

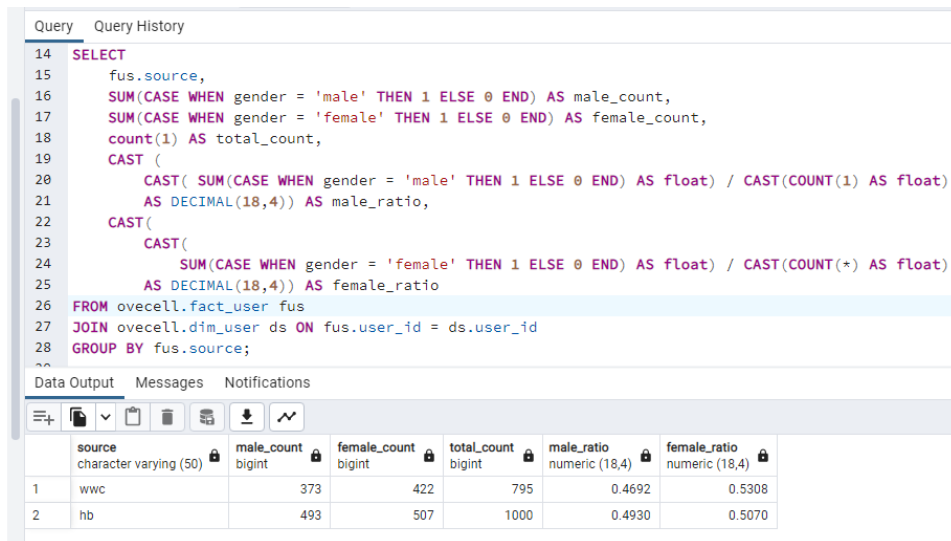
Questions and Validations :

Question#1) Could you find what is the gender ratio in each game?

Query:

```
SELECT
    fus.source,
    SUM(CASE WHEN gender = 'male' THEN 1 ELSE 0 END) AS male_count,
    SUM(CASE WHEN gender = 'female' THEN 1 ELSE 0 END) AS female_count,
    count(1) AS total_count,
    CAST (CAST( SUM(CASE WHEN gender = 'male' THEN 1 ELSE 0 END) AS float) / CAST(COUNT(1) AS float) AS DECIMAL(18,4)) AS male_ratio,
    CAST( CAST(SUM(CASE WHEN gender = 'female' THEN 1 ELSE 0 END) AS float) / CAST(COUNT(*) AS float) AS DECIMAL(18,4)) AS female_ratio
FROM ovecell.fact_user fus
JOIN ovecell.dim_user ds ON fus.user_id = ds.user_id
GROUP BY fus.source;
```

Result



The screenshot shows a SQL query editor with a query window and a results window. The query window contains the following SQL code:

```
14 SELECT
15     fus.source,
16     SUM(CASE WHEN gender = 'male' THEN 1 ELSE 0 END) AS male_count,
17     SUM(CASE WHEN gender = 'female' THEN 1 ELSE 0 END) AS female_count,
18     count(1) AS total_count,
19     CAST (
20         CAST( SUM(CASE WHEN gender = 'male' THEN 1 ELSE 0 END) AS float) / CAST(COUNT(1) AS float)
21         AS DECIMAL(18,4)) AS male_ratio,
22     CAST(
23         CAST(
24             SUM(CASE WHEN gender = 'female' THEN 1 ELSE 0 END) AS float) / CAST(COUNT(*) AS float)
25             AS DECIMAL(18,4)) AS female_ratio
26 FROM ovecell.fact_user fus
27 JOIN ovecell.dim_user ds ON fus.user_id = ds.user_id
28 GROUP BY fus.source;
```

The results window shows the following data:

	source character varying (50)	male_count bigint	female_count bigint	total_count bigint	male_ratio numeric (18,4)	female_ratio numeric (18,4)
1	wwc	373	422	795	0.4692	0.5308
2	hb	493	507	1000	0.4930	0.5070

