CyberMACs Conference DBMS

Chris Glettenberg,Mariam Ahmed, Joseph Joy, Aykhan Guliyev

10 July 2025

**Table of Contents**

**I. Database Initial Study and Analysis**

i. Summary of the data gathering method(s) used by the team

ii. Summary of the information requirements of the system

**II. Database Design: Conceptual Design**

i. Initial Entity Relationship (and/or EER) Model

ii. Business Rules

iii. Data Dictionary

**III. Database Design: Logical Design**

i. Relational Schema

**IV.Physical Design/Implementation**

i. Designing in Postgres

ii. Mockaroo

**V. Querying/Reporting**

i. Overall Explanation of Situation

ii. Screenshots and Explanations of Queries

1. **Summary**

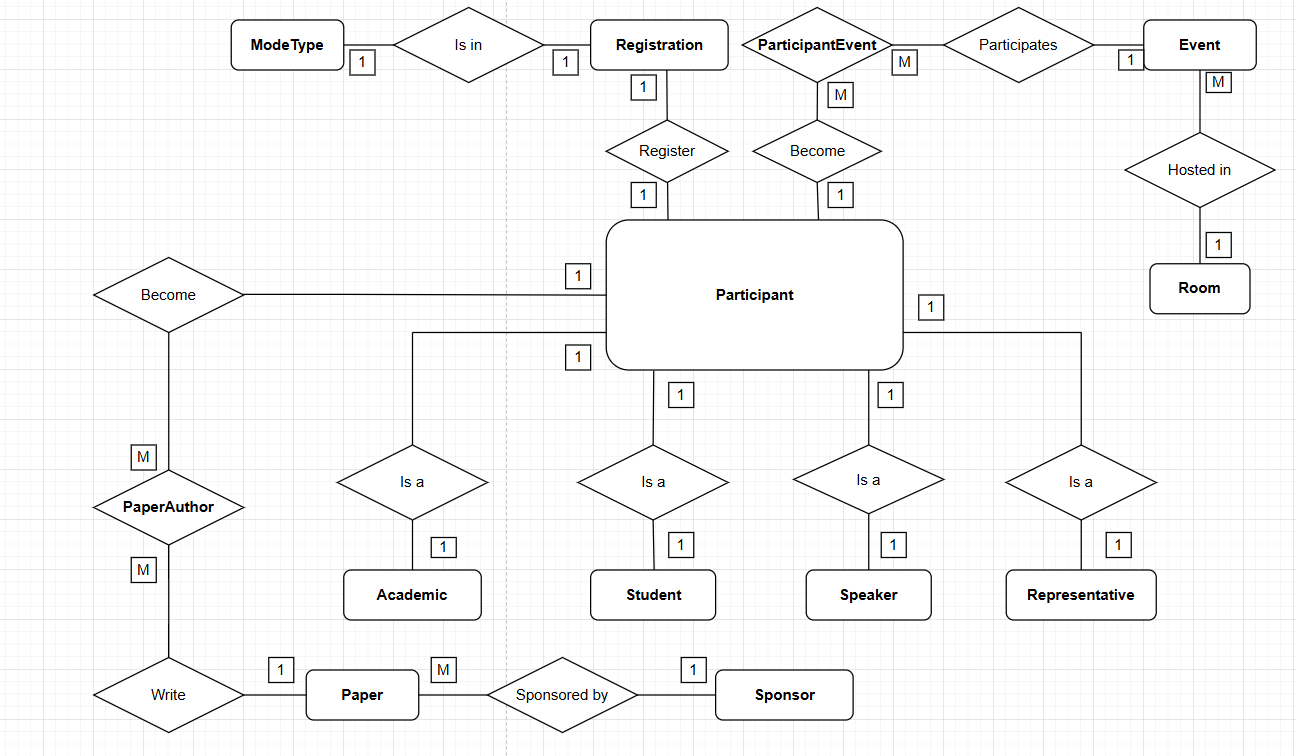
The scenario for this project was the CyberMacs conference, a tech conference where members of certain universities can register to present papers or topics in the Tech field. We built upon this idea to create a database of Participants, Speakers, Sponsorships, Students, Representatives, Papers,Academics, Events, Mode, Registrations and Rooms. We then further expanded this idea to incorporate charts that also included the different Events each participant was attending, as well as the mode (in person or offline) of the registration. We decided to focus more on the view of the system more from the Sponsors and Event managers perspective, so many of our data bases and entities are built around information that would be useful to these specific entities. Our idea with our database and queries, is that a sponsor or event manager would be able to see information on all participants and events, and make informed decisions using our database. Some examples include, seeing the average GPA of students attending an event, biggest sponsors, and most popular sessions(events). We will go further into the exact relationships and specifications of this data later.

Given the very specific scenario for this project, we chose to generate our own, random data. First, we made a detailed outline of our database using PostgreSQL defining exactly what data sets that we needed. After we completed this step, we filled the data using the online tool Mockaroo. Mockaroo, allows you to generate random data, up to 1000 rows of random values that you select. It was a great tool for this project because it has already loaded over 200 options for common fields in a data set, such as Job Title, University, and many more. This allowed us to easily generate random fields for almost all of our data, using the row number as a unique Primary Key for all of our data. Additionally, you can use a random number generator in Mockaroo, to simulate random Foreign Key relationships, setting the range for the random number as the range for the primary key you wish to reference. After entering in all of your values, you can export to an SQL file that can be directly opened in the Postgres PgAdmin hub that contains our database. Then, we ran the series of insert commands to fill our data set. Now we have a data set full of values, ready to perform some queries. First let us dive deeper into the entities we used and their relationships.

1. **Database Design: Conceptual Design**

*EER Model*

This was generated using the [Draw.io](http://draw.io). It helps to identify the relationships between the entities. Here Chen's model was used to draw this relationship. This showcases all of our individual entities, as well as the relationship between them.



<https://drive.google.com/file/d/1tIimmhjY8zkmc5IdafBORJzIsxJhF91r/view?usp=drive_link>

*Business Rules*

The business rules help showcase the type of relationship bonds between each entity in plain text. Here the business rules are built using the reference of the EER model shown above. These business rules strictly outline the relationships between the data as well as the type of relationship (One to Many, Many to Many, etc.)

1. A PARTICIPANT can do one REGISTRATION:

Each REGISTRATION is associated with one PARTICIPANT:

*[One-to-One: Participant → Registration]*

2. A REGISTRATION is done in one MODETYPE.

A MODETYPE can be used for one REGISTRATION:

*[One-to-One: ModeType → Registration]*

3. A PARTICIPANT can participate in one or many EVENTs.

An EVENT can include one or many PARTICIPANTs.

*[Many-to-Many via ParticipantEvent]*

4. Each EVENT is hosted in one ROOM.

A ROOM can host one or many EVENTs.

*[One-to-Many: Room → Event]*

5. A PARTICIPANT can write many PAPERs.

Each PAPER can be written by many PARTIPICANTs.

*[Many-to-Many via PaperAuthor]*

6. A PAPER is sponsored by one SPONSOR:

A SPONSOR can sponsor one or many PAPERS.

*[One-to-Many: Sponsor → Paper]*

7. PARTICIPANT can be an ACADEMIC.

An ACADEMIC is-a PARTICIPANT.

*[SUBCLASS- One-to-One: Academic → Participant]*

8. PARTICIPANT can be a STUDENT.

A STUDENT is-a PARTICIPANT.

*[SUBCLASS-One-to-One: Student → Participant]*

9. PARTICIPANT can be a SPEAKER.

A SPEAKER is-a PARTICIPANT.

