SQL Review

Create Table with Foreign Keys

```
CREATE TABLE Studios(
    name VARCHAR(20) PRIMARY KEY,
    website VARCHAR(255)
);
```

```
CREATE TABLE Stars (
    name VARCHAR(20) PRIMARY KEY,
    gender CHAR(1),
    birthyear INT,
    birthplace VARCHAR(40)
);
```

```
CREATE TABLE Movies (
    title VARCHAR(50),
    year INT,
    length INT,
    rating CHAR(2),
    studioname VARCHAR(20) REFERENCES Studios(name) ON DELETE CASCADE,
    PRIMARY KEY (title, year)
);
```

```
CREATE TABLE StarsIn (
     title VARCHAR(50),
     year INT,
     starname VARCHAR(20),
     PRIMARY KEY (title,year,starname),
     FOREIGN KEY (title,year) REFERENCES Movies(title,year) ON DELETE CASCADE,
     FOREIGN KEY (starName) REFERENCES Stars(name) ON DELETE CASCADE
);
```

Insert – Studios

```
INSERT INTO Studios
VALUES('Fox', 'foxmovies.com');

INSERT INTO Studios
VALUES('Disney', 'disney.com');

INSERT INTO Studios
VALUES('Paramount', 'www.paramount.com');
```

Insert – Movies

```
INSERT INTO Movies
VALUES('Walk the Line', 2005, 136, 'PG', 'Fox');

INSERT INTO Movies
VALUES('Pretty Woman', 1990, 119, 'R', 'Disney');

INSERT INTO Movies
VALUES('Wayne''s World', 1991, 104, 'PG', 'Paramount');

INSERT INTO Movies
VALUES('Unfaithful', 2002, 124, 'R', 'Fox');

INSERT INTO Movies
VALUES('Runaway Bride', 1999, 116, 'PG', 'Paramount');

INSERT INTO Movies
VALUES('The Princess and the Frog', 2009, 97, 'G', 'Disney');
```

Insert – Stars

```
INSERT INTO Stars
VALUES('Richard Gere', 'M', 1949, 'Philadelphia, Pennsylvania, USA');

INSERT INTO Stars
VALUES('Joaquin Phoenix', 'M', 1974, 'San Juan, Puerto Rico');

INSERT INTO Stars
VALUES('Reese Witherspoon', 'F', 1976, 'Baton Rouge, Louisiana, USA');

INSERT INTO Stars
VALUES('Julia Roberts', 'F', 1967, 'Smyrna, Georgia, USA');

INSERT INTO Stars
VALUES('Mike Myers', 'M', 1963, 'Scarborough, Ontario, Canada');

INSERT INTO Stars
VALUES('Oprah Winfrey', 'F', 1954, 'Kosciusko, Mississippi, USA');
```

Insert – StarsIn

```
INSERT INTO StarsIn VALUES('Walk the Line', 2005, 'Joaquin Phoenix');
INSERT INTO StarsIn VALUES('Walk the Line', 2005, 'Reese Witherspoon');

INSERT INTO StarsIn VALUES('Pretty Woman', 1990, 'Richard Gere');
INSERT INTO StarsIn VALUES('Pretty Woman', 1990, 'Julia Roberts');

INSERT INTO StarsIn VALUES('Wayne''s World', 1991, 'Mike Myers');

INSERT INTO StarsIn VALUES('Unfaithful', 2002, 'Richard Gere');

INSERT INTO StarsIn VALUES('Runaway Bride', 1999, 'Richard Gere');
INSERT INTO StarsIn VALUES('Runaway Bride', 1999, 'Julia Roberts');

INSERT INTO StarsIn VALUES('The Princess and the Frog', 2009, 'Oprah Winfrey');
```

Creation and insertion order

1. **Movies** after **Studios**

2. **StarsIn** after **Movies** and **Stars**

Ordering the Input

Example. Find the Disney movies and list them by length, shortest first.

```
SELECT *
FROM Movies
WHERE studioName = 'Disney'
ORDER BY length;
```

Example. Find the Disney movies and list them by length, shortest first, and among movies of equal length, sort alphabetically.

```
SELECT *
FROM Movies
WHERE studioName = 'Disney'
ORDER BY length, title;
```

Remarks

- Ordering is ascending, unless you specify the DESC keyword after **an attribute**.
- Ties are broken by the second attribute on the ORDER BY list, etc.

Natural Join with USING

Better than NATURAL JOIN:

```
SELECT *
FROM Movies JOIN StarsIn USING (title,year);
```

Because now it is explicit which attributes are used to join the tables.

Join with ON

A similar result can be obtained by:

```
SELECT *
FROM Movies JOIN StarsIn ON
                    Movies.title=StarsIn.title AND
                    Movies.year=StarsIn.year;
```

Exactly same as:

```
SELECT *
FROM Movies, StarsIn
WHERE     Movies.title=StarsIn.title AND
          Movies.year=StarsIn.year;
```

Outer Joins

```
SELECT *
FROM Movies FULL OUTER JOIN StarsIn USING(title,year);
```

One of LEFT, RIGHT, or FULL before OUTER (but not missing).

- LEFT = pad dangling tuples of Movies only.
- RIGHT = pad dangling tuples of StarsIn only.
- FULL = pad both.

