ratings: 1, Reviews: 0, List Adds: 0
TITLE 4 -> Ratings: 1, Reviews: 0, List Adds: 0
TITLE 5 -> Ratings: 1, Reviews: 0, List Adds: 0
TITLE 6 -> Ratings: 1, Reviews: 0, List Adds: 0
TITLE 7 -> Ratings: 1, Reviews: 0, List Adds: 0
TITLE 8 -> Ratings: 0, Reviews: 0, List Adds: 0
TITLE 9 -> Ratings: 0, Reviews: 0, List Adds: 0
```

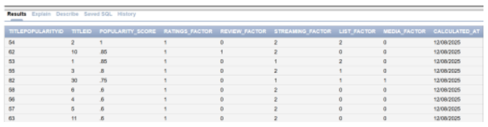


```
3. Person Activity Report:
Person: Christopher Nolan (ID=1)
 Acting: 0
 Directing: 3
 Producing: 0
 Awards Won: 0
 Avg Title Rating: 8.60
 Active Years: 6
----------------------------------------
Person: Leonardo DiCaprio (ID=2)
 Acting: 2
 Directing: 0
 Producing: 0
 Awards Won: 0
 Avg Title Rating: 8.25
 Active Years: 11
----------------------------------------
Person: Joseph Gordon-Levitt (ID=3)
 Acting: 1
 Directing: 0
 Producing: 0
 Awards Won: 0
 Avg Title Rating: 8.50
 Active Years: 0
----------------------------------------
Person: Elliot Page (ID=4)
 Acting: 1
 Directing: 0
 Producing: 0
 Awards Won: 0
 Avg Title Rating: 8.50
 Active Years: 0
----------------------------------------
Person: Tom Hardy (ID=5)
 Acting: 1
 Directing: 0
 Producing: 0
 Awards Won: 0
 Avg Title Rating: 8.50
 Active Years: 0
----------------------------------------
Person: Cillian Murphy (ID=6)
```

ALSO RECALCULATES THE POPULARITY SCORE.



```
SELECT * FROM TitlePopularity
ORDER BY Popularity_Score DESC;
```



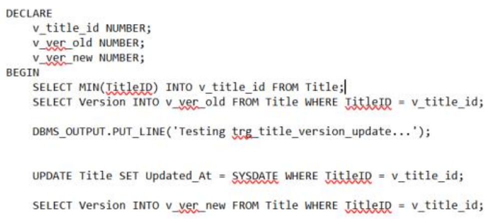
TITLEPOPULARITYID	TITLEID	POPULARITY_SCORE	RATINGS_FACTOR	REVIEW_FACTOR	STREAMING_FACTOR	LIST_FACTOR	MEDIA_FACTOR	CALCULATED_AT
54	2	1	1	0	2	2	0	12/08/2025
62	10	.85	1	1	2	0	0	12/08/2025
53	1	.85	1	0	1	2	0	12/08/2025
55	3	.8	1	0	2	1	0	12/08/2025
82	30	.75	1	0	1	1	1	12/08/2025
58	6	.6	1	0	2	0	0	12/08/2025
56	4	.6	1	0	2	0	0	12/08/2025
57	5	.6	1	0	2	0	0	12/08/2025
63	11	.6	1	0	2	0	0	12/08/2025

This is a batch-processing logic designed for Enterprise Analytics. It is too heavy for a real-time Trigger and too slow for a dynamic View.

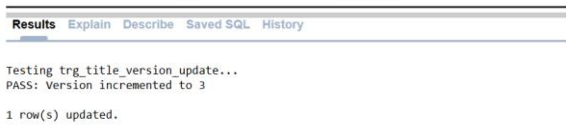
TRIGGERS

--
TESTING TRIGGER 1: trg_title_version_update
--

SUCCESS CASE:

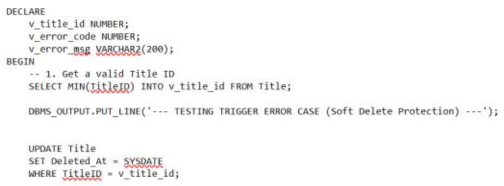