*[SUBCLASS-One-to-One: Speaker → Participant]*

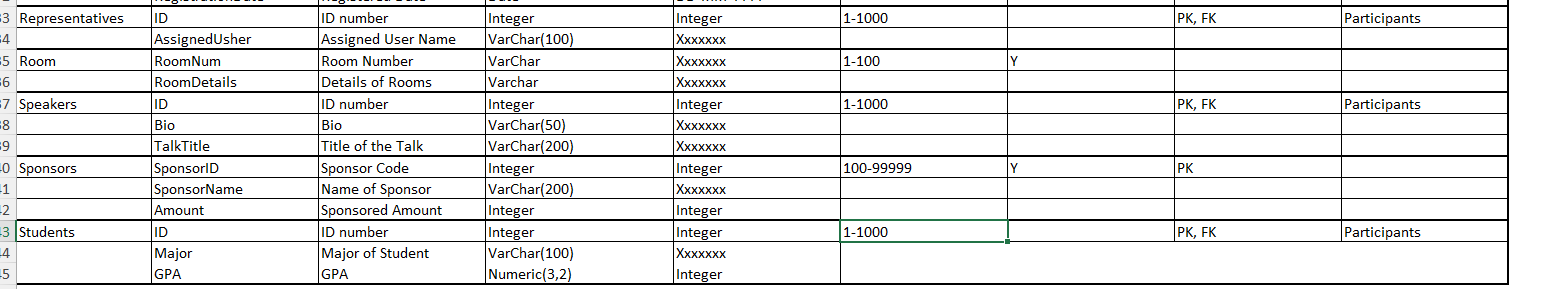
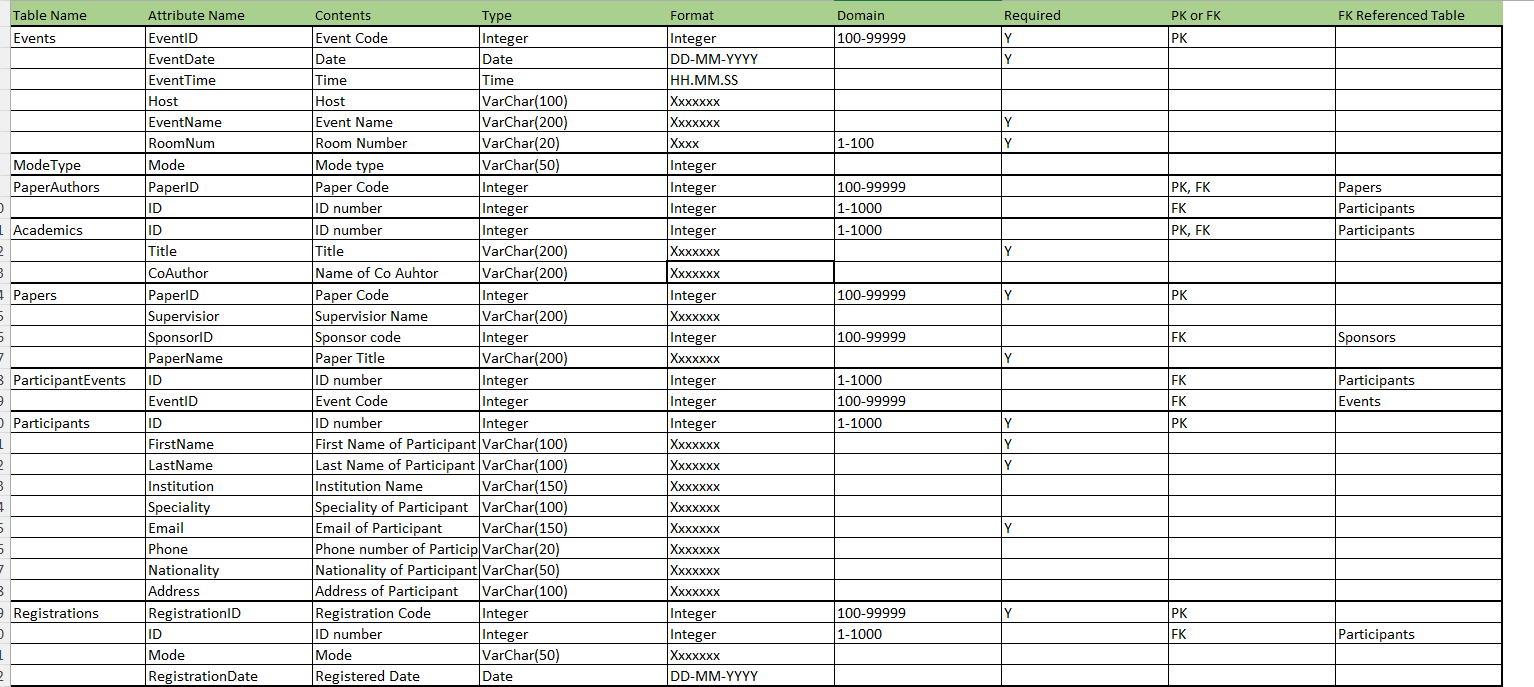
10. PARTICIPANT can be a REPRESENTATIVE.

An REPRESENTATIVE is-a PARTICIPANT.

*[SUBCLASS-One-to-One: Representative → Participant]*

*Data Dictionary*

Here we made a data dictionary which explains the different tables and its attributes, what their type, contents, formats, etc. This can give all details of the data. Data dictionary was developed in Microsoft Excel. The contents column in the data dictionary helps to know about the attribute name.



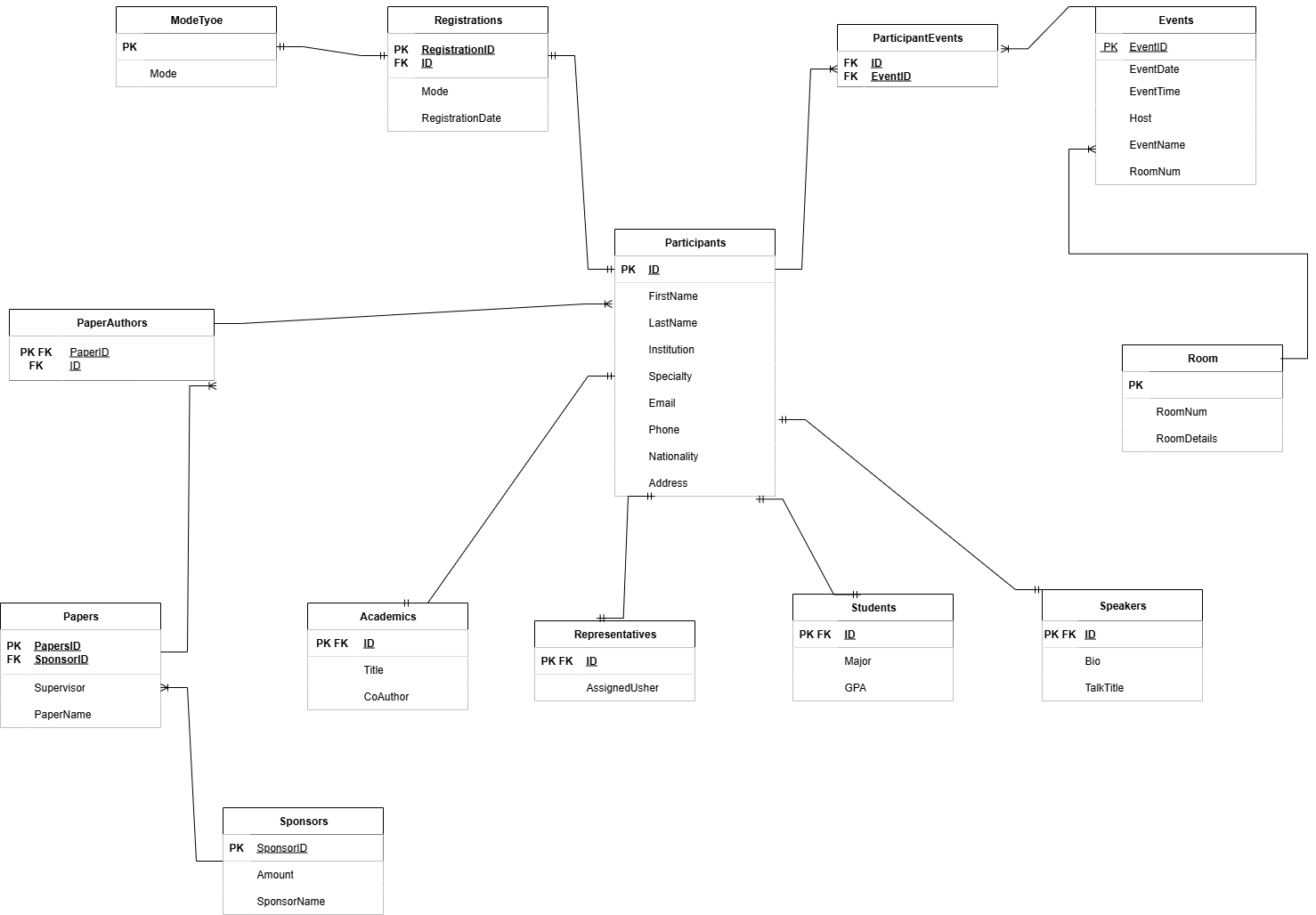
1. **III. Database Design: Logical Design**

The logical design of our database system translates the conceptual ER model into a relational schema that is ready for implementation. This relational schema was created using Draw.io and is based on the business rules and relationships previously defined in our EER model. The schema accurately represents the structure of our database by defining the tables, primary keys (PK), and foreign keys (FK) for each entity and the connections between them.

