Problem 1

We have a set of python data-frames:

- One data frame related to events info, with columns like (event_url, event_name, event_start_date, event_city, event_country, event_industry);
- One data frame related to event attendees with columns like (event_url, company_url, company_relation_to_event) columns,
- One data frame with company info with columns like (company_url, company_name, company_industry, company_revenue, company_country);
- One dataframe with company contact info with columns like (company_url, office_city, office_country, office_address, office_email);
- One dataframe with company employee info with columns like (company_url, person_id, person_first_name, person_last_name, person_email, person_city, person_country, person_seniority, person_department).

We want to write a filtering function/class in python that allows us to filter all the data-frames based on any conditions on any of the columns in these data frames, taking into account the interconnectedness of the data-frames. The purpose of this function/class is to enable us to find relevant events by selecting some company or people attributes (e.g. find events being attended by directors of oil & gas companies), or find people by events or companies (e.g. find emails of people working for companies attending tech events), etc.

For example, if we filter over event_name column, we should only get companies linked to that event, and we should only get people linked to those companies (that are linked to the event). If we select a few values for person_seniority, we should only get companies for which we have people with that seniority, and only get events attended by those companies. The same logic should hold if we filter over multiple columns, e.g. if we select a few values for event_city and a few values for person_seniority, we should only see events, companies, and people linked to both those selections.

The implemented function should be such that it works even if the number of connected data-frames increases (e.g. if we add another people related data frame containing their past employments).

We can assume that any event related data frame will have an event_url column to uniquely identify the event, any company related data-frame will have a company_url column to uniquely identify a company, and any person related data frame will have a person_id column to uniquely identify a person. Any data frame that contains info about multiple entities (e.g. both events and companies) will have ids for both entities.

Discuss two different ways to implement this filtering functionality, and implement one of them that you prefer.

Problem 2

For the same type of data frames as in Problem 1, let's say we have the data stored in the following PostgreSQL tables

- event_attributes: with (event_url, attribute, value) columns, where attribute is any event attribute, such as event_start_date, event_city, event_country, event_industry, etc;
- company_attributes: with (company_url, attribute, value) columns, where attribute is any company attribute, such as company_name, company_country, company_industry, company_revenue, etc.
- people_attributes: with (person_id, attribute, value) columns, where attribute is any person specific attribute, such as (person_first_name, person_last_name, person_email, person_seniority, person_department)

We want to write a function to query the relevant data. The inputs to the function will be a python dataframe specifying the conditions in (column_name, condition, value) format, and a list of column names specifying the attributes we need returned from the data.

For example, to retrieve tech companies attending events in Singapore, the input could look like this

```
filter_arguments = [
        ['event_city', 'includes', ['Singapore']],
        ['company_industry', 'includes', ['Software', 'IT']],
]
output_columns = ['event_city', 'event_name', 'event_country', 'company_industry', 'company_name', 'company_url']
```

To retrieve email addresses of directors of tech companies attending events in Singapore in Jan 2025, the inputs could look like this

```
filter_arguments = [
        ['event_city', 'includes', ['Singapore']],
        ['event_start_date', 'less-than-equal-to','2025-01-31'],
        ['event_start_date', 'greater-than-equal-to','2025-01-01'],
        ['company_industry', 'includes', ['Software', 'IT']],
        ['person_seniority', 'includes', ['director']],
]
output_columns = ['event_city', 'event_name', 'event_country', 'company_industry', 'company_name', 'company_url', 'person_first_name', 'person_last_name', 'person_seniority']
```

The function should be general enough to allow us to pass conditions on any of the attributes present in any of the databases. And the function should support at least these type of conditions: ['includes', 'greater-than-equal-to', 'less-than-equal-to'].

Problem 3
Consider the datasets you used for the first test, including events, companies, and people datasets. Let's assume we have implemented a system that uses an LLM to convert natural language queries to convert it into an SQL query. Now suggest an approach to automatically verify and improve the LLM-generated SQL query, to ensure that not only the query is executable, but that it is as close to the perfect query as possible. That is, we are trying to implement an automated way of identifying any issues/shortcomings in LLM-generated query, and rectify them, before generating the final answer, and sending it to the user.

Please keep your description to no more than one paragraph, and mention only the important conceptual aspects of your approach (and no need to discuss technical details of how you would implement them)