Advanced Database Management Project

**Chapter 1: Preparing Your Relational Schema**

**Relational Schema Design**

Using the credits.csv and titles.csv files, we will design a relational schema based on database modeling best practices. Here’s how we can structure the schema:

**Relationships and Constraints**

1. **Primary Keys**:
   * Titles.id (unique title identifier)
   * Genres.genre\_id (unique genre identifier)
   * Production\_Countries.country\_id (unique country identifier)
   * People.person\_id (unique person identifier)
   * Roles.role\_id (unique role identifier)
2. **Foreign Keys**:
   * Genres.title\_id → References Titles.id
   * Production\_Countries.title\_id → References Titles.id
   * Roles.title\_id → References Titles.id
   * Roles.person\_id → References People.person\_id
3. **Integrity Constraints**:
   * NOT NULL: Applied to essential columns (e.g., title, release\_year, role).
   * UNIQUE: Ensures unique values for imdb\_id in the Titles table.
   * ON DELETE CASCADE: Ensures dependent rows are removed when a parent row is deleted.

**Populating the Tables**

Here’s the order to populate the tables after you split the data:

1. Populate the Titles table:  
   Import unique title details.
2. Populate the Genres table:  
   Split the genres array into individual rows.
3. Populate the Production\_Countries table:  
   Split the production\_countries array into individual rows.
4. Populate the People table:  
   Extract unique person\_id and name from the Credits table.
5. Populate the Roles table:  
   Map person\_id, title\_id, role, and character from the Credits table.

Code:-

CREATE TABLE Titles ( id VARCHAR2(50) PRIMARY KEY, title VARCHAR2(255) NOT NULL, type VARCHAR2(50) NOT NULL, description CLOB, release\_year NUMBER(4) NOT NULL, age\_certification VARCHAR2(10), runtime NUMBER(5), genres CLOB, production\_countries CLOB, seasons NUMBER(3), imdb\_id VARCHAR2(20), imdb\_score NUMBER(3, 1), imdb\_votes NUMBER, tmdb\_popularity NUMBER, tmdb\_score NUMBER(3, 1) );

CREATE TABLE Credits ( person\_id NUMBER NOT NULL, id VARCHAR2(50) NOT NULL, name VARCHAR2(255) NOT NULL, character VARCHAR2(255), role VARCHAR2(50) NOT NULL, PRIMARY KEY (person\_id, id), FOREIGN KEY (id) REFERENCES Titles(id) );

CREATE TABLE Genres ( genre\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, title\_id VARCHAR2(50) NOT NULL, genre VARCHAR2(50) NOT NULL, FOREIGN KEY (title\_id) REFERENCES Titles(id) ON DELETE CASCADE );

CREATE TABLE Production\_Countries (

country\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

title\_id VARCHAR2(50) NOT NULL,

country VARCHAR2(50) NOT NULL,

FOREIGN KEY (title\_id) REFERENCES Titles(id) ON DELETE CASCADE

);

CREATE TABLE People (

person\_id NUMBER PRIMARY KEY,

name VARCHAR2(255) NOT NULL

);

CREATE TABLE Roles (

role\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

title\_id VARCHAR2(50) NOT NULL,

person\_id NUMBER NOT NULL,

role VARCHAR2(50) NOT NULL,

character VARCHAR2(255),

FOREIGN KEY (title\_id) REFERENCES Titles(id) ON DELETE CASCADE,

FOREIGN KEY (person\_id) REFERENCES People(person\_id) ON DELETE CASCADE

);

Populating the Tables:-

Populate Genres Table:-

INSERT INTO Genres (title\_id, genre)

SELECT id, TRIM(REGEXP\_SUBSTR(genres, '[^,]+', 1, LEVEL))

FROM Titles

CONNECT BY LEVEL <= REGEXP\_COUNT(genres, ',') + 1

AND PRIOR id = id

AND PRIOR DBMS\_RANDOM.VALUE IS NOT NULL;

4,106 rows inserted.

Populate Production\_Countries Table:-

INSERT INTO Production\_Countries (title\_id, country)

SELECT id, TRIM(REGEXP\_SUBSTR(production\_countries, '[^,]+', 1, LEVEL))

FROM Titles

CONNECT BY LEVEL <= REGEXP\_COUNT(production\_countries, ',') + 1

AND PRIOR id = id

AND PRIOR DBMS\_RANDOM.VALUE IS NOT NULL;

1,677 rows inserted.