Union/Intersection/Difference

```
SELECT title, year
FROM StarsIn
WHERE starName='Richard Gere'

  UNION / INTERSECT / EXCEPT  (use one of them depending on request)

SELECT title, year
FROM StarsIn
WHERE starName='Julia Roberts' ;
```

Aliases

Find pairs of stars who have played together in the same movie.

```
SELECT S1.starname, S2.starname
FROM StarsIn S1, StarsIn S2
WHERE S1.title = S2.title AND S1.year = S2.year
        AND S1.starname < S2.starname;
```

Grouping

```
SELECT studioName, AVG(length)
FROM Movies
GROUP BY studioName;
```

Another Example

From **Movies** and **StarsIn**, find the star's total length of film played.

```
SELECT starName, SUM(length)
FROM Movies, StarsIn
WHERE Movies.title=StarsIn.title AND Movies.year=StarsIn.year
GROUP BY starName;
```

Compute those tuples first, then group by starName.

HAVING Clauses

```
SELECT starName, SUM(length)
FROM Movies, StarsIn
WHERE Movies.title=StarsIn.title AND Movies.year=StarsIn.year
GROUP BY starName
HAVING MIN(StarsIn.year) < 2000;
```

Requirements on HAVING Conditions

- They may refer to attributes that make sense within a group; i.e., they are either:
  1. Grouping attribute, or
  2. Aggregated attribute.

Restriction on SELECT Lists With Aggregation

- If any aggregation is used, then each element of the SELECT list must be either:
  1. Aggregated, or
  2. An attribute on the GROUP BY list.

Exercise

Using **Movies**, **StarsIn**, and **Stars**,

find the star's total length of film played.

We are interested only in Canadian stars and

who first appeared in a movie before 2000.

```
SELECT starName, SUM(length)
FROM Movies, StarsIn, Stars
WHERE Movies.title=StarsIn.title AND Movies.year=StarsIn.year
      AND Stars.name=StarsIn.starName
      AND Stars.birthplace LIKE '%Canada%'
GROUP BY starName
HAVING MIN(StarsIn.year) < 2000;
```

Correlated Subqueries

- Suppose StarsIn table has an additional attribute “salary”

  StarsIn(movie, movie, starName, salary)

  Now, find the stars who were paid for some movie more than the average salary for that movie.

```
SELECT starName, title, year
FROM StarsIn X
WHERE salary >
        (SELECT AVG(salary)
        FROM StarsIn
        WHERE title = X.title AND year=X.year);
```

Remark

Semantically, the value of the X tuple changes in the outer query, so the database must rerun the subquery for each X tuple.

Another Solution (Nesting in FROM)

```
SELECT X.starName, X.title, X.year
    FROM StarsIn X, (SELECT title, year, AVG(salary) AS avgSalary
                     FROM StarsIn
                     GROUP BY title, year) Y
    WHERE X.salary>Y.avgSalary AND
                X.title=Y.title AND X.year=Y.year;
```

Views

- A view is a “*virtual table*”, a relation that is defined in terms of the contents of other tables and views.

Example

```
CREATE VIEW DMovies AS
    SELECT title, year, length, rating
    FROM Movies
    WHERE studioName = 'Disney';
```

Constraints – mutually exclusive subclasses

```
CREATE TABLE Vehicles (
  vin CHAR(17) PRIMARY KEY,
  vehicle_type CHAR(3) CHECK(vehicle_type IN ('SUV', 'ATV')),
  fuel_type CHAR(4),
  door_count INT CHECK(door_count >= 0),
  UNIQUE(vin, vehicle_type)
);

CREATE TABLE SUVs (
  vin CHAR(17) PRIMARY KEY,
  vehicle_type CHAR(3) CHECK(vehicle_type ='SUV'),
  FOREIGN KEY (vin, vehicle_type) REFERENCES Vehicles (vin, vehicle_type)
);

CREATE TABLE ATVs (
  vin CHAR(17) PRIMARY KEY,
  vehicle_type CHAR(3) CHECK(vehicle_type ='ATV'),
  FOREIGN KEY (vin, vehicle_type) REFERENCES Vehicles (vin, vehicle_type)
);
```

```sql
CREATE VIEW ParamountMovies AS
    SELECT title, year
    FROM Movies
    WHERE studioName = 'Paramount'
WITH CHECK OPTION;
```

```sql
INSERT INTO ParamountMovies VALUES ('Star Trek', 1979);
```

Insertion fails!

Why?

Rationale for this behavior is:

- Were the insertion allowed, it would insert a tuple with NULL for studioName in base table Movie.
- However, such a tuple doesn't satisfy the condition for being in the ParamountMovie view! ("**invisible tuple to the view**")
- Thus, it shouldn't be allowed to get into the database through the ParamountMovie view.

Another example: Cars

```sql
-- Another example: Cars
CREATE TABLE cars (
   car_id serial primary key,
   car_name varchar(255),
   brand VARCHAR(20)
);

CREATE VIEW audi_cars AS
  SELECT car_id, car_name, brand
  FROM cars
  WHERE brand = 'Audi'
WITH CHECK OPTION;

CREATE VIEW ford_cars AS
  SELECT car_id, car_name, brand
  FROM cars
  WHERE brand = 'Ford'
WITH CHECK OPTION;

INSERT INTO audi_cars (car_name,brand) VALUES('Q2','Audi');
INSERT INTO audi_cars (car_name,brand) VALUES('S1','Audi');

INSERT INTO ford_cars (car_name,brand) VALUES('Edge','Ford');
INSERT INTO ford_cars (car_name,brand) VALUES('Mustang','Ford');

INSERT INTO ford_cars(car_name,brand) VALUES('RS6 Avant','Audi');
-- ERROR: new row violates check option for view "ford_cars"
```