**Reeya Kumbhojkar - 900823980**

**CS 486/586 Introduction to DBMS**

**Fall 2021**

Assignment 3 – Subqueries, Views, ER Diagram

Due: Friday, Oct 29, 11:59PM on D2L

Please note the following:

* do this assignment in groups of 2.
* ensure that each group member’s name is listed on the assignment to ensure credit
* submit your assignment in PDF format
* turn in your completed assignment on D2L, one submission per group
* 100 points total

This assignment is partially based on the Spy relational database. Information about this database and information about how to access it is on the Database Info Page. (<http://web.cecs.pdx.edu/~maier/db_resources/db.htm>)

**Part I - Subqueries (60 pts)**

This part of the assignment is based on the Spy relational database. For each statement below, write a SQL query to find the answer to that question. Show the query, first 5 rows of the result relation, and row count. Use subqueries for all answers in this section.

**Hint**: You can use subqueries in the FROM clause and the HAVING clause.

Example subquery in the FROM clause:

**SELECT MAX(cityaverages.avsal)   
 FROM (SELECT AVG(salary) AS avsal**

**FROM agent**

**GROUP BY city) AS cityaverages;**

Example subquery in the HAVING clause:

**SELECT city, avg(salary)   
FROM agent   
GROUP BY city   
HAVING 2\*COUNT(agent\_id) > ALL(SELECT COUNT(agent\_id)**

**FROM agent**

**GROUP BY city);**

1. List the name of the mission with the lowest [mission\_id](http://dbclass.cs.pdx.edu/redirect.php?server=localhost%3A5432%3Aprefer&database=f20tdb74&schema=spy&table=mission&subject=table&sortkey=1&sortdir=asc&strings=collapsed&page=1). (5 points)

* SELECT name FROM mission where mission\_id = (SELECT MIN(mission\_id) FROM mission);

name

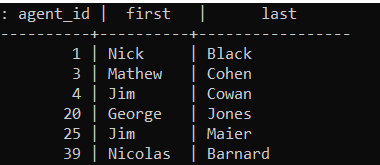
-----------

Third Age

(1 row)

1. List the agents (agent\_id, first, last) who have been on a ‘Presidential’ mission that succeeded. (10 points)

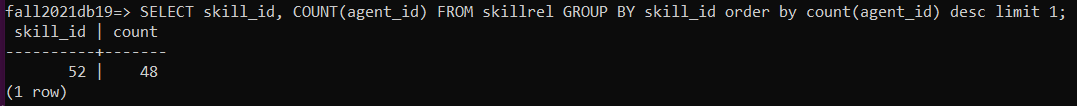
* SELECT agent\_id, first, last FROM agent WHERE agent\_id IN (SELECT tr.agent\_id FROM teamrel tr, mission m WHERE TR.team\_id = m.team\_id AND m.access\_id IN (SELECT sc\_id FROM securityclearance WHERE sc\_level = 'Presidential') AND mission\_status = 'success');



142 rows

1. Find the skill and the number of people, that the most people have that skill(s). (15 points)

* SELECT skill\_id, COUNT(agent\_id) FROM skillrel GROUP BY skill\_id ORDER BY COUNT(agent\_id) DESC LIMIT 1;

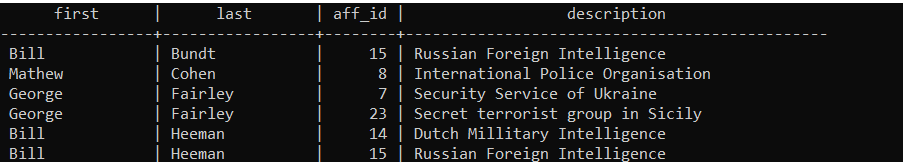


1 row

**Part II - Views (40 pts)**

1. Write an SQL view definition that displays agent names(first, last), the affiliation they belong to and the description of the affiliation(s). Show your view definition and the first 5 rows returned when you SELECT \* from the view, and the row count. (5 points)

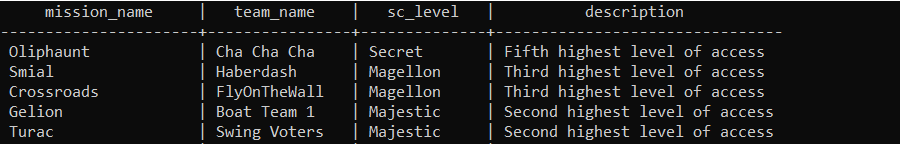
* CREATE VIEW agent\_affiliation AS SELECT ag.first, ag.last, af.aff\_id, af.description FROM agent ag, affiliation af, affiliationrel afr WHERE ag.agent\_id = afr.agent\_id AND afr.aff\_id = af.aff\_id;