```
DECLARE
    v_title_id NUMBER;
    v_ver_old NUMBER;
    v_ver_new NUMBER;
BEGIN
    SELECT MIN(TitleID) INTO v_title_id FROM Title;
    SELECT Version INTO v_ver_old FROM Title WHERE TitleID = v_title_id;

    DBMS_OUTPUT.PUT_LINE('Testing trg_title_version_update...');


    UPDATE Title SET Updated_At = SYSDATE WHERE TitleID = v_title_id;

    SELECT Version INTO v_ver_new FROM Title WHERE TitleID = v_title_id;
```

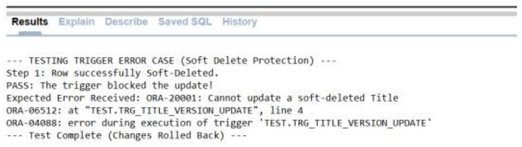


```
Testing trg_title_version_update...
PASS: Version incremented to 3

1 row(s) updated.
```

FAILURE:
THIS THROWS ERROR WHEN A USER TRIES TO UPDATE A RECORD WHICH IS ALREADY IN THE BIN



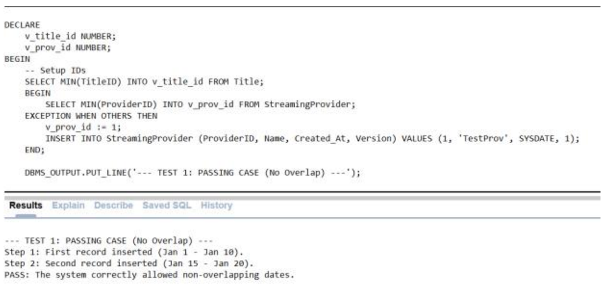
```
DECLARE
    v_title_id NUMBER;
    v_error_code NUMBER;
    v_error_msg VARCHAR2(200);
BEGIN
    -- 1. Get a valid Title ID
    SELECT MIN(TitleID) INTO v_title_id FROM Title;

    DBMS_OUTPUT.PUT_LINE('--- TESTING TRIGGER ERROR CASE (Soft Delete Protection) ---');


    UPDATE Title
    SET Deleted_At = SYSDATE
    WHERE TitleID = v_title_id;
```



```
--- TESTING TRIGGER ERROR CASE (Soft Delete Protection) ---
Step 1: Row successfully Soft-Deleted.
PASS: The trigger blocked the update!
Expected Error Received: ORA-20001: Cannot update a soft-deleted Title
ORA-06512: at "TEST.TRG_TITLE_VERSION_UPDATE", line 4
ORA-04088: error during execution of trigger 'TEST.TRG_TITLE_VERSION_UPDATE'
--- Test Complete (Changes Rolled Back) ---
```

--
TESTING TRIGGER 2: trg_prevent_streaming_overlap
--

SUCCESS:
IF THE STREAMING OF A SAME MOVIE IS NOT HAPPENING TWICE ON SAME RANGE OF STARTING AND ENDING DATE



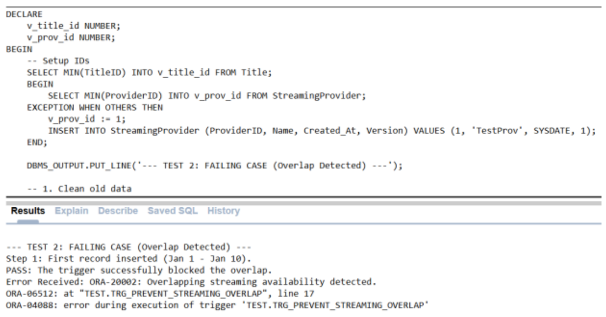
```
DECLARE
    v_title_id NUMBER;
    v_prov_id NUMBER;
BEGIN
    -- Setup IDs
    SELECT MIN(TitleID) INTO v_title_id FROM Title;
    BEGIN
        SELECT MIN(ProviderID) INTO v_prov_id FROM StreamingProvider;
    EXCEPTION WHEN OTHERS THEN
        v_prov_id := 1;
        INSERT INTO StreamingProvider (ProviderID, Name, Created_At, Version) VALUES (1, 'TestProv', SYSDATE, 1);
    END;