Populate People Table (From Credits):-

INSERT INTO People (person\_id, name)

SELECT DISTINCT person\_id, name

FROM Credits;

4,540 rows inserted.

Populate Roles Table (From Credits):-

INSERT INTO Roles (title\_id, person\_id, role, character)

SELECT id, person\_id, role, character

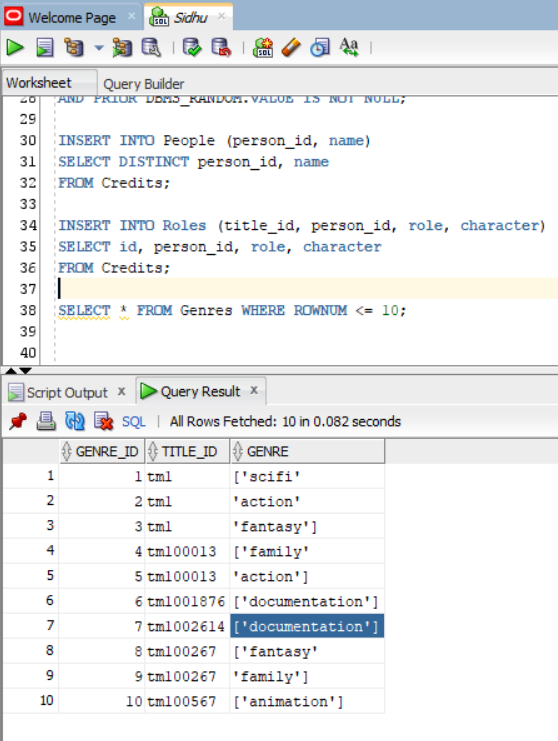
FROM Credits;

5,418 rows inserted

Validate the Data:-

**Check Genres:**

SELECT \* FROM Genres WHERE ROWNUM <= 10;



**Check Production Countries**:

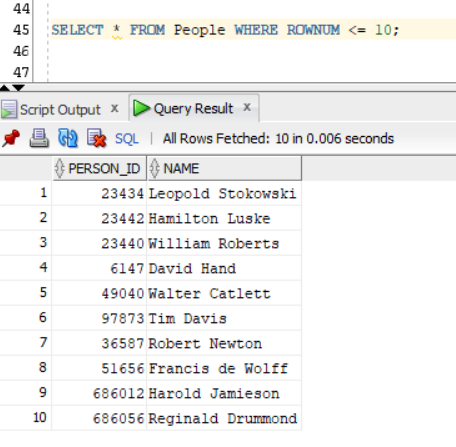
SELECT \* FROM Production\_Countries WHERE ROWNUM <= 10;

A screenshot of a computer

Description automatically generated

**Check People**:

SELECT \* FROM People WHERE ROWNUM <= 10;



**Check Roles**:

SELECT \* FROM Roles WHERE ROWNUM <= 10;

A screenshot of a computer

Description automatically generated

**Chapter 2: Security and User Management:-**

| User | Privileges |
| --- | --- |
| admin\_user | Full database access (DBA, CONNECT, RESOURCE) |
| manager\_user | Create tables, views, sequences (CONNECT, RESOURCE, etc.) |
| regular\_user | Read-only access to Titles and Credits tables |

CREATE USER ##Siddartha\_Admin IDENTIFIED BY "1234";

GRANT CONNECT, RESOURCE, DBA TO ##Siddartha\_Admin;

CREATE USER C##Manager\_Shreeram IDENTIFIED BY "Manager@123";

GRANT CONNECT TO C##Manager\_Shreeram;

GRANT SELECT, INSERT, UPDATE, DELETE ON Titles TO C##Manager\_Shreeram;

GRANT SELECT, INSERT, UPDATE, DELETE ON Credits TO C##Manager\_Shreeram;

CREATE USER C##Abbas\_regular IDENTIFIED BY "User@123";

GRANT CONNECT TO C##Abbas\_regular;

GRANT SELECT ON Titles TO C##Abbas\_regular;

GRANT SELECT ON Credits TO C##Abbas\_regular;

**Password Management**

* Set policies for password complexity and expiration.
* Setting password complexity

ALTER PROFILE DEFAULT LIMIT

PASSWORD\_LIFE\_TIME 30

PASSWORD\_REUSE\_MAX 5;

**Limiting Sensitive Data Access**

* Use Views or Column Masking to restrict access to sensitive data:

SELECT title, type, release\_year

FROM Titles;

GRANT SELECT ON Public\_Titles TO C##Abbas\_regular;

**Verification**

* Testing the users' permissions by logging in as each user and performing allowed action

CONNECT C##Siddartha\_Admin/"1234";

SELECT \* FROM Titles;

INSERT INTO Titles (id, title, type, release\_year) VALUES ('tm99999', 'Test Movie', 'MOVIE', 2024);

1 row inserted.

