IT5100C: Database Modelling and Programming

Project 01: SQL Query AY2023/24 Sem 2: Week 08 (Saturday, 16 March 2024)

Instructions

1. Please read the **Instructions** and **Preliminary** before continuing.
2. This is a **Group Project**.
3. This project consists of **FIFTEEN (15)** SQL questions. The total is 60 marks.
   - Answer the questions based on the group size as specified in **Task**.
4. **Late submission penalty:**
   - Ten marks will be deducted for each late day *up to six late days*.
   - Submissions *after* the sixth late day will receive **zero** marks and will not be graded.
5. The project is to be submitted on Canvas.
   - Only one person per group need to submit.
   - Copy the query for each question to the individual box on Canvas.
6. The project will be **auto-graded**.
   - Additional *hidden* test cases may be added to ensure no hard-coding.
   - The SQL query will be run on PostgreSQL 16.
   - You **must** follow the requirement given in **Preliminary**.
   - Partial marks may be given on a categorical basis (*e.g.*, correctly answer given test data, correctly answer private test data, *etc*).
7. The deadline for the submission is on Week 08 (Saturday, 16 March 2024) at 12:00
   - You can submit multiple times, only the latest submission will be graded.
   - Submit only `Answer.py`.
   - Only one person per group need to submit.
   - Make sure that your latest submission is before the deadline or the penalty applies.
   - Make sure that your latest submission is complete (*i.e.*, does not miss any question) as we will not look in previous submissions for missing answers.

Good Luck!

Preliminary

In this project, you will formulate **15** SQL queries and submit your solutions into canvas project SQL Queries (https://canvas.nus.edu.sg/courses/54297/assignments/107143). As the project will be auto-graded, it is very important that your submitted answers conform to the following requirements:

- ~~Submit each question into its own individual answer box on Canvas.~~ Submit your `Answer.py` to Canvas. Make sure that you submit only **ONE** SQL statement for each question (*i.e.*, each variable `ansXX` only has a single statement). This statement should be an SQL query and ends with a semi-colon (*i.e.*, `;`).
- Each question has a specification on the output schema. You **MUST** follow the expected column name, otherwise there will be penalty even if you have the correct result. Consider the following template below using `AS` keyword for column renaming.

```
SELECT attr1 AS name, attr2 AS year
  query
;
```

  If the code above can be run correctly, then it satisfies the following schema:

name	year

- Each answer must be a syntactically valid SQL query without any typographical errors: an SQL answer that is syntactically invalid or contains very minor typographical error may receive **ZERO (0)** marks even if the answer is semantically correct.
- For each question, your answer must not contain any duplicate records. You may use *`DISTINCT`* as you see fit.
- Each question must be answered independently of other questions.
- You are **NOT** allowed to do any of the following for this project
  - `CREATE TABLE`
  - Using transaction
- Unless a value is explicitly specified in the question (*e.g.*, `'James Dean'`), you are not allowed to hardcode any value.
- You are **NOT** allowed to import any libraries and your answers must be executable on PostgreSQL.