954 rows

1. Write an SQL view definition that displays unique mission (name), the team(name) that performed this mission and the text of the missions security clearance level (access id) and description. Show your view definition and the first 5 rows returned when you SELECT \* from the view, and the row count. (5 points)

* CREATE VIEW mission\_sub AS SELECT DISTINCT m.name mission\_name, t.name team\_name, s.sc\_level, s.description FROM mission m, team t, securityclearance s WHERE m.team\_id = t.team\_id AND m.access\_id = s.sc\_id;



404 rows

**Part III - ER Diagrams and ER->Relations (60 pts)**

You may use [**erdplus.com**](https://erdplus.com)(free version) or[**LucidChart**](https://www.lucidchart.com/) for completing these exercises.

Notes on erdplus:

* The erdplus GUI sometimes applies the cardinality symbol on the opposite entity than expected. Enter the data in erdplus so that the resulting diagram is correct.
* ***Regularly save your diagrams locally*** as a backup in case the website loses your account. It happens, I learned the hard way.

In this homework assignment, you're designing the database for an online fan-of-music database. You won't be building the entire online application, or accounting for all possible user interactions.

Background

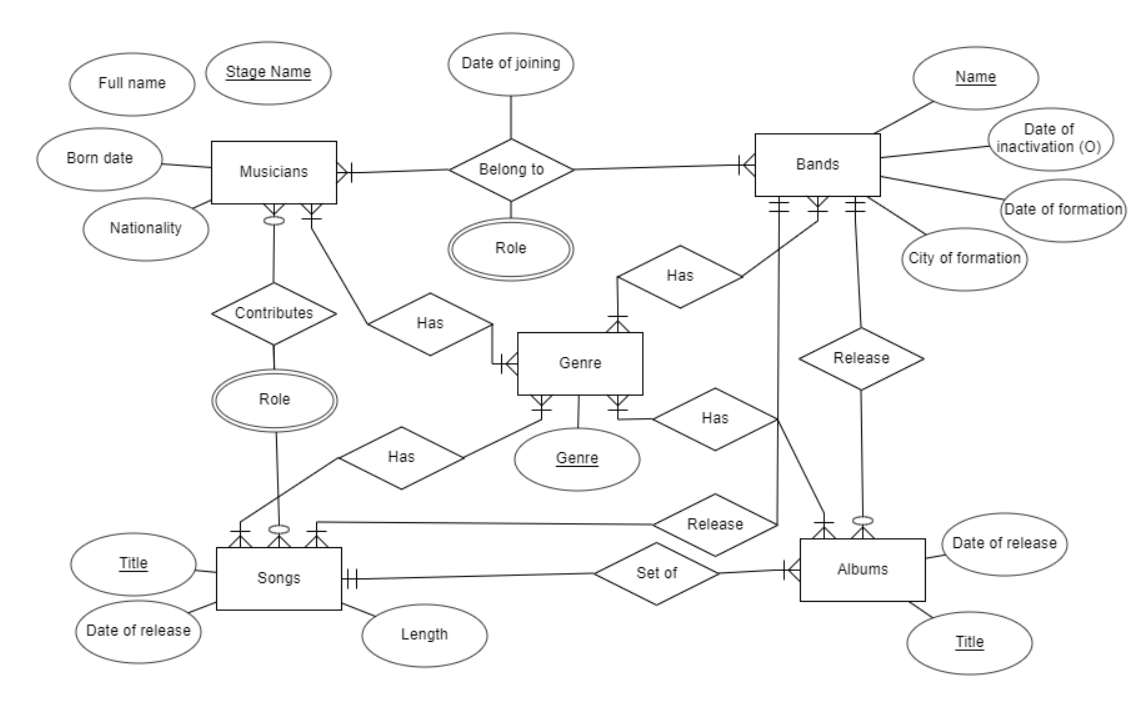
There are many online sources for information about music, but much of the data isn't well structured, and you often have to go to multiple sites to get answers to your questions regarding musicians, bands, songs, albums, sales and concerts. When did Aerosmith form? Look in Wikipedia. When did Roger Waters leave Pink Floyd? Look on a different page on Wikipedia. What songs are on Drake's album "*Take Care*"? Another page in Wikipedia, or Amazon, or iTunes. How many certified album sales does Beyonce have in the US? Yet another page in Wikipedia, or RIAA. When is [Alicia Keys](https://www.ticketoffices.com/Alicia-Keys-(Rescheduled-from-9/2/2020)/1458/3411058) next giving a concert in Oregon or Washington? (September 1st at the Moda…)

You will design a database schema that stores information about musicians, bands, songs, and albums. It won't contain all relevant attributes for each object, but enough to identify uniqueness constraints, foreign key constraints, and answer some of these interesting queries with one database.