GRANT Admin\_Role TO C##Siddartha\_Admin;

GRANT Manager\_Role TO C##Manager\_Shreeram;

GRANT Regular\_User\_Role TO C##Abbas\_regular;

REVOKE INSERT, UPDATE, DELETE ON Titles FROM C##Abbas\_regular;

**Chapter 3: Queries and Optimization: -**

1. **Designing SQL Queries:**

Query 1: Find the Top 5 Highest-Rated Movies: -

A screenshot of a computer

Description automatically generated

**2: Count the Number of Movies by Genre**

SELECT g.genre, COUNT(\*) AS movie\_count

FROM Titles t

JOIN Genres g ON t.id = g.title\_id

WHERE t.type = 'MOVIE'

GROUP BY g.genre

ORDER BY movie\_count DESC;

**A screenshot of a computer

Description automatically generated**

**3: Retrieve Movies with the Most Appearances of a Specific Actor**

**A screenshot of a computer

Description automatically generated**

**4: List Directors Who Have Directed More Than 3 Movies**

**A screenshot of a computer

Description automatically generated**

**5: Calculate the Average IMDb Score for Each Year**

**A screenshot of a computer

Description automatically generated**

**6: Find the Most Popular Genre in Each Year**

**A screenshot of a computer

Description automatically generated**

1. **Query Optimization:**
2. Indexing

Create Indexes on Frequently Queried Columns: Indexes speed up search operations:-

CREATE INDEX idx\_titles\_imdb\_score ON Titles(imdb\_score);

CREATE INDEX idx\_genres\_title\_id ON Genres(title\_id);

CREATE INDEX idx\_titles\_release\_year\_genre ON Titles(release\_year, type);

Index IDX\_TITLES\_IMDB\_SCORE created.

Index IDX\_GENRES\_TITLE\_ID created.

Index IDX\_TITLES\_RELEASE\_YEAR\_GENRE created.

**3. Materialized Views**

CREATE MATERIALIZED VIEW mv\_top\_movies\_new AS

SELECT title, imdb\_score, genre

FROM (

SELECT t.title, t.imdb\_score, g.genre

FROM Titles t

JOIN Genres g ON t.id = g.title\_id

WHERE t.type = 'MOVIE'

ORDER BY t.imdb\_score DESC

)

WHERE ROWNUM <= 100;

Materialized view MV\_TOP\_MOVIES\_NEW created.

**4. Query Rewriting & Optimized Using JOIN:**

**A screenshot of a computer program

Description automatically generated**

**Partitioning Tables**

CREATE TABLE Titles\_partitioned (

id VARCHAR2(50) NOT NULL,

title VARCHAR2(255) NOT NULL,

release\_year NUMBER(4),

imdb\_score NUMBER(3, 1),

type VARCHAR2(50),

PRIMARY KEY (id)

)

PARTITION BY RANGE (release\_year) (

PARTITION p1 VALUES LESS THAN (2000),

PARTITION p2 VALUES LESS THAN (2010),

PARTITION p3 VALUES LESS THAN (2020),

PARTITION p4 VALUES LESS THAN (MAXVALUE)

);

Table TITLES\_PARTITIONED created.

