Overview
In this project, you'll create a simple Oracle database & perform some common database operations. The schema of the database is almost the same as that you used in the first project.
Information about getting your Oracle account and general initial configuration is available at: [http://www.cs.purdue.edu/oracle](http://www.cs.purdue.edu/oracle). The department has already setup Oracle accounts for you. You can get your password by logging into the CS portal (top right corner of [http://www.cs.purdue.edu](http://www.cs.purdue.edu)) and clicking on “My Accounts”. As soon as you start working on the project, check that your account exists and you are able to run the sqlplus command in a terminal (make sure you follow the setup instructions at [http://www.cs.purdue.edu/oracle](http://www.cs.purdue.edu/oracle) to get sqlplus to work). Keep in mind you will need to use “x@csora” as the username when logging into your Oracle account (after entering the sqlplus command), where “x” is your Purdue username. If you have problems with your account, please email oracle@cs.purdue.edu or software@cs.purdue.edu as appropriate to get the problem fixed ASAP. If you would like to work on the project remotely, you can ssh to any of the lab machines.

Step 1: Create the tables
The schema for the database is as follows:
1. ACTOR(actor_name: string, gender: string, date_of_birth: date)
2. MOVIE(movie_title: string, release_year: integer, genre: string, movie_length: integer)
3. CAST_MEMBER(actor_name: string, movie_title: string, release_year: integer, actor_role: string)
4. AWARDS_EVENT(event_name: string, event_year: integer, venue: string)
5. NOMINATION(event_name: string, event_year: integer, movie_title: string, release_year: integer, category: string, won: string)
6. DB_USER(userid: string)
7. MOVIE_RATING(userid: string, movie_title: string, release_year: integer, rating: real)

The underlined attributes are primary key attributes.

The meaning of these relations is straightforward:

- **ACTOR** contains one record per actor/actress identified by actor_name, the gender attribute has a value of either ‘Female’ or ‘Male’.
- **MOVIE** contains one record per movie uniquely defined by its movie_title and release_year.
• *CAST_MEMBER* contains a record for each actor/actress who is a cast member in the movie identified by *movie_title* and *release_year*; *actor_role* is the role of the actor in the movie.

• *AWARDS_EVENT* contains one record per award event uniquely identified by the *event_name* and *event_year*.

• *NOMINATION* contains one record for each movie nomination at the awards event identified by *event_name* and *event_year*; the *category* attribute gives the nomination category and the *won* attribute specifies whether the movie won an award for this particular nomination (the value for this attribute is either ‘Yes’ or ‘No’).

• *DB_USER* contains one record for each user of the movie database, identified by *userid*.

• *MOVIE_RATING* contains one record for each user-movie pair, uniquely identified using *userid*, *movie_title* and *release_year*; *rating* gives the rating for that movie as provided by this specific user of the database.

Create *all the key and referential integrity constraints necessary to model the application*. Make sure that your attribute & table names correspond to those listed above. For this project, we’re not asking you to create domain constraints or indices on any of the tables (although you will not lose any points for creating them).

**Your task:** Create a file called tables.sql, which contains seven create ... statements corresponding to the tables listed above.

If you’re in the directory containing tables.sql, you can create your database tables as follows:

```
$ sqlplus
...
SQL> @ tables.sql
```

Remember the ‘@’ operator forces SQL to execute commands from a file.

**Step 2: Read data files**

You are given a sample data file (data.sql) that you can use to populate your database for your own tests.

**Your task:** Check that all data insertion runs without any problem.

If you’re in the directory containing data.sql, you can insert the given data as follows:

```
$ sqlplus
...
SQL> @ data.sql
```
Step 3: Query your database

Queries:
Write SQL queries that answer the questions below (one query per question) and run them on the Oracle system. The query answers should be duplicate-free, but you should use `distinct` only when necessary. If you are making any assumptions, state them clearly, and document your queries. We will run your queries on a different dataset from the one provided with the assignment, so be careful not to hardcode values to produce correct answers 😊

1. For each movie, print the movie title, release year and the number of female cast members in that movie. Print 0 as the number of female cast members if a movie has no female cast members.
2. Print the name(s) and birth date(s) of the actress(es) with the greatest number of total nominations among all actresses (Note: An actress is nominated for a movie if her role in the movie is ‘lead actress’ and the movie has a nomination record with the category ‘best lead actress’ or if the actress has the role ‘supporting actress’ and the movie has a nomination record with the category ‘best supporting actress’. Note that the roles and nomination categories here are case sensitive).
3. Print the movie title, release year and the total number of nominations of movies with the greatest number of total nominations among the movies of genre ‘Comedy’ (the movie itself should have the genre ‘Comedy’ as well).
4. Print the name(s) of actor(s)/actress(es) who have attended award events at all venues in the database (Note: An actor/actress attends an award event if a movie in which he/she is a cast member (not necessarily lead/supporting actor/actress) is nominated at that award event).
5. Print the titles and release years of movies, which have either won more than 2 awards or have more than 2 cast members, but not both.
6. Find the id(s) of database users who have submitted the highest rating for the movie that has the lowest average rating among all movies. Print the userid, movie title, release year and the rating submitted by the user.
7. For each movie genre, print the genre, title and release year of the movie with the greatest length among all movies of that genre (if there is only one movie for a particular genre, include that movie in the results). Sort by increasing order of length.

General Hints:
- When you’re using aggregate functions for a specific group of records, the `GROUP BY` operator should be used in the query.
- There might be multiple ways to form a query and all of them will be acceptable as long as they result in the same record list.
- Ordering of results will not matter if the query does not explicitly tell you to sort the results.
You can use the keyword MINUS for set difference in Oracle SQL.