    DBMS_OUTPUT.PUT_LINE('--- TEST 1: PASSING CASE (No Overlap) ---');
```

Results Explain Describe Saved SQL History

```
--- TEST 1: PASSING CASE (No Overlap) ---
Step 1: First record inserted (Jan 1 - Jan 10).
Step 2: Second record inserted (Jan 15 - Jan 20).
PASS: The system correctly allowed non-overlapping dates.
```

FAILURE:

IT THROWS ERROR IF STREAMING COLLAPSES



```
DECLARE
    v_title_id NUMBER;
    v_prov_id NUMBER;
BEGIN
    -- Setup IDs
    SELECT MIN(TitleID) INTO v_title_id FROM Title;
    BEGIN
        SELECT MIN(ProviderID) INTO v_prov_id FROM StreamingProvider;
    EXCEPTION WHEN OTHERS THEN
        v_prov_id := 1;
        INSERT INTO StreamingProvider (ProviderID, Name, Created_At, Version) VALUES (1, 'TestProv', SYSDATE, 1);
    END;

    DBMS_OUTPUT.PUT_LINE('--- TEST 2: FAILING CASE (Overlap Detected) ---');

    -- 1. Clean old data
```

Results Explain Describe Saved SQL History

```
--- TEST 2: FAILING CASE (Overlap Detected) ---
Step 1: First record inserted (Jan 1 - Jan 10).
PASS: The trigger successfully blocked the overlap.
Error Received: ORA-20002: Overlapping streaming availability detected.
ORA-06512: at "TEST.TRG_PREVENT_STREAMING_OVERLAP", line 17
ORA-04088: error during execution of trigger 'TEST.TRG_PREVENT_STREAMING_OVERLAP'
```

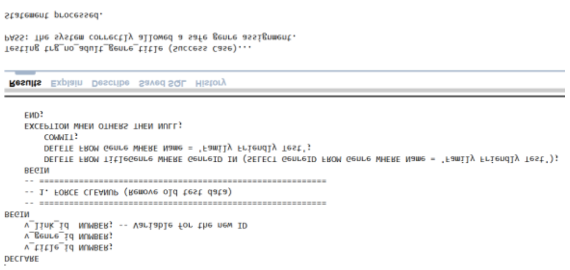
--

TESTING TRIGGER 3: trg_no_adult_genre_title

--

SUCCESS CASE:

ASSIGNMENT OF A NON ADULT MOVIE OR ASSIGNMENT OF ADULT MOVIE TO NON ADULT GENRE OR ADULT GENRE RESPECTIVELY.



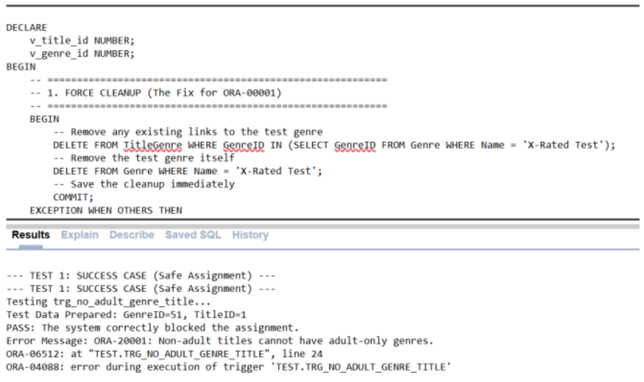
```
DECLARE
    v_title_id NUMBER;
    v_genre_id NUMBER;
    v_link_id  NUMBER; -- Variable for the new ID
BEGIN
    -- ==========================================================
    -- 1. FORCE CLEANUP (Remove old test data)
    -- ==========================================================
    BEGIN
        DELETE FROM TitleGenre WHERE GenreID IN (SELECT GenreID FROM Genre WHERE Name = 'Family Friendly Test');
        DELETE FROM Genre WHERE Name = 'Family Friendly Test';
        COMMIT;
    EXCEPTION WHEN OTHERS THEN NULL;
    END;
```

Results Explain Describe Saved SQL History

```
Testing trg_no_adult_genre_title (Success Case)...
PASS: The system correctly allowed a safe genre assignment.