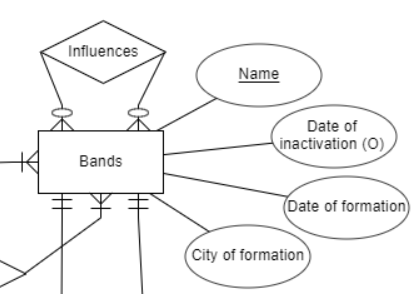
**Consider the following statements about your database:**

* There are several core objects to be modeled in your database, including:
  + Musicians. Each musician has a unique stage name, full name, born date, and nationality
  + Bands. Each musical group is called a band, and is identified by their band name. It holds information being active (unless it's still active).
  + Songs. Each song is identified by a title, the date it was released, and the Band who released it. It also holds information about the length of the song.
  + Albums. Each album is identified by a title and the band who released it. It holds information about the date it was released.
* Note: Solo acts, like Lady Gaga, are modeled with both a Band named "Lady Gaga" and a musician named "Lady Gaga".
* Note: Album is the term used to release a set of music with a title, even if the music collection is released as a CD, or some other format.
* Here are the key relationships to model in your database…
  + Bands are comprised of Musicians. The database records the date when a musician joined a band, and what role they had in the band (such as Lead Vocals, Bass, Drums, etc)
  + Musicians contribute to Songs, whether or not they are in the Band that released the song. The db records what role the musician had in the song.
  + Albums contain Songs.
* The database has a list of Genre's, and tracks which Genre(s) are applicable to a Band, a Song, a Musician, or an Album.

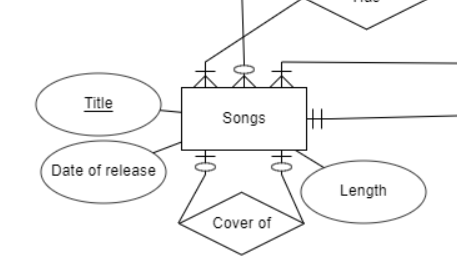
1. Draw an ER diagram that represents these data requirements. Be sure to mark the key attributes and include cardinality constraints on relationships (1, many). Don’t specify the types of attributes. (40 points)



1. Modify your ER diagram to handle the following extensions. Do each part as a separate change from the original diagram. You only need to show the parts of the diagram that change. (10 points)
   1. A band may have one or more influences, which are other bands.



* 1. A song may be a cover of another (one) song.



1. Translate Musicians, Bands, Songs and Albums (any any relations that tie those relations together) from your ERD into a db schema using the format:

*TableName1(Attribute1, Attribute2, Attribute3,…)*

*Attribute2 is a foreign key referencing Table3.*

Underline the attributes making up the primary key of each table. Describe foreign keys in a separate line. (10 points)

* Musicians(stage\_name, full\_name, born\_date, nationality, genre)

genre is foreign key referencing Genre\_table

* Bands(name, date\_of\_formation, city\_of\_formation, date\_of\_inactivation, genre)

genre is foreign key referencing Genre\_table

* Songs(title, date\_of\_release, length, genre)

genre is foreign key referencing Genre\_table

* Albums(title, date\_of\_release, genre)

genre is foreign key referencing Genre\_table

* Belong\_to(musician\_name, band\_name, date\_of\_joining, role)

musician\_name is foreign key referencing Musicians

band\_name is foreign key referencing Bands

* Contributes(name, title, role)

name is foreign key referencing Musicians

title is foreign key referencing Songs

* Set(album\_title, song\_title)

album\_title is foreign key referencing Albums

song\_title is foreign key referencing Songs

* Release\_info(name, album\_title, song\_title)

name is foreign key referencing Bands

album\_title is foreign key referencing Albums

song\_title is foreign key referencing Songs

* Genre\_table(genre)