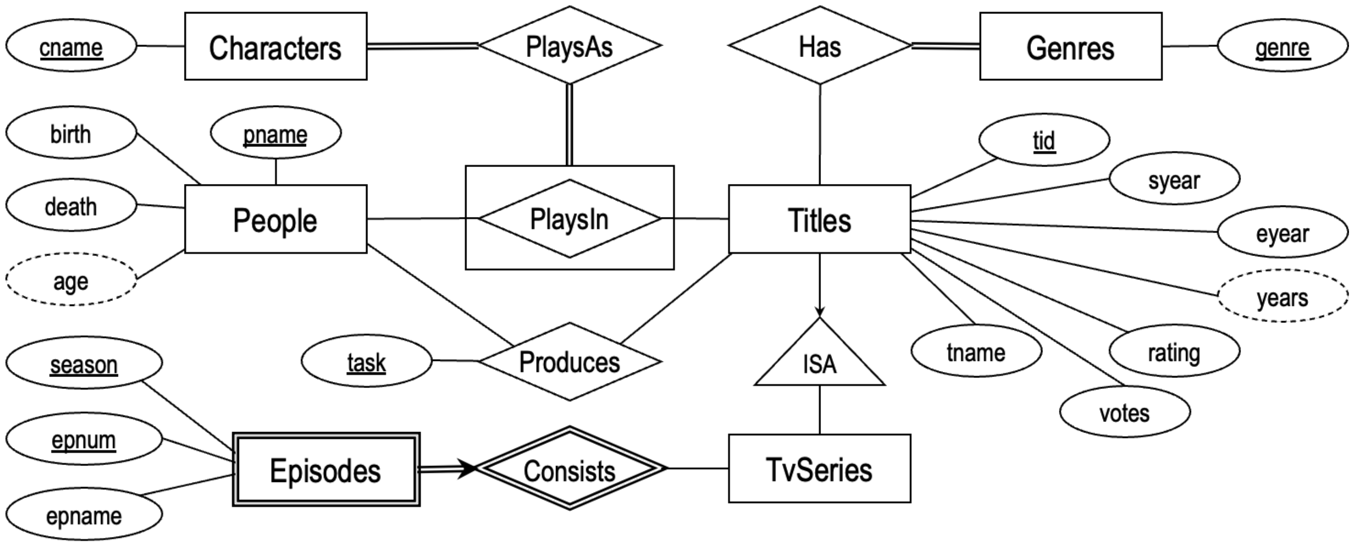
Database

The database is available on Canvas. The files are available in "Canvas > Files > Project" (https://canvas.nus.edu.sg/courses/54297/files/folder/Project).

- Clean.sql: Contains the series of `DROP TABLE` statement to clean your database.
- DDL.sql: Contains the series of `CREATE TABLE` statement to (re-)initialize your database.
- Data.sql: Contains the series of `INSERT INTO` statement to (re-)initialize your tables.

For completeness, the approximate ER diagram for the schema is given below. You may assume that the data will be consistent with both the schema and the ER diagram. In particular, assume that total participation constraints are always enforced.

Do note that unless specified in the schema with `NOT NULL`, the attributes may contain `NULL` value. You should try to make your code work with `NULL` value whenever possible.



Additional Constraints

- We refer to entries in `Titles` as "film title" even when the title may also be a TV series (*i.e.*, an entry in `TvSeries`).
- If a film title is in `TvSeries`, we refer to them as "TV title", "Series title", "TV series title", or simply "TV series".
- If a film title is in `Titles` but **NOT** in `TvSeries`, we refer to them as "movie title" or simply "movies".
- We refer to "actor" as entries in `People` who are also entries in `PlaysIn`.
- The derived attribute `age` in `People` is derived purely from `death - birth` regardless of the date.
- If the person (*i.e.*, entries in `People`) is still alive, the value of `death` is `NULL`. Otherwise, the value of `death` is recorded and it must be greater than `birth` (*i.e.*, no person with age less than 1 year is recorded).
- The derived attribute `years` in `Titles` is derived purely from `eyear - syear` regardless of the date.
- If the film title (*i.e.*, entries in `Title`) is still ongoing, the value of `eyear` is `NULL`. Otherwise, the value of `eyear` is recorded.
- The `runtime` in `Titles` is specified in minutes.
- The `rating` in `Titles` must be between 0 and 10 (*inclusive of both*).
- For simplicity, the type of `birth`, `death`, `syear` (*i.e.*, start year), `eyear` (*i.e.*, end year) are *`INT`*.
- The people who produces film title (*i.e.*, entries in `Produces`) may be `'director'`, `'composer'`, or other task (*e.g.*, `'producer'`, *etc*).

Task

Depending on the size of your group, we will try to simplify the task a little. However, everyone is expected to do the **Hard** questions. Consult the following table based on your group size:

Size	**Easy**	**Medium**	**Hard**
1	1, 2, 3	9, 10, 11	12, 13, 14, 15
2	1, 2, 3, 4	8, 9, 10, 11	12, 13, 14, 15
3	1, 2, 3, 4, 5, 6	7, 8, 9, 10, 11	12, 13, 14, 15

The marks will be redistributed in the case of fewer questions.

Questions

Easy

1. We will start with a simple question first.

   Find all the names as well as the age of the people who are still alive. We define the age as of 31 December 2023 at 23:59. So if the person is born in the year 2000, the age is 23 regardless of the actual date and time of birth.

   The output schema should be

name	**age**

   You should see "Brigitte Bardot" with the age of 89.

