CIS560 – Database System Concepts **Name:_____**

SQL Assignment 1 (20 points) – due Friday, September 6th at 11:59PM

***Collaboration policy**: This is an individual assignment. You are allowed to discuss the assignment with your colleagues, but your submission should reflect your own work. Sharing or copying code is not permitted.*

In this assignment, you are asked to create a relational **movie** database, populate the database and practice SQL queries on this database. The **movie** database consists of six tables:

movie_info (movie_id, movie_name, year, rating)
actor_ids (actor_id, actor_name, gender)
actor_movies (actor_id, movie_id)
producer_ids (producer_id, producer_name)
producer_movies (producer_id, movie_id)
genre(movie_id, genre)

actor_movies.actor_id refers to **actor_ids.actor_id**
actor_movies.movie_id refers to **movie_info.movie_id**

producer_movies.producer_id refers to **producer_ids.producer_id**
producer_movies.movie_id refers to **movie_info.movie_id**

genre.movie_id refers to **movie_info.movie_id**

We will use the following syntax to create tables in MySQL:

```
CREATE TABLE IF NOT EXISTS `example` (
`example_index` varchar(100) NOT NULL,
`sample_id` varchar(100) NOT NULL,
`examples` varchar(100) NOT NULL,
PRIMARY KEY (`example_index`),
FOREIGN KEY (`sample_id`) references sample_info(`sample_id`) ON
DELETE CASCADE
) ENGINE=InnoDB;
```

Note: There are several engines that MySQL can use and each has different properties. We will be using ENGINE=InnoDB in this class, as this type of engine supports transactions, row-level locking and foreign keys. You can read about other engines in MySQL at: http://dev.mysql.com/doc/refman/5.0/en/storage-engines.html

SQL scripts for creating and populating the database are provided in the files `TableCreation.sql` and `MovieData.sql`, available on KSOL.

1. (2p) Log-in to `phpmyadmin`. Select your database on the left hand side (your database has your account name). Use the `Import` tab to import the `TableCreation.sql` and `MovieData.sql` files, for creating tables and importing data. (*Note: tables must be created before data can be imported.*) Run a COUNT query for each relation to show the total number of tuples loaded.

   ```
   Select * from TABLENAME;
   ```

2. (1p) Find all movies with your favorite star (actor/actress). Your query should return movie name and rating.

   ```
   SELECT movie_name, rating
   FROM movie_info, actor_ids,actor_movies
   WHERE actor_ids.actor_id = actor_movies.actor_id AND
   actor_movies.movie_id = movie_info.movie_id AND actor_name =
   'XXX';
   ```

3. (1p) List all the producers who produced a *'Film-Noir'* movie in a leap year. (You need to check that the genre is *'Film-Noir'* and year is divisible by 4.) Your query should return the producer name, movie name, and year.

   ```
   SELECT producer_name, movie_name, year
   FROM movie_info, producer_ids, producer_movies, genre
   WHERE producer_ids.producer_id = producer_movies.producer_id
   AND genre.movie_id = producer_movies.movie_id
   AND producer_movies.movie_id = movie_info.movie_id
   AND genre = 'Film-Noir'
   AND year %4 =0
   ```

   or

   ```
   SELECT producer_name, movie_name, year
   FROM movie_info, producer_ids, producer_movies, genre
   WHERE producer_ids.producer_id = producer_movies.producer_id
   AND genre.movie_id = producer_movies.movie_id
   AND genre.movie_id = movie_info.movie_id
   AND genre = 'Film-Noir'
   AND year %4 =0
   ```

4. (1p) List, in alphabetical order, the names of all the actors who played in the movie 'alien: resurrection'.

```
SELECT actor_name
FROM actor_movies, actor_ids, movie_info
WHERE movie_info.movie_name = 'alien: resurrection'
AND actor_movies.actor_id = actor_ids.actor_id
AND movie_info.movie_id = actor_movies.movie_id
ORDER BY actor_name
```

5. (2p) List all producers who produced 10 movies or more, in descending order of the number of movies they produced. Return the producers' names and the number of movies each of them produced.

```
SELECT PID.producer_name, count(PM.movie_id)
FROM producer_ids PID, producer_movies PM
WHERE PID.producer_id = PM.producer_id
GROUP BY PID.producer_name
HAVING count(PM.movie_id)>=10
ORDER BY count(PM.movie_id) DESC
```

6. (2p) Find the movies which had both ‘spacey,kevin’ and ‘zachary,emily’ in their cast.