**Analyzing Execution Plans:-**

EXPLAIN PLAN FOR

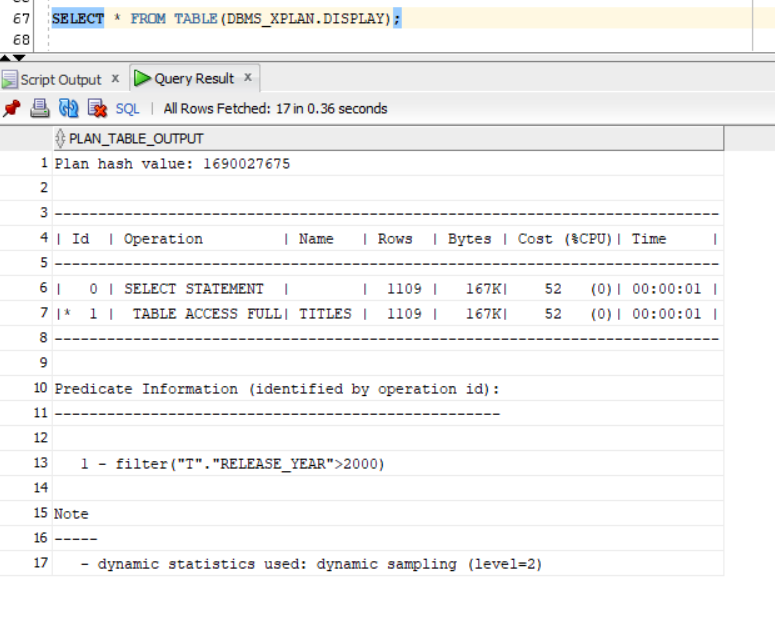
SELECT t.title, t.imdb\_score

FROM Titles t

WHERE t.release\_year > 2000;

Explained.

**View the Plan:**

****

**Avoiding Wildcard Searches: -**

**Inefficient**: Using % at the beginning of a LIKE query disables index use, forcing a full table scan.

**A screenshot of a computer

Description automatically generated**

**Optimized:**

**A screenshot of a computer

Description automatically generated**

**5. Creating Indexes**

Indexes improve query performance by reducing the amount of data scanned.

CREATE INDEX idx\_release\_year ON Titles (release\_year);

CREATE INDEX idx\_genre ON Genres (genre);

Index IDX\_RELEASE\_YEAR created.

Index IDX\_GENRE created

**6.Materialized Views for Precomputed Data**

Materialized views store the results of expensive queries, reducing runtime for repetitive operations.

CREATE MATERIALIZED VIEW mv\_top\_movies AS

SELECT title, imdb\_score

FROM (

SELECT title, imdb\_score

FROM Titles

WHERE type = 'MOVIE'

ORDER BY imdb\_score DESC

)

WHERE ROWNUM <= 100;

Materialized view MV\_TOP\_MOVIES created.

A screenshot of a computer

Description automatically generated

**Chapter 4: Automation of the Information System**

**1. Triggers**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated UPDATE Titles**

**SET title = 'Test Movie'**

**WHERE id = '1';**

**SELECT id, title, last\_updated**

**FROM Titles**

**WHERE id = '1';**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**2. Stored Functions and Procedures**

This procedure generates a monthly report of movies added in the last 30 days by inserting relevant statistics into a Monthly\_Reports table.

**Monthly\_Reports**

CREATE TABLE Monthly\_Reports (

report\_date DATE NOT NULL,

total\_movies NUMBER,

avg\_imdb\_score NUMBER(3, 1)

);

**Check Table Accessibility**

CREATE OR REPLACE PROCEDURE generate\_monthly\_report

IS

BEGIN

INSERT INTO Monthly\_Reports (report\_date, total\_movies, avg\_imdb\_score)

SELECT SYSDATE, COUNT(\*), AVG(imdb\_score)

FROM Titles

WHERE LAST\_UPDATED >= ADD\_MONTHS(SYSDATE, -1);

DBMS\_OUTPUT.PUT\_LINE('Monthly report generated successfully.');

END;

/

**Testing the Procedure**

SHOW ERRORS PROCEDURE generate\_monthly\_report;

EXEC generate\_monthly\_report;

Verify results in the Monthly\_Reports table:

SELECT \* FROM Monthly\_Reports;

16-DEC-24 0

SELECT table\_name FROM user\_tables WHERE table\_name IN ('TITLES', 'MONTHLY\_REPORTS');

DESC Titles;

UPDATE Titles SET LAST\_UPDATED = SYSTIMESTAMP WHERE LAST\_UPDATED IS NULL;

Name Null? Type

-------------------- -------- -------------

ID NOT NULL VARCHAR2(50)

TITLE NOT NULL VARCHAR2(255)

TYPE NOT NULL VARCHAR2(50)

DESCRIPTION CLOB

RELEASE\_YEAR NOT NULL NUMBER(4)

AGE\_CERTIFICATION VARCHAR2(10)

RUNTIME NUMBER(5)

GENRES CLOB

PRODUCTION\_COUNTRIES CLOB

SEASONS NUMBER(3)

IMDB\_ID VARCHAR2(20)

IMDB\_SCORE NUMBER(3,1)

IMDB\_VOTES NUMBER

TMDB\_POPULARITY NUMBER

TMDB\_SCORE NUMBER(3,1)

LAST\_UPDATED TIMESTAMP(6)

1,521 rows updated.