Our relational schema contains key entities such as **Participants**, **Events**, **Papers**, **Sponsors**, and **Registrations**, among others. Each entity has been normalized and properly structured to maintain referential integrity and minimize data redundancy. The relationships between the entities include one-to-one (e.g., Participant–Speaker), one-to-many (e.g., Sponsor–Papers), and many-to-many connections (e.g., Participants–Events via the ParticipantEvents associative entity), all of which are accurately captured using appropriate junction tables.

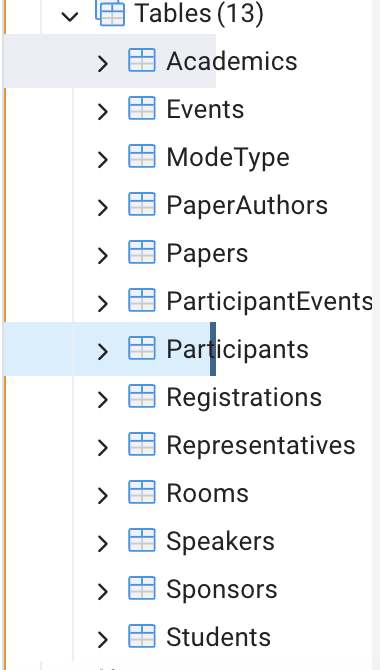
To complement the relational schema, we developed a Data Dictionary, which provides detailed information about each table and its attributes, including data types, contents, and purpose. This helped ensure consistency and clarity during the physical implementation phase.

Overall, this relational schema serves as the backbone for our database and provides a clear, normalized structure that supports efficient querying and accurate data analysis for the CyberMACs Conference DBMS.



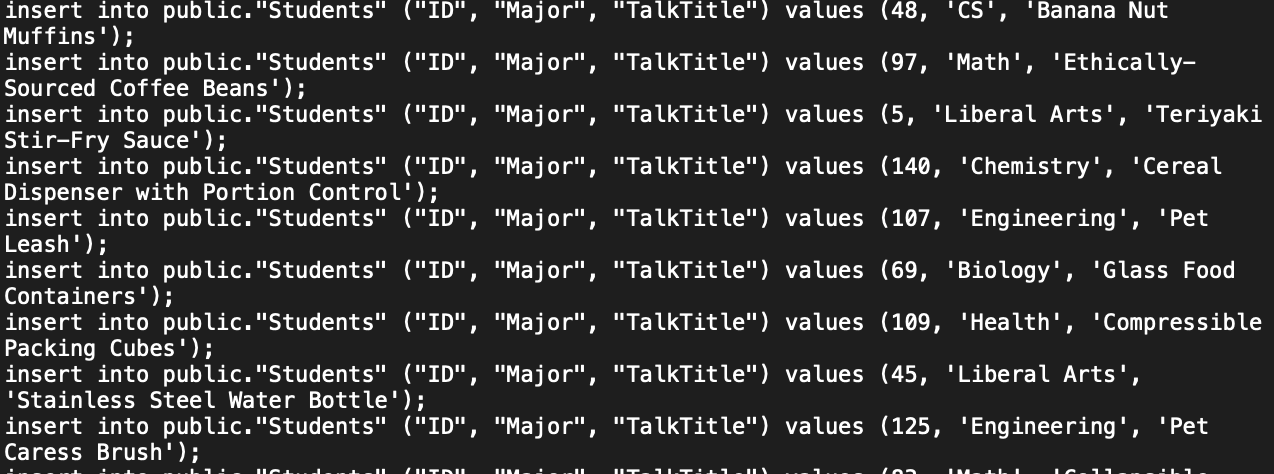
Link: https://drive.google.com/file/d/1k82cv\_2n2YbyoxV40UQhM2gtQP4lG\_t6/view?usp=sh

**IV. Physical Design and Implementation**

As we discussed in the first section, we decided to use a combination of Postgres SQL and mockaroo in order to bring our database to life. Initially, we created our database outline in Postgres pgAdmin. This design is the backbone of our database and allows us to outline the relationships between the data, the contents of the tables themselves, as well as define the primary keys of the data. The full document can be found in our deliverables, but once the outline was completed based on our plans stated in the earlier sections, it was now time to fill in the structure with random data.

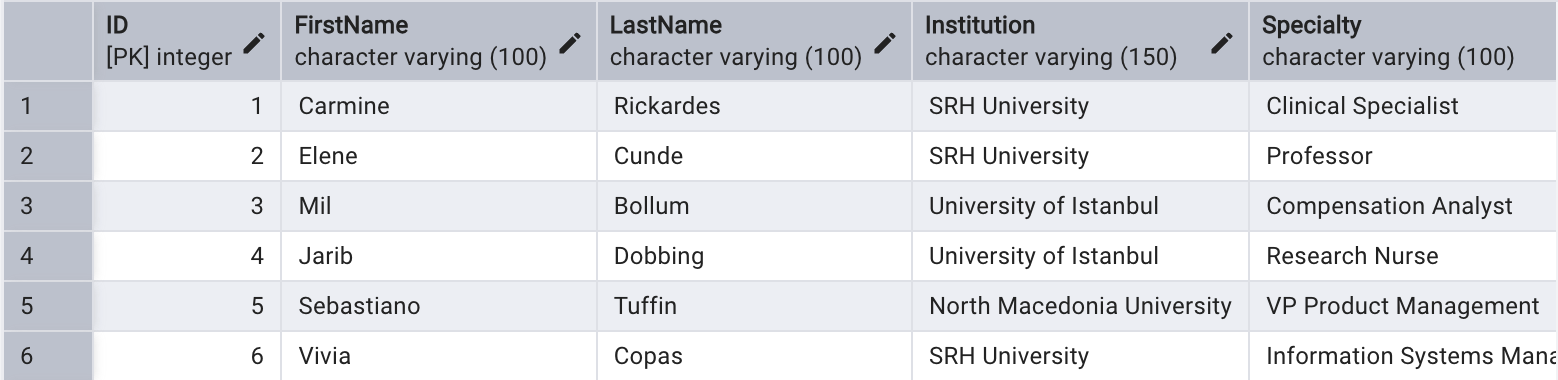
*List of what the tables looked like before being filled with data.*

Now in our PGAdmin screen, we have the empty tables as shown in the image above. Using mockaroo, we were able to fill this data exactly to our specifications. The trick is to ensure that all of the column and table titles match to exactly the specifications in the Relational Diagram. Once you enter in all of the values exactly as specified, and generate the data, you can export it to an SQL document that looks like this:



*Mockaroo Expor*

With this document, you can upload directly into PgAdmin as a query, run it, and then as long as there are no errors (usually not on the first try) the data will fill into the tables. Additionally, the query will give you an error if the data you entered does not follow the business rules outlined in the original database plan, **so you can be sure all the data you entered will only execute if it is correct.** This was a painstaking process for all tables you saw earlier, whether it be through typos or changing values in mockaroo, etc. However, after much trial and error, we repeated this process many times and allowed our data to be full and ready for queries. We decided to keep our data set relatively small, so we could double check accuracy in our queries with most tables ranging from 20-250 entries depending on the category. See below a sample from the Participants table in PgAdmin:

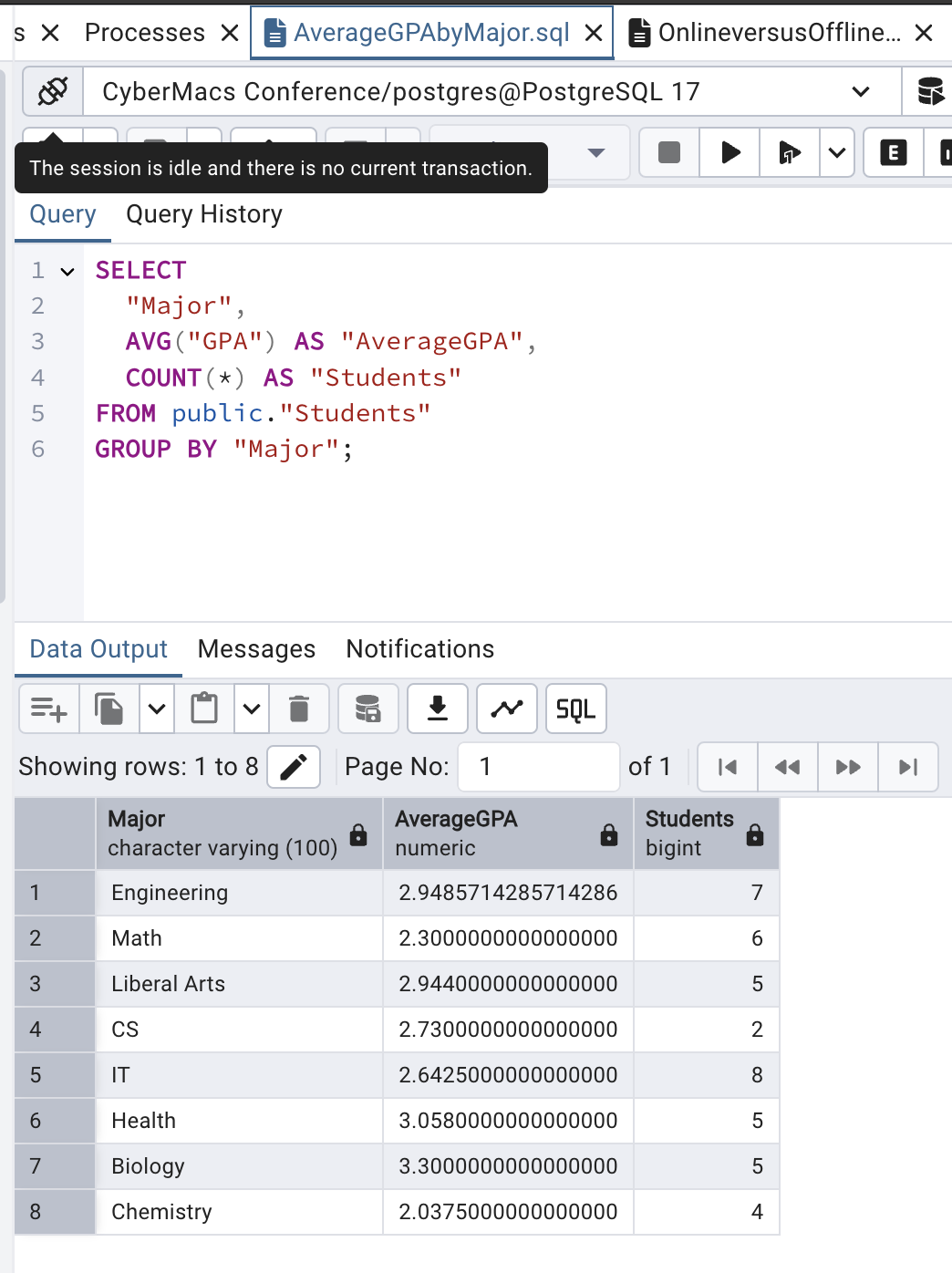


*Screenshot of Participants Table from PGAdmin*

**V. Querying/Reporting**

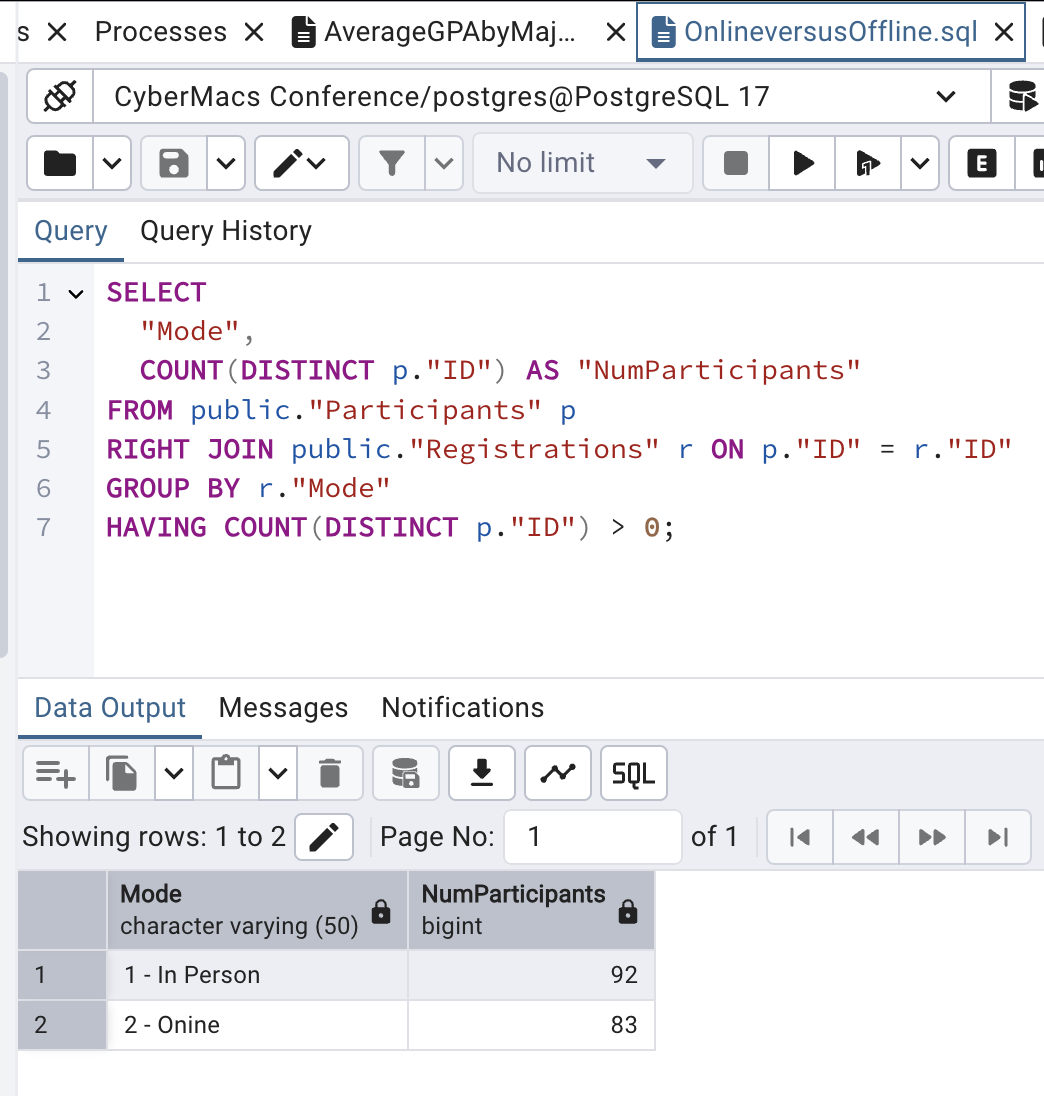
Our queries focused on the view from the event organizer and sponsor perspective, allowing them to see many different highlights from the data, by linking relevant data from multiple tables together. Our queries that we used in this report are as follow:

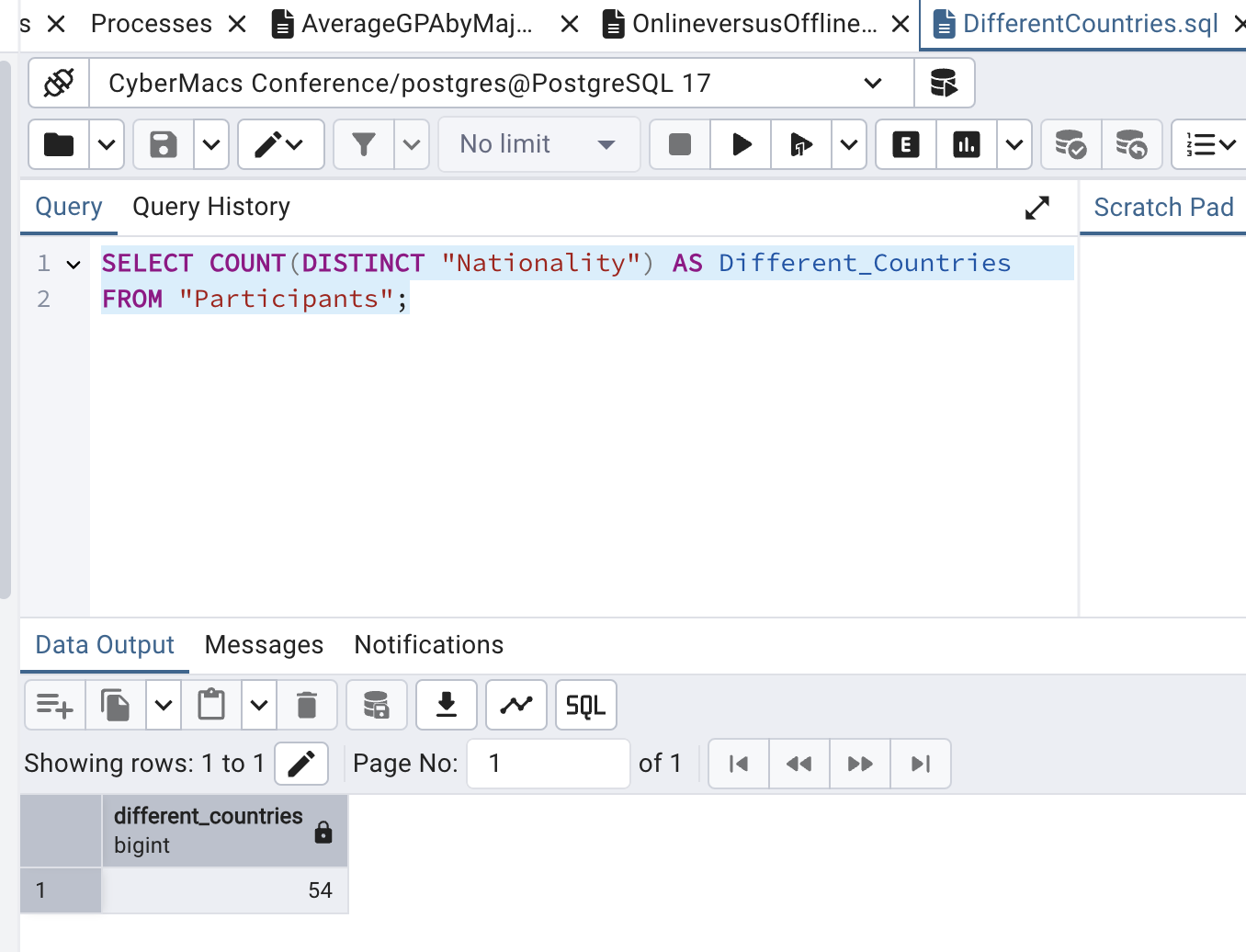
1. The first query we have in our presentation is AverageGPAbyMajor. This query could potentially be used by the Sponsors in order to see the smartest demographic attending and see if it fits their specific industry needs. The query simply SELECTS “Major” from the public ”Students” table and averages the GPA and counts the students grouped by the “Major”. This in turn will produce a chart with columns, “Major”, “AverageGPA” and “Students”. See below for the exact results



*Screenshot AverageGPAbyMajor.sql*

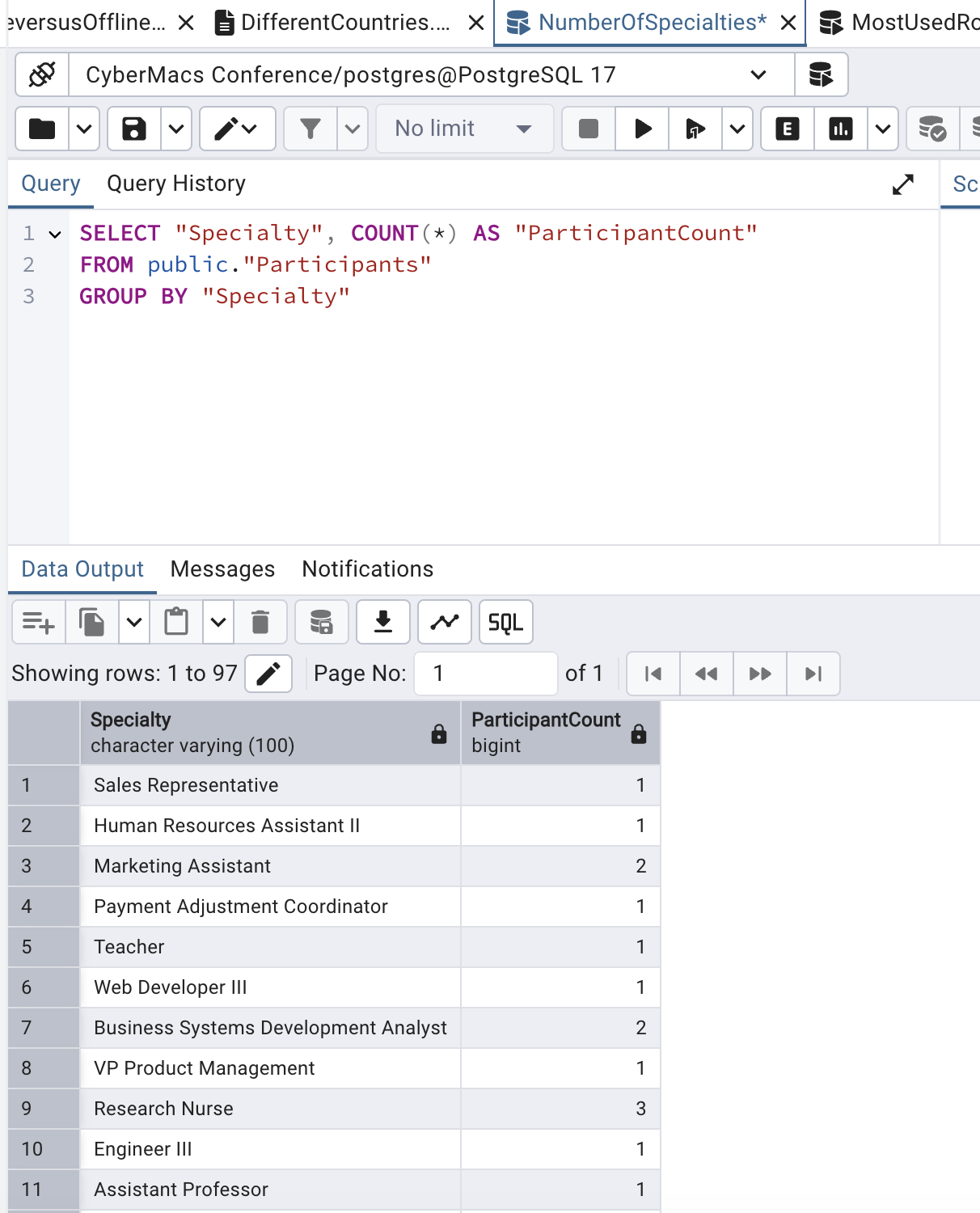
1. This second was rather complicated, as we wanted to see the number of people who were attending the events online versus offline, which would be very useful for event organizers to know. To accomplish this, we had to SELECT “Mode” from the “Participants” table; however, the participants table does not directly have the “Mode” in it. So, we had to join the “Registrations” table to the “Participants” table, in order to be able to count how many participants joined online versus in person. The full query from PgAdmin is listed below:

*OnlineversusOffline.sql*

1. The third query is rather simple as we just aim to see the amount of different countries represented by the participants, a good fact to know and advertise for being international. To do this we simply COUNT DISTINCT “Nationality” values in the Participants table . 