**Your task:** Create a file called queries.sql, which contains the queries listed above, **in the order they are listed** (jumbling the order of queries may cost you points). Before each query $i$, please put the following comment: rem Query $i$. If you are not able to provide a specific query, just type “rem Query $i$,” where $i$ is the query number. So, your file should look something like this:

rem Query 1
select ...

rem Query 2
select ....
...

Don’t forget to add a semicolon at the end of each query in the file, which is what actually runs them.

**Step 4: Views**
Here, you'll create some simple views.
Create two views (Please name them VIEWA and VIEWB) and print their contents.
- A. A view that shows the actor/actress name followed by the number of movies in which he/she is a cast member, ordered by actor name. Print 0 if the number of movies is 0.
- B. A view that shows the names of nominees (actors/actresses) at awards events, sorted by actor name. The view should contain six fields: actor name, event name, event year, movie title, release year and nomination category.

**Your task:** Create a file called views.sql, which contains SQL commands (create view FOO as ...) to create the views listed above and the SELECT statements that list all the data of both views, i.e. your file should contain two view creation statements followed by two query statements (all of which are ended with semicolons).

**Step 5: PL/SQL Procedures**
In this part you are going to use PL/SQL (Oracle's procedural extension to SQL) to write procedures to process data.

Create a file named procedures.sql. The first line of this file should be:

```
set serveroutput on size 32000
```

Your file should contain code to create and run three procedures: pro_actor_stats, pro_histogram, pro_add_rating. The description of each procedure is provided below.

Your file should look something like this:
/* create the procedure */
create or replace procedure pro_actor_stats as

/* declarations */
begin
  /* code */
end;

/

/* actually run the procedure */
begin
  pro_actor_stats;
end;

/

create or replace procedure pro_histogram as
begin
  /* code */
end;

/

begin
  pro_histogram;
end;

/
...

Procedures:

1. **pro_actor_stats**: Generate a report about the total number of movies in which an actor is a cast member. Sort results by actor name. The number of movies for each actor should be marked with an X under the corresponding bin.
   
   You should create 3 equal-sized bins for the number of movies based on the minimum and maximum number of movies for any actor. For example, if the minimum number of movies for any actor is 0 and the maximum is 6, the bins created should be those in the sample output below. If the difference between the minimum and maximum number of movies is not divisible by 3, your output should include an extra bin for the remainder of the numbers. For example, if the minimum is 0 and maximum is 5, the categories would be {0}, {>0, <=1}, {>1, <=2}, {>2, <=3}, {>3}. If the minimum is 2 and maximum is 12, the categories would be {2}, {>2, <=5}, {>5, <=8}, {>8, <=11}, {>11}. This means your output will have either 4 or 5 categories for the number of movies. You can assume that the difference between
the minimum and maximum number of movies will always be greater than or equal to 3.
The maximum number of characters in an actor name will not exceed 30 and the total number of movies for each actor will not have more than two digits, so you can format your output accordingly. Make sure the X's in your output align with the bins of movies corresponding to each actor.

**Sample output:**

<table>
<thead>
<tr>
<th>Actor name</th>
<th># Movies:</th>
<th>0</th>
<th>&gt;0, &lt;= 2</th>
<th>&gt;2, &lt;= 4</th>
<th>&gt;4, &lt;=6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brad Pitt</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tom Cruise</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uma Thurman</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. **pro_histogram:** Generate a histogram for the number of nominations of movies. Include all discrete integer values in the range min(number of nominations) .. max(number of nominations). Mark the **statistical median** on the graph (in the sample output below, it’s 2). [Aside: Do you really know the definition of median? What if the size of your input set is even? Be careful on this one.] Only mark the median if it’s an integer value.

**Sample output:**

<table>
<thead>
<tr>
<th>#nominations</th>
<th># movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

3. **pro_add_rating:** Write a procedure to add a rating for a movie. The input parameters will be as follows: userid_in, movie_title_in, release_year_in, rating_in. You can also assume that the given user and movie already exist in the database and the rating to be added will not be a duplicate. The result will be a new movie rating record added to the database.
Execute this procedure to insert the rating (user1, American Hustle, 2013, 7) to the database. Do a select * from Movie_Rating before and after this procedure is run.
(i.e. include a query that retrieves the whole content of the Movie_Rating table before and after the statements for executing this procedure to insert data).

Running the Procedures:
You can run your procedures file by typing ‘@procedures.sql’ after executing the sqlplus command in a terminal. If there are errors in creating the procedures, you can see the errors by typing the command “show errors;”. Note that the sample outputs provided above are not the sample outputs for data.sql, they are just to give you an idea about the formatting.

Evaluation:
Your project will mostly be evaluated based on the correctness of your output for the queries, views and procedures. Some points will also be allocated for the proper creation of the database tables. Make sure you comply with the database schema provided above when creating the tables and writing the queries. If your table creation fails somehow, we will test your queries on a database populated with the correct tables and the attribute names in your queries have to match those given in the assignment to produce correct output.

Submission Instructions:

Please create a README file that contains identifying information. For example:

CS348 - Project 2
Author: John Doe
Login: jdoe
Email: jdoe@cs.purdue.edu

Include here anything you might want us to know when grading your project.

To turn in your project, ssh to lore.cs.purdue.edu, create a folder named project2 in your home directory and copy your .sql files and your README.txt to that folder.

After copying your files (tables.sql, queries.sql, views.sql, procedures.sql) in the folder project2, execute the following command in your home directory:

```bash
turnin -c cs348 -p proj2 project2
```

To verify the contents of your submission, execute the following command right after submission:

```bash
turnin -c cs348 -p proj2 --v
```
Resources:
You can find additional information about creating stored procedures/functions in PL/SQL on the following websites.