2. This one requires some additional thinking.

   Find all the information about the people with at least 3 parts to their name. In other words, their name is at least of the form "first_name middle_name last_name" and potentially more (*e.g.*, two middle names). Note that there should be only a single space between first_name, middle_name, and last_name.

   For this question, you may assume that there will always be only a single space in between the name.

   The output schema should be

name

   There should be only `'Olivia de Havilland'` in the current database instance.

3. Something similar again.

   Find all episode names (`epname`) with `'Singapore'` as a word in its name. A word must be separated by at least a single space.

   So `'Singapore Story'`, `'Souvenir from Singapore'`, and `'This is Singapore Story'` are accepted. However, `'Story of Singaporean'` is not accepted because `'Singapore'` appeared as part of another word.

   The check should be case-sensitive.

   The output schema should be

episode

   There should be only `'Souvenir from Singapore'` in the current database instance.

4. What about the opposite?

   Find all character names and the actor who plays them such that the character only has exactly one word in their name. Recap as above that a word must be separated by at least a single space.

   So `'Becket'` is accepted but `'Thomas Becket'` is not.

   The check should be case-insensitive.

   The output schema should be

character	**name**

5. Now we will have a slightly more complex conditions.

   For each movie title, find the movie name and the start year such that the start year from 1960 or above.

   Recap that movie are films that are not TV series.

   The output schema should be

title	**syear**

   There should at least be three movies from the year 1960 namely “La Dolce Vita”, “Spartacus”, and “The Truth”. In total, there should be 26 rows in the output.

6. Let us find some good movies that survived “review bombing”.

   Find all the movie titles, runtime, and starting year with rating of at least 7.5 and at least 10000 number of votes.

   The output schema should be

title	**runtime**	**syear**

   There should be 48 good movies to watch while you complete the rest of the project.

Medium

7. Let us now start joining tables.

   For each director name, find the film title that has been produced by the director with the genre of `'Drama'`. Recap that a director is the person that has a task of `'director'`.

   Note that the genre `'Drama'` is the genre of the film. This is not about a director that has produced at least one `'Drama'` genre.

   The output schema should be

director	**title**

   There should be 7 rows in the output.

   WARNING: You may easily forget the `'Drama'` requirement and you will see no changes in the result. It seems drama is a popular genre.

8. Please be careful with the following question as it involves dealing with `NULL` values.

   For the people who is not known to have played in a film title, find their name, birth year, and death year (if any, otherwise keep it as `NULL`).

   Note that these people may actually be director.

   The output schema should be

name	**birth**	**death**

   There should be 3 rows in the output.

9. And now we arrive at queries that may require aggregation (*but not all*).

   For each person who plays in a film title, find their name, the total number of distinct films they have played in, the earliest starting year of the film they have played in, and the longest runtime for the film they have played in.

   Exclude people who have not played in any films at all. This also excludes people who produced the films unless the person has also been an actor. Currently, there is no such person in our database instance.

   Note that by excluding, the row should not even be there. If you use the number 0 to indicate the count, it is considered wrong.

The output schema should be

name	count	earliest	longest

There should be 17 rows in the output.

10. Recap that aggregation can also be filtered. Also remember that you should not hardcode any values unless the value is specified in the question.

    Find the name of the person as well as the number of films the person has played in where the person has played more than or equal number of distinct films from 1950 onwards when compared to `'Marlene Dietrich'` in her entire known career (*i.e.*, only based on the values in the database).

    In other words, if a person P has played in n films from 1950 onwards and `'Marlene Dietrich'` has played in m films in her entire career including those from before 1950, we include P in the output if $n \geq m$.

    Note that your code should still work even if the database instance is changed. Additionally, we only care about the appearance in the film title and not episodes (if the film is a TV series). So even if the film has 1000 episodes, the person has only played in 1 film and not 1000.

    The output schema should be

name	count

    There should be 5 rows in the output. Note that `'Marlene Dietrich'` has played a total of 3 films known in the database but your code should work for other known numbers too.

11. We define talented actors as actors that have played in a film as two different characters.

    Find all the actor names and character names such that the actor has played as at least two different characters in the same film.

    Note that two characters are different if they have different character names.

    The output schema should be

name	character

    For instance, Kevin Spacey is both Roger Kint and (*spoiler alert*) Keyser Soze in the film called The Usual Suspect. In the Batman Trilogy by Christopher Nolan, we also have Aaron Eckhart playing both Harvey Dent and the character named Two Face.

