INFS3200 Advanced Database Systems

**Prac 1: Distributed Databases (5%)**

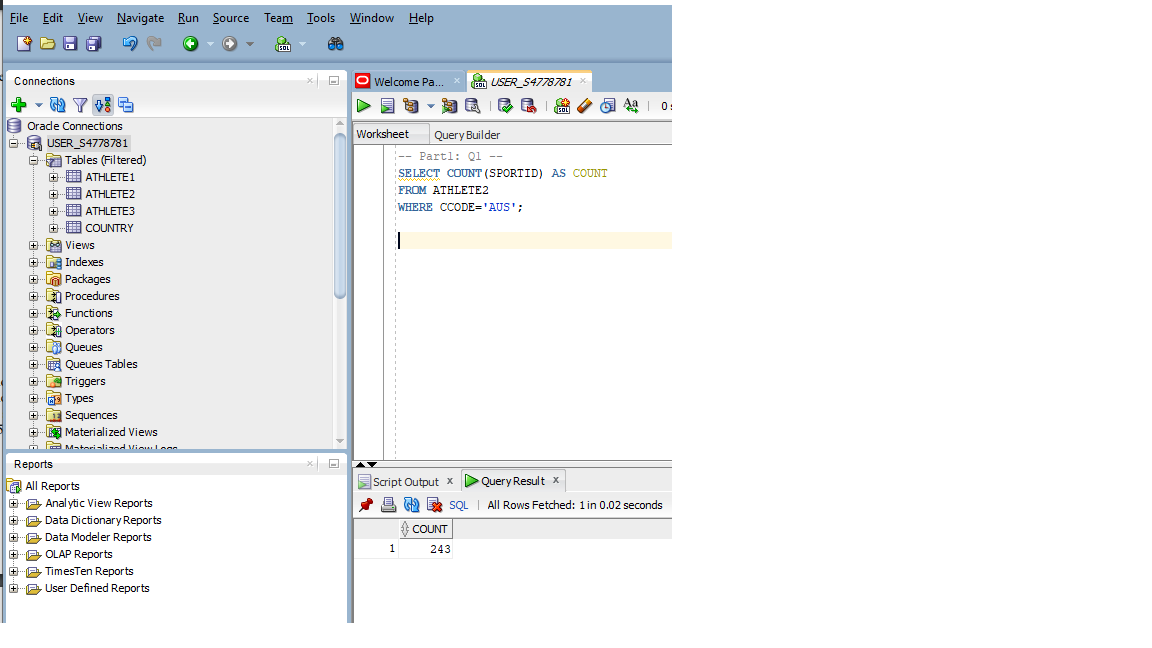
*Semester 1, 2023*

**Student’s name: Nguyen Van Quoc Chuong**

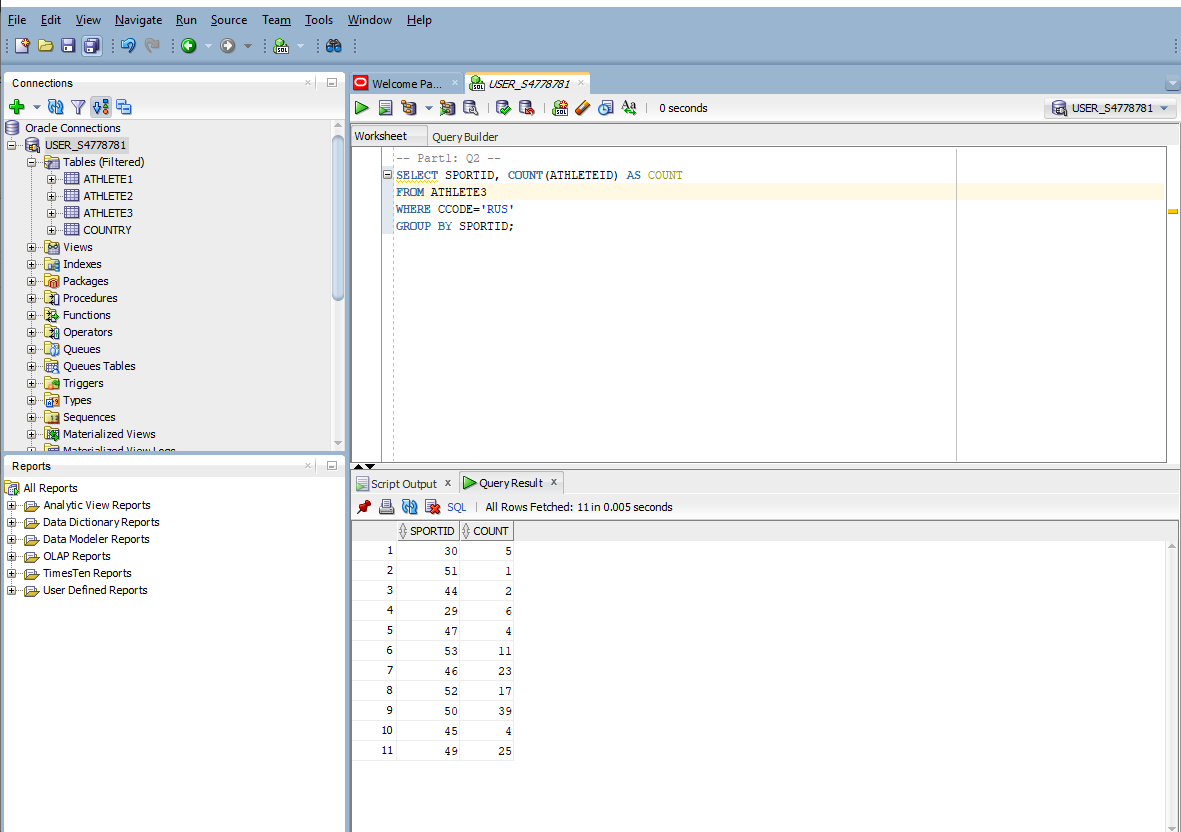
**ID: 47787810**

**Part 1:**

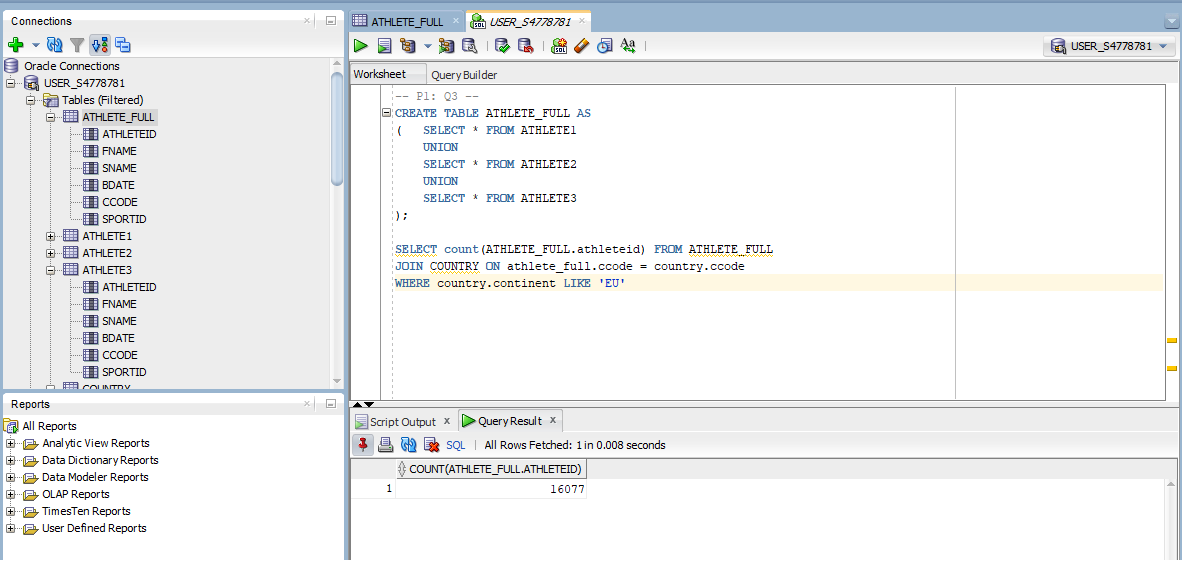
**Task 1:**

 *Question 1: Count the number of players from Australia (country code=AUS) in Athlete2 table.*

Explanation: I use the SELECT, WHERE commands with the condition CCODE = ‘AUS’ to choose specific rows FROM the table, then COUNT all of tuples to get the result.

 *Question 2: For all Russian (RUS) players in table Athlete3, count the number of players participating in each sport.*

Explanation: same as question 1, however, I use GROUP BY to group rows with the same SportId together.