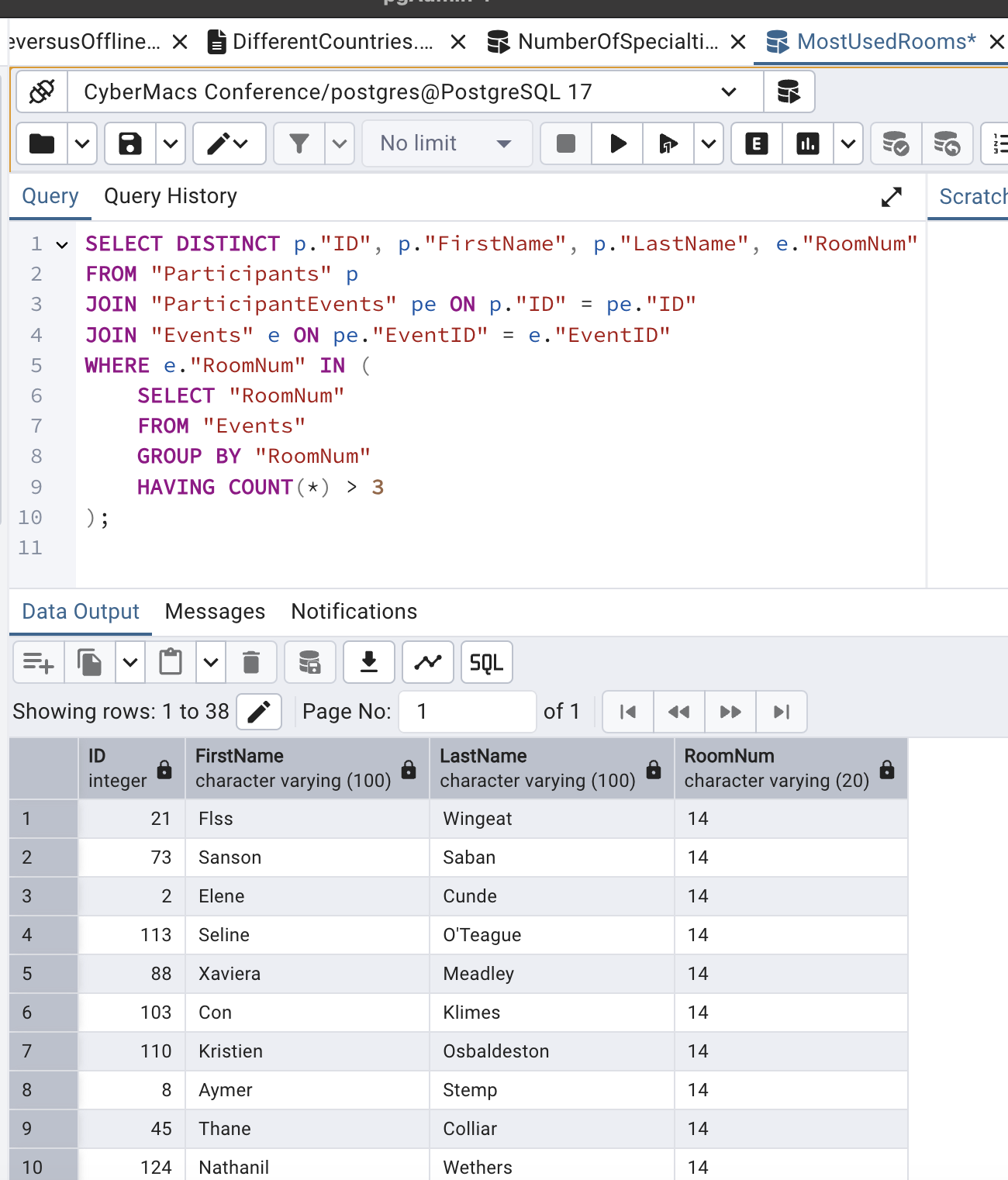
*DifferentCountries.sql*

1. The fourth query allows us to see the number of participants per Speciality, which is extremely important information to know for both Sponsors and Event managers. Now they can advertise and know what will be talked about in the events. To accomplish this we simply write a query that COUNTs the number of “Participants” and GROUPs by “Specialty”. The exact code and output are here:



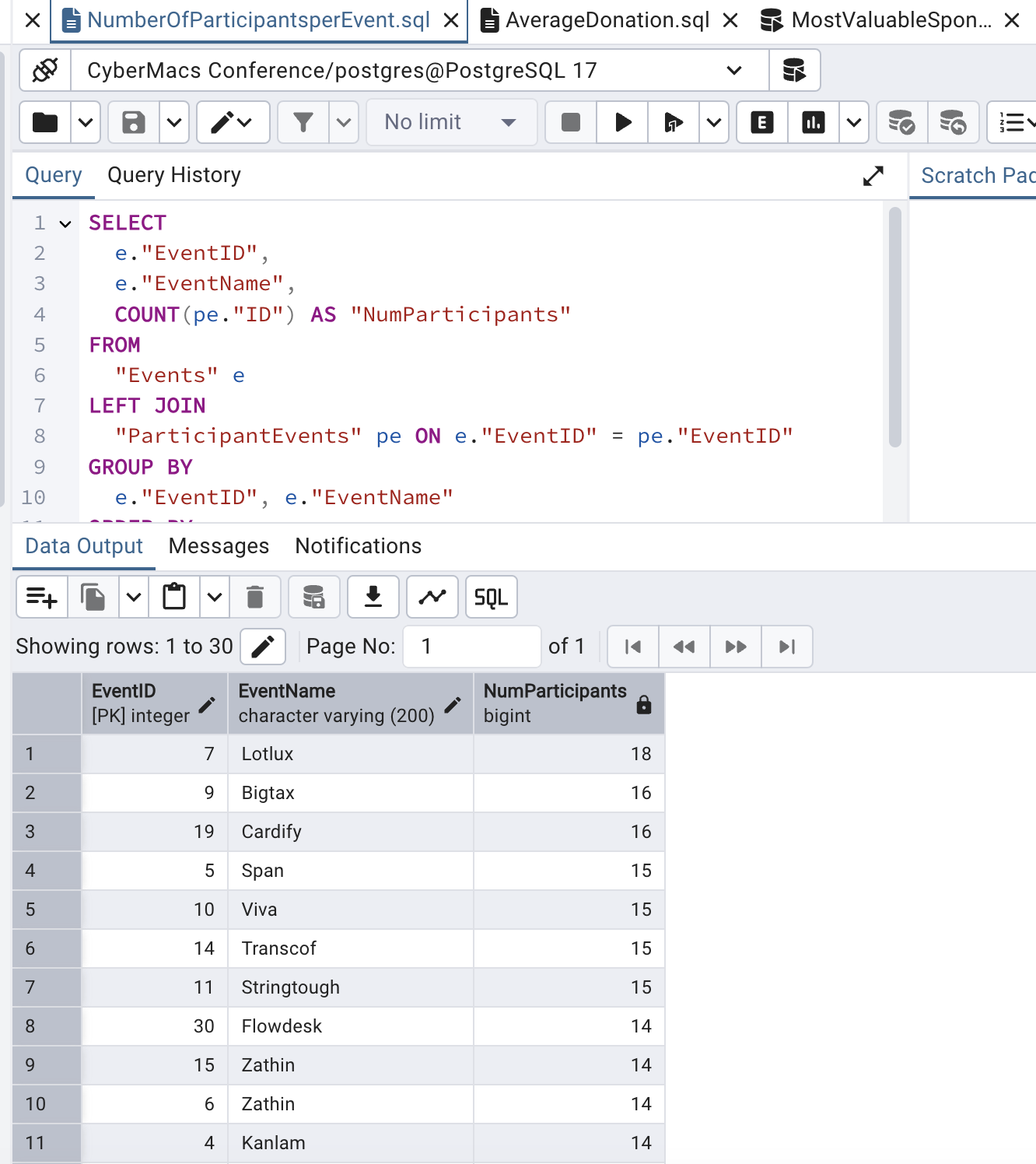
*NumberOfSpecialties.sql*

1. The fifth query is perhaps the most complicated. We wanted all participants who have attended at least one event in a room that's hosted more than three events in total. This would be useful to see which room to book for the more popular events as well as who is going to be at these events. To do this we SELECT DISTINCT values from the Participants table that is joined to the ParticipantsEvents and then to the Events table. We then filter this by showing values only where “RoomNum” has more than 3 events. This was a nested query with multiple joins, definitely very tricky but we are happy to say we figured it out in the end. Here is the full code:



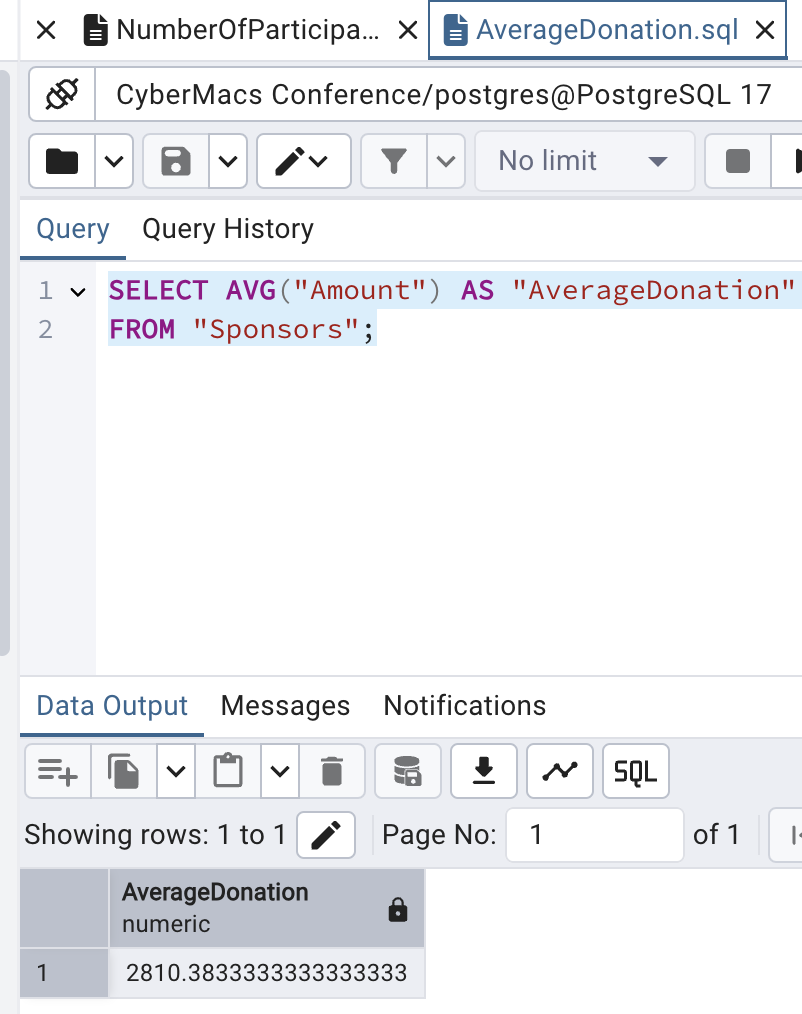
*MostUsedRooms.sql*

1. The sixth query is shows the number of participants per event, this would be extremely useful to know for both the event managers and sponsors, to see which events are most popular and plan accordingly. To do this we simply SELECT the “EventID”, “EventName” and COUNT of participants from the “Event” table which we JOIN to the “ParticipantEvents” table. This will result in a table listing the “EventID”, “EventName” and “NumParticpants”. The full code and results are listed below.



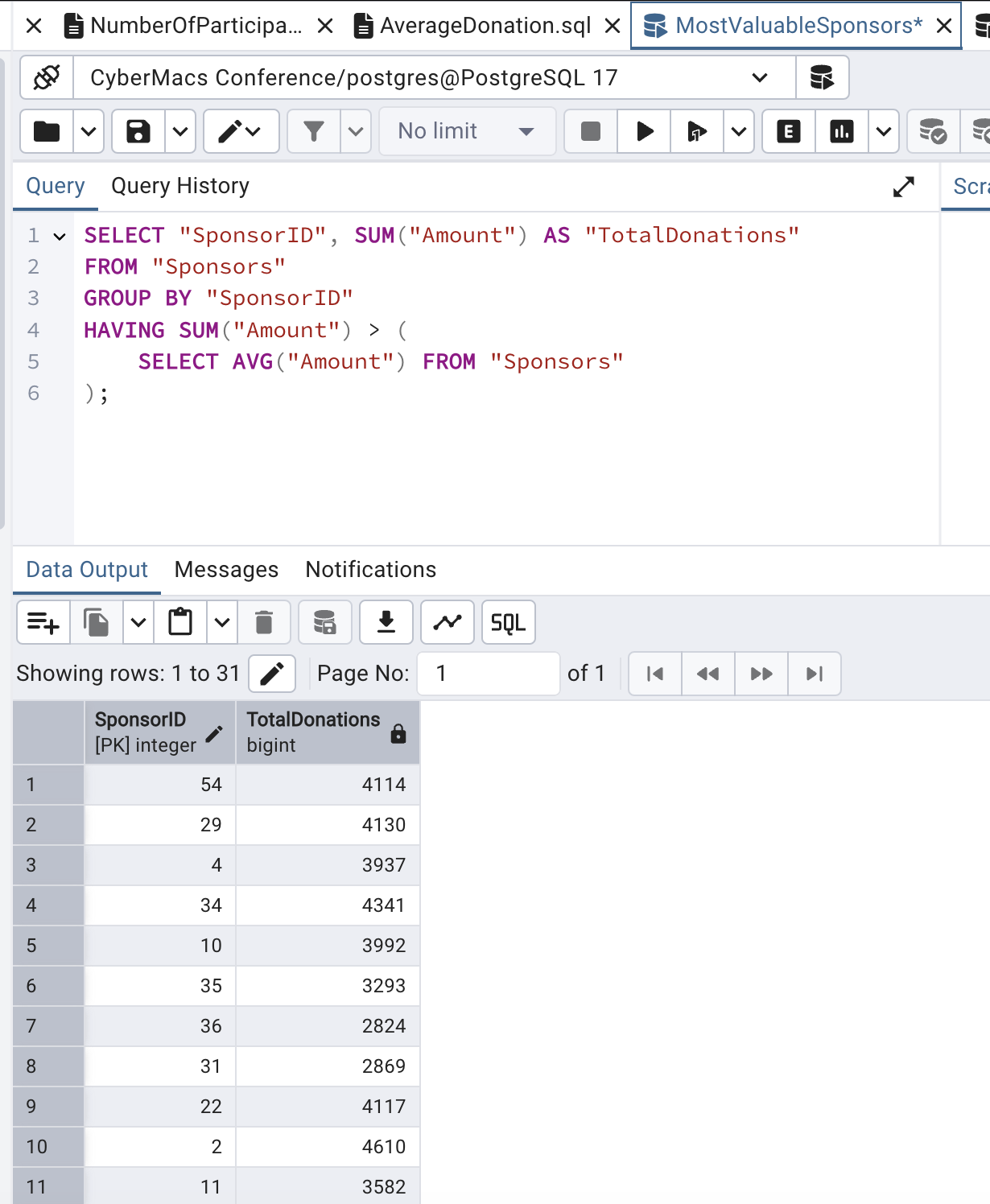
*NumberOfParticipantsperEvent.sql*

1. The seventh query is rather simple and just allows us to see the average donations per sponsor. This simple data point is very useful to know for event managers as well as sponsors, so they both know around how much to expect per client/ transaction. To accomplish this we simply SELECT the AVG totaled from the “Amount” column in the “Sponsors” table. Here is the full code and documentation:

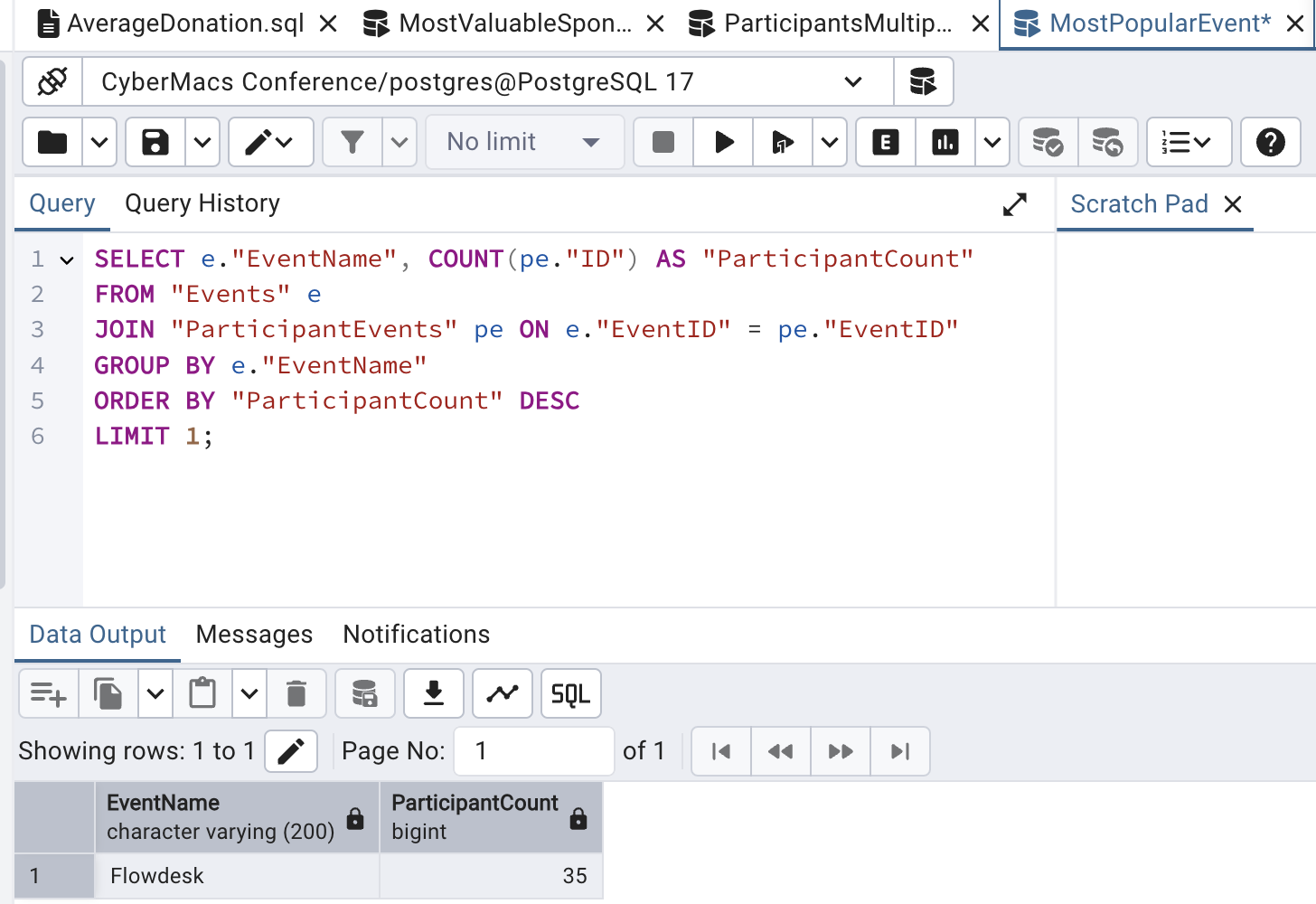


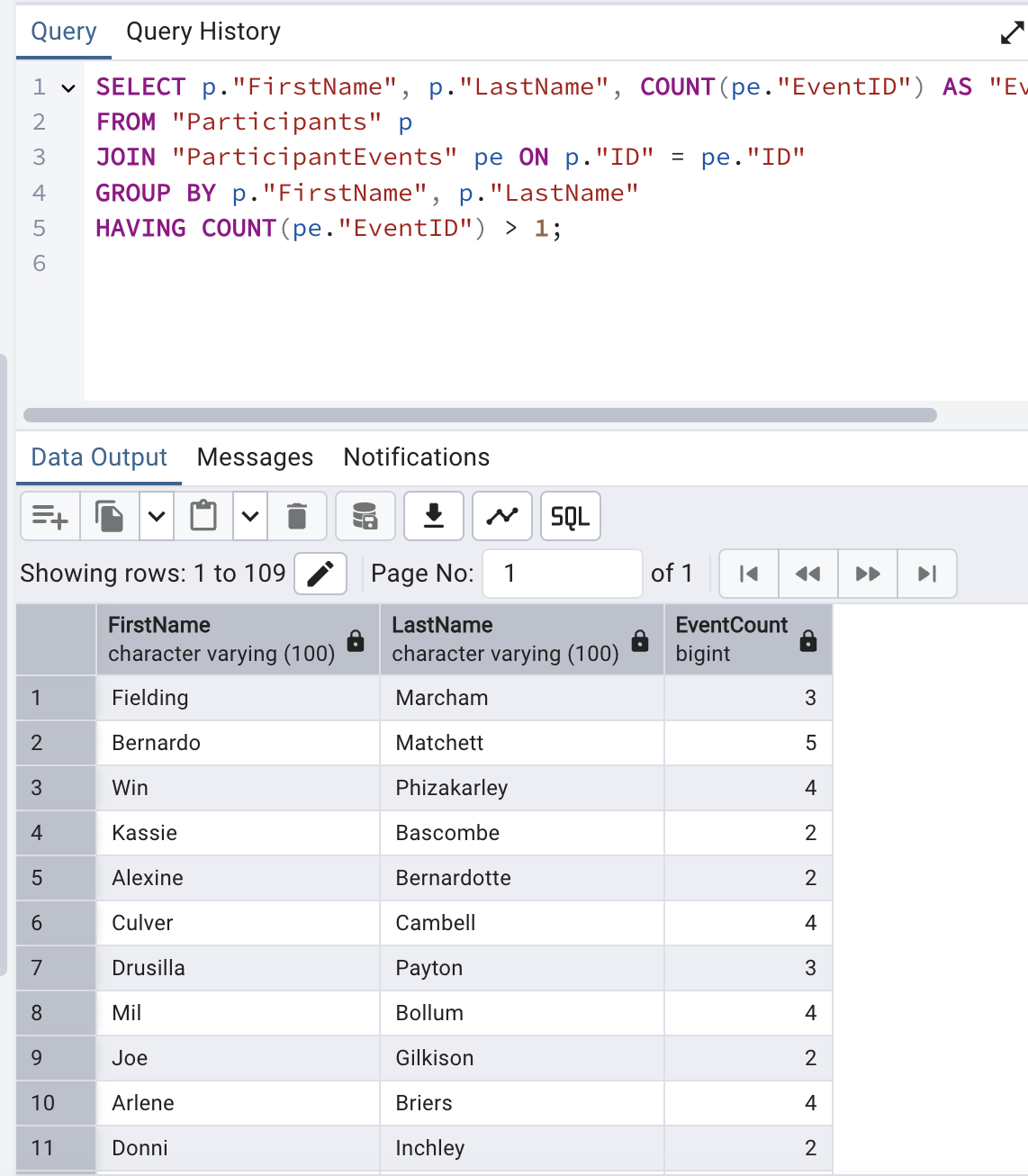
*AverageDonation.sql*

1. The next query builds on this last one slightly, allowing us to see which sponsors are actually over the average threshold, ranking from highest to lowest. This information would be very useful for an event manager, as they can see the VIPs through this data, and know who to look out for. To do this, we simply sum up the total donations by SELECT the total number of donations from the “Sponsor” table, and then we filter by saying the “Sum” MUST HAVE an amount greater then the AVG of “Amount” in Sponsor. The full query and data are listed below.

*MostValuableSponsor.sql*

1. This query shows you which event is most popular and how many participants will be at this event, of course this is valuable information to know for both Sponsors and event managers. To accomplish this we simply SELECT “EventName” and COUNT ID from a table where “Events” and “ParticipantEvents” are joined. We then GROUP BY “EventName” but LIMIT the query so it only displays the highest result. The full query and result are here:

 *MostPopularEvent.sql*

1. The final query shows us which participants are participating in multiple events. This is good to know so sponsors and event managers can see who is the most active and experienced among the people attending all events. To do this, we simply SELECT “FirstName”, “LastName” and count “EventID” in a joined “Participants” and “ParticipantsEvents” table. We also eliminate anyone who is not participating in any events or only one event with the HAVING COUNT >1 field. The full query and result are shown below: 

*ParticipantsMultipleEvents.sql*

**GAI use in the project:**

1. Rephrasing of the summary in the logical design section
2. Brainstorming few ideas for querying scenarios
3. Looking into examples similar to the domains field in data dictionary
4. Debugging errors in the SQL code used to create the DB

**Platforms used:**

1. PostgreSQL
2. [Draw.io](http://draw.io)
3. Mockaroo