    In our data set, we have one actor that fits this. Richard Burton has played as both "Becket" and "Thomas Becket". Although this is *likely* due to incorrect data input, we will still acknowledge the talent of Mr. Richard Burton.

Hard

12. For this question, the **order** of the rows matter.

    For each TV series that has ended, find all the known TV series names, season number, episode number, and episode name. Recap that TV series are films that are also TV series.

    Arrange the result in ascending order of TV series title, followed by descending order of season number, and lastly in ascending order of episode number.

    Note that there may be episodes without name (*i.e.*, `epname` is `NULL`). These episodes should still be included.

    The output schema should be

title	**season**	**episode**	**name**

    There should be 359 rows in the output. All from Schlitz Playhouse. Note that 330 of them have non-`NULL` value on `epname`. So if your get 330 rows in your result, you may have extra constraints.

13. Here is another question where the **order** matters.

    For each genre, find the total number of film title with that genre. Display only genre with at least 5 film titles. Furthermore, display only the bottom 3 genres in terms of the number of film titles and display them in descending order of total number of film title. For genre with the same total number of film title, sort them in ascending order of genre.

    Note that the bottom 3 genre is

    - the bottom among the genre with at least 5 film titles, and
    - we must have 3 unique count of film titles regardless of the number of genres.

    The output schema should be

genre	**count**

    There should be 7 rows in the output.

    For genres with at least 5 film titles, the bottom 3 counts are 6, 7, and 9 film titles. There is only one genre with 6 and 9 film titles but there are five genres with 7 film titles. This gives the total number of rows as 7.

    To help you, the results should be the following:

genre	count
Film-Noir	9
Adventure	7
Biography	7
Mystery	7
Thriller	7
War	7
Action	6

14. James Dean is one of the most famous icon of western drama films. He is very focused on both drama and western genre. We want to find out if there are other actors with more varied genre than James Dean.

    Find the name of the person as well as the film genre of people who have played in a film genre that has been played by `'James Dean'`. Since James Dean has only played in drama and western (*in our current database*), we want to find the people who have played in both drama and western movie but possibly more genre.

    Please be reminded that your code should work in all instances even when James Dean has played in more genre or we have no data on James Dean at all.

    Also note that you should exclude James Dean himself although obviously he has played in all the genre he has played.

    The output schema should be

name	genre

    There should be 23 rows in the output.

    James Dean has played in 2 genres: Western and Drama. So anyone who has played at least these two genres should be included. We should include the following people:

    - **Doris Day** has played in 8 genres
      - Biography, Comedy, Drama, Music, Musical, Romance, Thriller, and Western
    - **Gary Cooper** has played in 8 genres
      - Adventure, Biography, Comedy, Drama, History, Romance, Thriller, and Western
    - **Lauren Bacall** has played in 7 genres
      - Comedy, Crime, Drama, Film-Noir, Mystery, Romance, and Western

    This gives a total of 23 rows. If you see 25 rows, you likely still include James Dean in your output.

NOTE:

What happen when there is no James Dean in the data or James Dean has not played in any film? The idea is this, let's fix the set of people in our data as P. Consider the following scenario:

- Let James Dean played in 3 genre and indicate the subset of P as P_3.
- Let James Dean played in 2 genre and indicate the subset of P as P_2.
- Let James Dean played in 1 genre and indicate the subset of P as P_1.

The natural question is which of the following property holds?

- $P_3 \subseteq P_2 \subseteq P_1$
- $P_1 \subseteq P_2 \subseteq P_3$

Extending this to P_0 for the case where James Dean played in 0 genre, the answer above can be extended naturally to

- $P_3 \subseteq P_2 \subseteq P_1 \subseteq P_0$
- $P_0 \subseteq P_1 \subseteq P_2 \subseteq P_3$

The behavior is then the one that follows the natural progression as laid out above.

15. Now let us talk about "popularity".

For each year, find the most popular genre after 1960. The most popular genre is the genre with the most number of films for the given year. Note that the most popular genre may not be unique. In which case, you output all genres that are the most popular for that particular year.

Include the year without any films. You may still receive partial marks if you "**exclude** the year without any films".

HINT: Can you first find all the year from 1960 onwards?