Question#2) List the youngest and oldest players per country

Query:

SELECT

dl.nationality as country,

MIN(ds.dob) AS oldest_player_dob,

MAX(ds.dob) AS youngest_player_dob

FROM

ovecell.fact_user fus

JOIN ovecell.dim_user ds ON fus.user_id = ds.user_id

JOIN ovecell.dim_location dl ON fus.location_id = dl.location_id

GROUP BY dl.nationality;

Result:

```
34 SELECT
35     dl.nationality as country,
36     MIN(ds.dob) AS oldest_player_dob,
37     MAX(ds.dob) AS youngest_player_dob
38 FROM
39 ovecell.fact_user fus
40 JOIN ovecell.dim_user ds ON fus.user_id = ds.user_id
41 JOIN ovecell.dim_location dl ON fus.location_id = dl.location_id
42 GROUP BY dl.nationality;
```

	country character varying (10)	oldest_player_dob date	youngest_player_dob date
1	IE	1944-09-08	1994-09-16
2	BR	1945-02-13	1995-07-23
3	CA	1944-12-22	1994-06-21
4	DK	1944-11-11	1994-11-17
5	ES	1946-07-29	1995-06-20
6	US	1945-09-12	1993-08-08
7	FR	1944-09-17	1995-05-28
8	GB	1945-07-04	1994-02-04

Total rows: 15 of 15 Query complete 00:00:00.068

Question#3) If you suddenly had millions of new events for the accounts to process per day, how would

you make the data pipeline faster and more scalable and more reliable?

Answer:

There are several ways to make the data pipeline faster, more scalable, and more reliable:

- a) **Parallel Processing**: One of the most effective ways to handle increased data load is by parallel processing. With tools like Apache Spark, you can distribute data processing tasks across multiple nodes. This helps in faster processing of large amounts of data.
- b) **Batch Processing**: Instead of processing each event as it comes in, you can group events together and process them in batches. This can be more efficient because it reduces the overhead of starting and stopping the processing task for each event.
- c) **Using a Stream Processing Framework**: If the data needs to be processed in near real-time, you can use a stream processing framework like Apache Kafka or Apache Flink. These frameworks are designed to handle high volumes of data and provide features like backpressure to prevent data overload.
- d) **Proper Indexing**: In the database, proper indexing can help in faster query execution which can make the pipeline faster.
- e) **Data Partitioning**: Partitioning the data based on some logical divisions like date, user ID, etc., can help in faster querying and data retrieval.
- f) **Ensuring Data Quality**: By implementing checks and validations in the pipeline to ensure data quality, we can make the pipeline more reliable. This will help to catch any issues with the data early in the pipeline before it affects downstream processes.
- g) **Monitoring and Alerting**: Implementing a robust monitoring and alerting system can help in quickly identifying any issues with the pipeline and rectifying them before they impact the entire process.

Bonus Question#4) Can you summarise a list of principles you would follow when developing an event pipeline?

Answer:

- a) **Scalability**: The pipeline should be able to handle increased data load without significant degradation in performance
- b) **Resilience**: The pipeline should be able to recover from failures without data loss.
- c) **Data Integrity**: The pipeline should ensure that the data is accurate, consistent, and reliable.
- d) **Efficiency**: The pipeline should be designed to process data as efficiently as possible to minimize resource usage.
- e) **Automated Testing**: Implement robust testing for each component of the pipeline to catch any issues early.
- f) **Monitoring and Alerting**: Implement a system to monitor the health and performance of the pipeline and alert if any issues arise.
- g) **Documentation**: Properly document the design and operation of the pipeline so that it's easier to understand, maintain, and troubleshoot.
- h) **Security**: Implement appropriate security measures to protect sensitive data and prevent unauthorized access.

- i) **Ease of Use**: Design the pipeline in a way that is easy to use for the end-users, which in this case would be data analysts or other data consumers.