Statement processed.
```

FAILURE CASE:

ASSIGNMENT OF A NON ADULT MOVIE OR ANY TITLE TO A ADULT GENRE OR VICE VERSA.



```
DECLARE
    v_title_id NUMBER;
    v_genre_id NUMBER;
BEGIN
    -- ==========================================================
    -- 1. FORCE CLEANUP (The Fix for ORA-00001)
    -- ==========================================================
    BEGIN
        -- Remove any existing links to the test genre
        DELETE FROM TitleGenre WHERE GenreID IN (SELECT GenreID FROM Genre WHERE Name = 'X-Rated Test');
        -- Remove the test genre itself
        DELETE FROM Genre WHERE Name = 'X-Rated Test';
        -- Save the cleanup immediately
        COMMIT;
    EXCEPTION WHEN OTHERS THEN
```

Results Explain Describe Saved SQL History

```
--- TEST 1: SUCCESS CASE (Safe Assignment) ---
--- TEST 1: SUCCESS CASE (Safe Assignment) ---
Testing trg_no_adult_genre_title...
Test Data Prepared: GenreID=51, TitleID=1
PASS: The system correctly blocked the assignment.
Error Message: ORA-20001: Non-adult titles cannot have adult-only genres.
ORA-06512: at "TEST.TRG_NO_ADULT_GENRE_TITLE", line 24
ORA-04088: error during execution of trigger 'TEST.TRG_NO_ADULT_GENRE_TITLE'
```

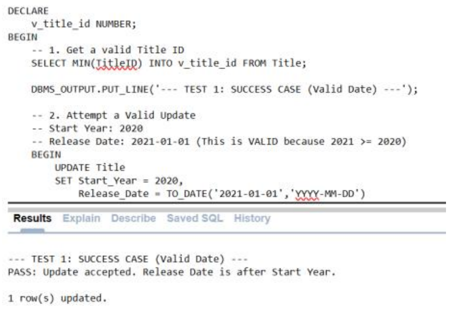
--

TESTING TRIGGER 4: trg_prevent_title_bad_dates

--

SUCCESS CASE :

Scenario: We update a movie to have a Release Date that is *after* the Start Year. Expected Result: The database allows this change because it makes sense (e.g., Started in 2020, Released in 2021).

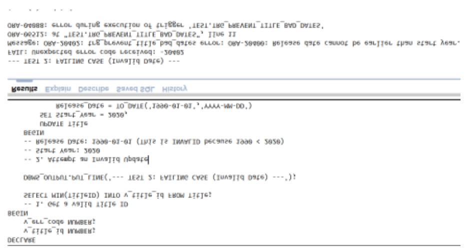


```
DECLARE
    v_title_id NUMBER;
BEGIN
    -- 1. Get a valid Title ID
    SELECT MIN(TitleID) INTO v_title_id FROM Title;

    DBMS_OUTPUT.PUT_LINE('--- TEST 1: SUCCESS CASE (Valid Date) ---');

    -- 2. Attempt a Valid Update
    -- Start Year: 2020
    -- Release Date: 2021-01-01 (This is VALID because 2021 >= 2020)
    BEGIN
        UPDATE Title
        SET Start_Year = 2020,
            Release_Date = TO_DATE('2021-01-01','YYYY-MM-DD')
```

```
--- TEST 1: SUCCESS CASE (Valid Date) ---
PASS: Update accepted. Release Date is after Start Year.

1 row(s) updated.
```

FAILURE CASE:

Scenario: We try to set the Release Date to be *before* the Start Year. Expected Result: The trigger blocks this change and raises error -20400.



```
DECLARE
    v_title_id NUMBER;
    v_err_code NUMBER;
BEGIN
    -- 1. Get a valid Title ID
    SELECT MIN(TitleID) INTO v_title_id FROM Title;

    DBMS_OUTPUT.PUT_LINE('--- TEST 2: FAILING CASE (Invalid Date) ---');

    -- 2. Attempt an Invalid Update
    -- Start Year: 2020
    -- Release Date: 1990-01-01 (This is INVALID because 1990 < 2020)
    BEGIN
        UPDATE Title
        SET Start_Year = 2020,
            Release_Date = TO_DATE('1990-01-01','YYYY-MM-DD')
```