The output schema should be

year	**genre**

There should be 53 rows in the output.

Some years have multiple most popular genre. For instance, in 1961, there are 2 popular genres: Drama and War. On the other hand, some years have no record at all. For instance, there are no films between 1966 and 1971. So the genre should be `NULL`.

DB Checker

We have provided a Python script to automatically check your SQL query against the public test data. Please ensure that your code works for other data as well.

Initialization

You will need the following:

- Python (*preferrably v3.11 and above*).
- Psycopg library for Python.

Psycopg

To install Psycopg, run the following command on terminal:

```
$ pip install psycopg
```

or alternatively

```
$ python -m pip install psycopg
```

depending on how you installed Python. If those does not work, try either `pip3` or `python3` first before trying the fix below.

If the fix above does not work, then you may need to update `PG_HOME` or `PATH`. The following fix was needed on MacOS and was not needed on Windows.

```
$ export PG_HOME=/Library/PostgreSQL/16
$ export PATH=$PATH:$PG_HOME/bin
```

Try to perform the `pip` install again.

If there is still error, this is likely due to the missing binary. Run the following additional commands:

```
$ pip install "psycopg[binary,pool]"
```

or

```
$ python -m pip install "psycopg[binary,pool]"
```

Writing Your Answer

Your answer should be in the file called `Answer.py`. We have provided 15 variables `ans01` to `ans15`. You should write your answer in between the triple quotes.

For example, the following is the template for `ans01`.

```
ans01 = '''

'''
```

If the answer is `SELECT * FROM People WHERE birth > 1930;`, write your answer as follows (*indentation and semi-colon optional*).

```
ans01 = '''
SELECT *
FROM   People
WHERE  birth > 1930;
'''
```

Checking Your Answer

You can check your answer using the file `Checker.py`. You should first open `Config.py` to add your PostgreSQL password to the field `'pswd'`. (*i.e.*, replace the `'********'` with your password). Do **NOT** reveal your password to others.

There are several ways to check your answer. The first is to answer all questions and check the overall result. This can be done by calling the following function:

```
test_all()
```

This will print a table with the following header:

qns	header	body

For the column **header** and **body**, you will see a ✓ if the answer match and a ✗ if the answer does not match.

The second is to check individual question. This can be done by

```
test(qn) # qn is the question number (e.g., test(1) for question 1)
```

The output is more straightforward, you will see two value: `(head, body)`. `head` corresponds to the correctness check on the header and `body` corresponds to the correctness check on the body. So if you see `(True, True)`, it means both are correct. However, if you see `(True, False)`, it means the header is as expected while the body has incorrect results.

Finally, if you just want to see what your code produce, you can type

```
display(qn)
```

This can also be paired with

```
expected(qn)
```

to see the expected output.

More Information

By default, the printing is kept to a minimum so as to not clutter your screen. If you need more information, you can change the setting at the beginning of `Checker.py` to print more information. This is controlled by the variable: `LOG_LEVEL`.

There are 4 levels of details. The smaller the value of `LOG_LEVEL`, the more information are printed.

Level 0: Print all information.

Level 1: Print trace information.

Level 2: Print important information.

Level 3: No print (*i.e.*, only final result).

We recommend starting at level 3 and only go down when you are not passing the test cases.

Prepared Statement

The code illustrate how SQL can be combined with other languages. You may use the code as a template for your future project. However, the current practice of having SQL query in a variable may not be the best option as it does not take into account user inputs.

While you may be tempted to simply construct query string on the fly based on user input, that is not a good practice. For instance, the following code:

```
# we use f-string, assume there is variable called 'birth'
ans01 = f'''
SELECT *
FROM   People
WHERE  birth > {birth};
'''
```

may work but it relies on the construction of the string which may be abused by others.

Without going into much details, the proper way to execute this using our library is the following:

```python
ans01 = '''
SELECT *
FROM   People
WHERE  birth > %s;
'''
# Here %s is a placeholder for values to be inserted later
db = DB(**config)      # initiate connection
db.fetch(ans01, 1930) # 1930 will be placed in placeholder
db.close()             # close connection
print(db.res[-1])      # print last result
```

If you have multiple arguments, you need multiple `'%s'`. You can then simply add more arguments when calling `db.fetch(sql, arg1, arg2, ..., argn)`.