   We use two tuple variables, one for each actor.

```
SELECT x.movie_name
FROM movie_info AS x, actor_ids AS a, actor_ids AS b,
actor_movies AS y, actor_movies AS z
WHERE a.actor_name = 'spacey,kevin'
AND b.actor_name = 'zachary,emily'
AND x.movie_id = y.movie_id
AND x.movie_id = z.movie_id
AND a.actor_id = y.actor_id
AND b.actor_id = z.actor_id
```

7. (3p) How many movies had ‘spacey,kevin’ but not ‘zachary,emily’ in their cast?

```
SELECT count(*)
FROM actor_movies a, actor_ids b
WHERE a.actor_id = b.actor_id
 AND b.actor_name = 'spacey,kevin'
 AND NOT EXISTS
(
 SELECT c.movie_id
 FROM actor_movies c, actor_ids d
 WHERE c.actor_id = d.actor_id
       AND d.actor_name = 'zachary,emily'
            AND a.movie_id= c.movie_id
)
```

or

Search for movies that have `'spacey,kevin'` as an actor, and do not belong to the set of movies that have `'zachary,emily'` as an actor.

```
SELECT count(*)
FROM actor_movies a, actor_ids b
WHERE a.actor_id = b.actor_id
    AND b.actor_name = 'spacey,kevin'
    AND a.movie_id NOT IN
(
 SELECT c.movie_id
 FROM actor_movies c, actor_ids d
 WHERE c.actor_id = d.actor_id
       AND d.actor_name = 'zachary,emily'
)
```

8. (2p) Find all actors that played in at least two different movies during the year 2005.

```
SELECT x.actor_name, COUNT(*) AS Movie_Count
FROM actor_ids AS x, actor_movies AS y, movie_info AS z
WHERE x.actor_id = y.actor_id
AND y.movie_id = z.movie_id
AND z.year =2005
GROUP BY x.actor_name
HAVING Movie_Count >=2
```

9. (2p) Find the number of movies by genre, for movies released in between 2005 and 2010.

```
SELECT a.genre, count( * )
FROM genre a, movie_info b
WHERE a.movie_id = b.movie_id
AND b.year BETWEEN 2005 AND 2010
GROUP BY a.genre
```

10. (4p) (a) For each year, count the number of movies in that year that had only female actors.

```
SELECT m.year, count( m.year )
FROM movie_info m, actor_movies a, actor_ids b
WHERE a.actor_id = b.actor_id
AND a.movie_id = m.movie_id
```

```
AND NOT EXISTS (
  SELECT d.actor_id
  FROM actor_movies d, actor_ids e
  WHERE d.actor_id = e.actor_id
  AND e.gender = 'male'
  AND m.movie_id = d.movie_id
)
GROUP BY m.year
```

(b) Now make a small change: for each year, report the percentage of movies with only female actors made that year, and also the total number of movies made that year. For example, one answer can be:

2011 10.81 135

meaning that in 2011 there were 135 movies, and 10.81% had only female actors.

Within your query, construct a table NumMoviesByYear that contains the total number of movies each year. Use the counts from this table to calculate the percentage (make sure the years match). Everything else is similar to what we had in a).

```
SELECT m.year, ((count(m.year)/NumMoviesByYear.count)*100) AS
percentage, NumMoviesByYear.count AS TotalNum_Movies
FROM movie_info m, actor_movies a, actor_ids b, (
    SELECT y.year, count(*) AS count
    FROM movie_info AS y
    GROUP BY y.year
    ) AS NumMoviesByYear
WHERE a.actor_id = b.actor_id
AND a.movie_id = m.movie_id
AND NOT EXISTS (
   SELECT d.actor_id
   FROM actor_movies d, actor_ids e
   WHERE d.actor_id = e.actor_id
   AND e.gender = 'male'
   AND m.movie_id = d.movie_id
     )
AND m.year = NumMoviesByYear.year
GROUP BY m.year
```

What to turn it: A *.txt* file with all SQL queries and the result of each query (if there are too many tuples in a result, show only the first result page). Submit this file using the course Dropbox.