 *Question 3: Create a new table named “ATHLETE\_FULL” which combines all records from tables Athlete1, 2 and 3. Use this table along with the country information from the Country table to count the total number of players from Europe.*

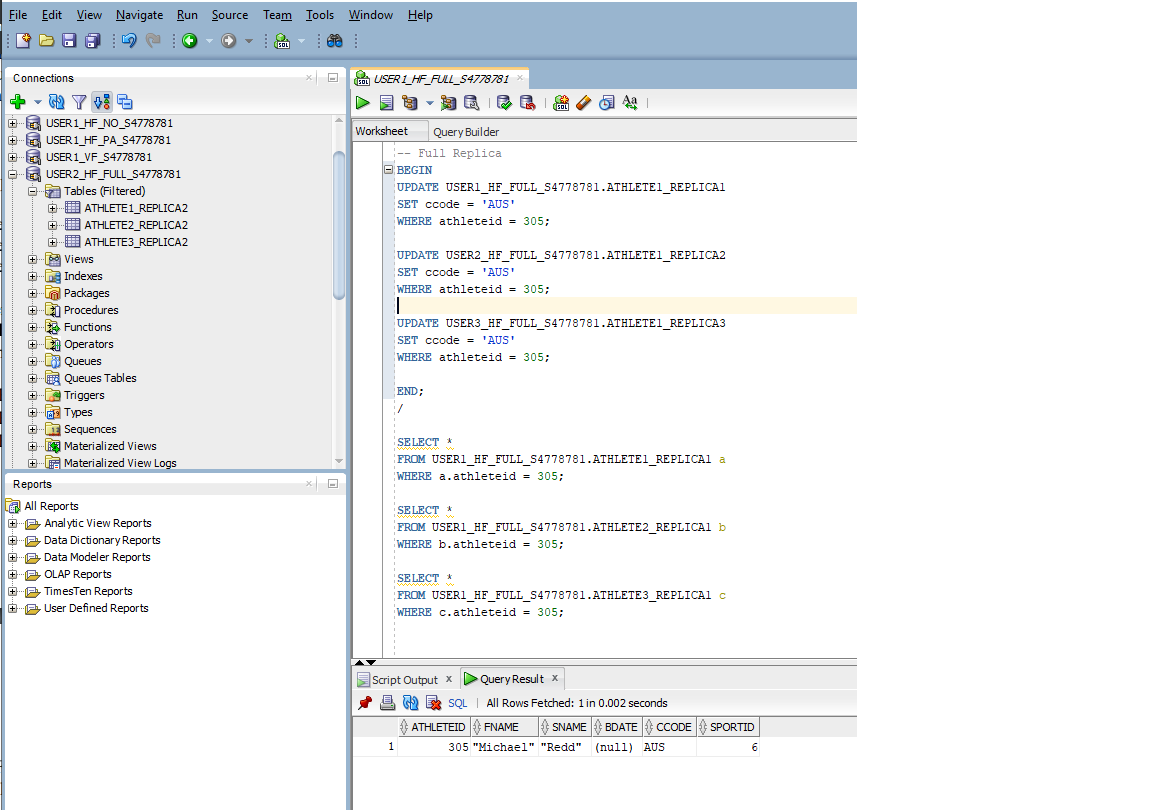
Explanation: First, I create a table ATHLETE\_FULL containing all records from 3 tables (UNION), then use JOIN operation to map with athlete's country information in the COUNTRY table, and using WHERE to select specific rows with continent = ‘EU’.

**Part 2:**

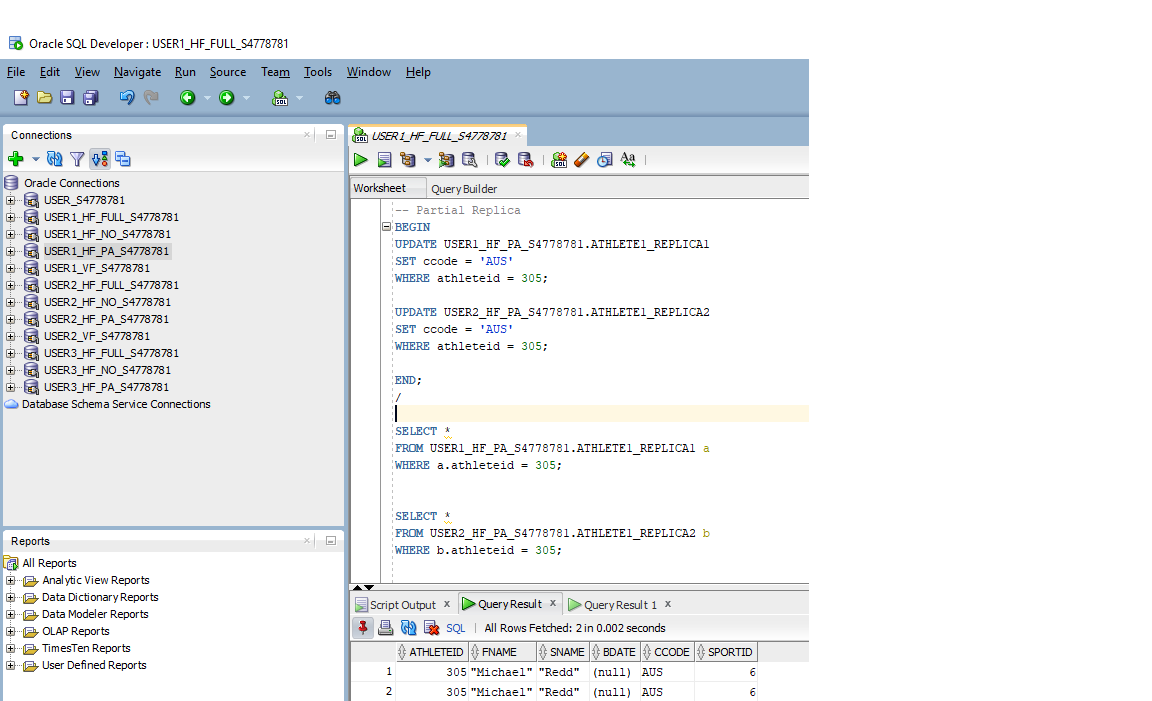
**Task 2:** Given the update query below, write a set of SQL queries (or one transaction preferably) which applies this update to the system under each replication strategy, respectively. (Hint: Three sets of SQL queries (or three transactions) in total for three different strategies. Each of your update transaction should guarantee consistency between copies and should not perform update to sites which are not possible to have the record).