```
--- TEST 2: FAILING CASE (Invalid Date) ---
FAIL: Unexpected error code received: -20402
Message: ORA-20402: trg_prevent_title_bad_dates error: ORA-20400: Release date cannot be earlier than start year.
ORA-06512: at "TEST.TRG_PREVENT_TITLE_BAD_DATES", line 11
ORA-04088: error during execution of trigger 'TEST.TRG_PREVENT_TITLE_BAD_DATES'
```

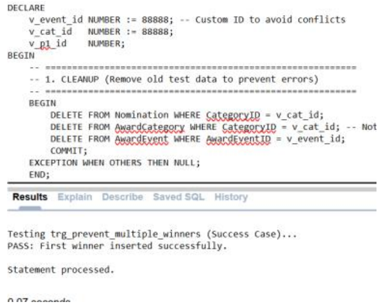
--

TESTING TRIGGER 5: trg_prevent_multiple_winners

--

SUCCESS CASE:

Scenario: A category has no winners yet. We insert the first winner.   Expected Result: The database accepts the entry.



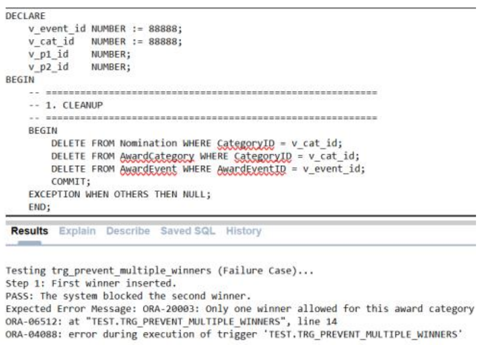
```
DECLARE
    v_event_id NUMBER := 88888; -- Custom ID to avoid conflicts
    v_cat_id   NUMBER := 88888;
    v_p1_id    NUMBER;
BEGIN
    -- ======================================================
    -- 1. CLEANUP (Remove old test data to prevent errors)
    -- ======================================================
    BEGIN
        DELETE FROM Nomination WHERE CategoryID = v_cat_id;
        DELETE FROM AwardCategory WHERE CategoryID = v_cat_id; -- Not
        DELETE FROM AwardEvent WHERE AwardEventID = v_event_id;
        COMMIT;
    EXCEPTION WHEN OTHERS THEN NULL;
    END;
```

```
Testing trg_prevent_multiple_winners (Success Case)...
PASS: First winner inserted successfully.

Statement processed.

0.07 seconds
```

FAILURE:

Scenario: We insert one winner successfully. Then, we try to insert a second winner for the exact same category.

Expected Result: The trigger blocks the second insert with Error -20003.

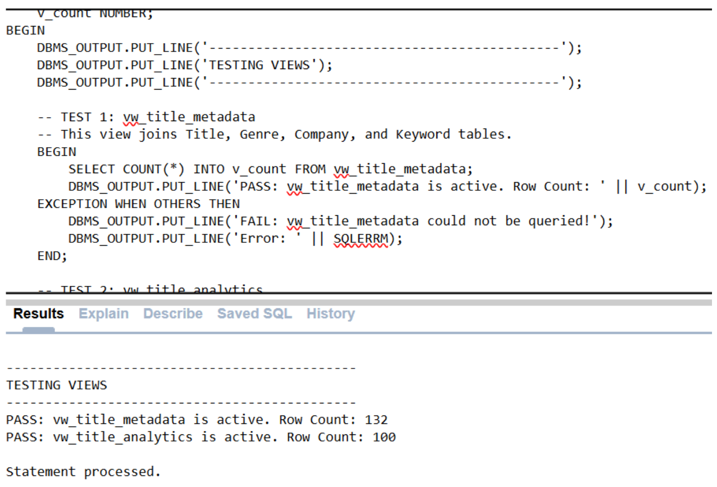


```
DECLARE
    v_event_id NUMBER := 88888;
    v_cat_id   NUMBER := 88888;
    v_p1_id    NUMBER;
    v_p2_id    NUMBER;
BEGIN
    -- ==========================================================
    -- 1. CLEANUP
    -- ==========================================================
    BEGIN
        DELETE FROM Nomination WHERE CategoryID = v_cat_id;
        DELETE FROM AwardCategory WHERE CategoryID = v_cat_id;
        DELETE FROM AwardEvent WHERE AwardEventID = v_event_id;
        COMMIT;
    EXCEPTION WHEN OTHERS THEN NULL;
    END;
```

```
Testing trg_prevent_multiple_winners (Failure Case)...
Step 1: First winner inserted.
PASS: The system blocked the second winner.
Expected Error Message: ORA-20003: Only one winner allowed for this award category
ORA-06512: at "TEST.TRG_PREVENT_MULTIPLE_WINNERS", line 14
ORA-04088: error during execution of trigger 'TEST.TRG_PREVENT_MULTIPLE_WINNERS'
```

VIEWS

TESTING VIEWS : BOTH ARE SUCCESSFULLY RUNNING.   (vw_title_metadata , vw_title_analytics)

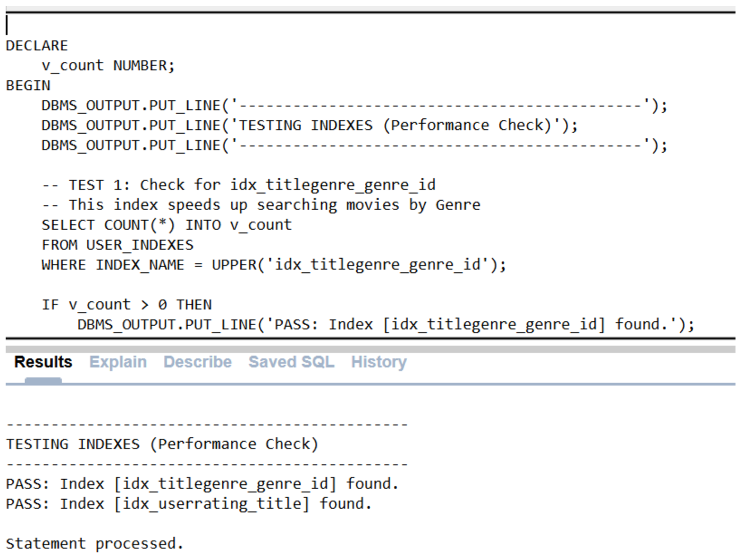


```
    v_count NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('---------------------------------------------');
    DBMS_OUTPUT.PUT_LINE('TESTING VIEWS');
    DBMS_OUTPUT.PUT_LINE('---------------------------------------------');

    -- TEST 1: vw_title_metadata
    -- This view joins Title, Genre, Company, and Keyword tables.
    BEGIN
        SELECT COUNT(*) INTO v_count FROM vw_title_metadata;
        DBMS_OUTPUT.PUT_LINE('PASS: vw_title_metadata is active. Row Count: ' || v_count);
    EXCEPTION WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('FAIL: vw_title_metadata could not be queried!');
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;

    -- TEST 2: vw_title_analytics
```

```
---------------------------------------------
TESTING VIEWS
---------------------------------------------
PASS: vw_title_metadata is active. Row Count: 132
PASS: vw_title_analytics is active. Row Count: 100

Statement processed.
```

Indexes

BOTH INDEXES ARE FOUND .



```
DECLARE
    v_count NUMBER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('---------------------------------------------');
    DBMS_OUTPUT.PUT_LINE('TESTING INDEXES (Performance Check)');
    DBMS_OUTPUT.PUT_LINE('---------------------------------------------');

    -- TEST 1: Check for idx_titlegenre_genre_id
    -- This index speeds up searching movies by Genre
    SELECT COUNT(*) INTO v_count
    FROM USER_INDEXES
    WHERE INDEX_NAME = UPPER('idx_titlegenre_genre_id');

    IF v_count > 0 THEN
        DBMS_OUTPUT.PUT_LINE('PASS: Index [idx_titlegenre_genre_id] found.');
```

```
---------------------------------------------
TESTING INDEXES (Performance Check)
---------------------------------------------
PASS: Index [idx_titlegenre_genre_id] found.
PASS: Index [idx_userrating_title] found.

Statement processed.
```

THE END

LITERALLY, IT TAKES ALOT TIME FOR DEBUGGING.