*Query: Change the country code of the player whose ID is 305 to “AUS”.*

Full Replication strategy:

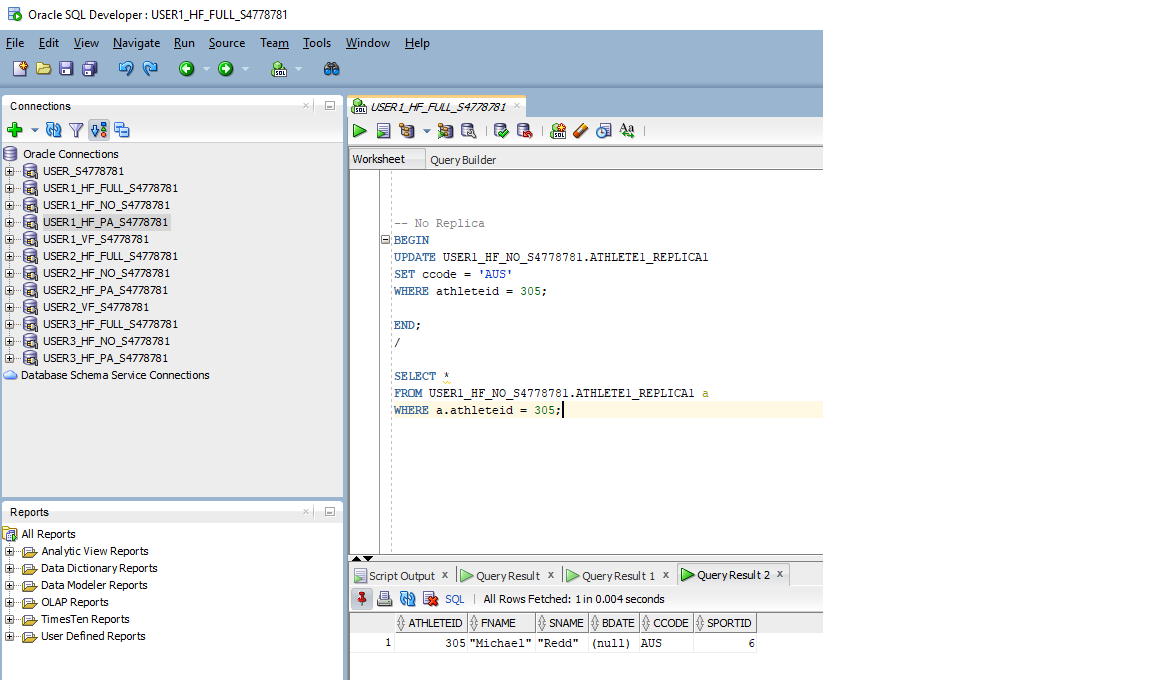


Explanation: Due to the provided fragmentation condition, I already know which table the athletes with id equal to 305 belong to. I create a transaction to update CCODE in all the tables containing that athlete with id 305 to ensure the *correctness*.

 Partial Replication strategy:

Explanation: Same as Full Replication strategy but I just need to update 2 tables ATHLETE1\_REPLICA1 and ATHLETE1\_REPLICA2 in User 1 and 2 respectively.

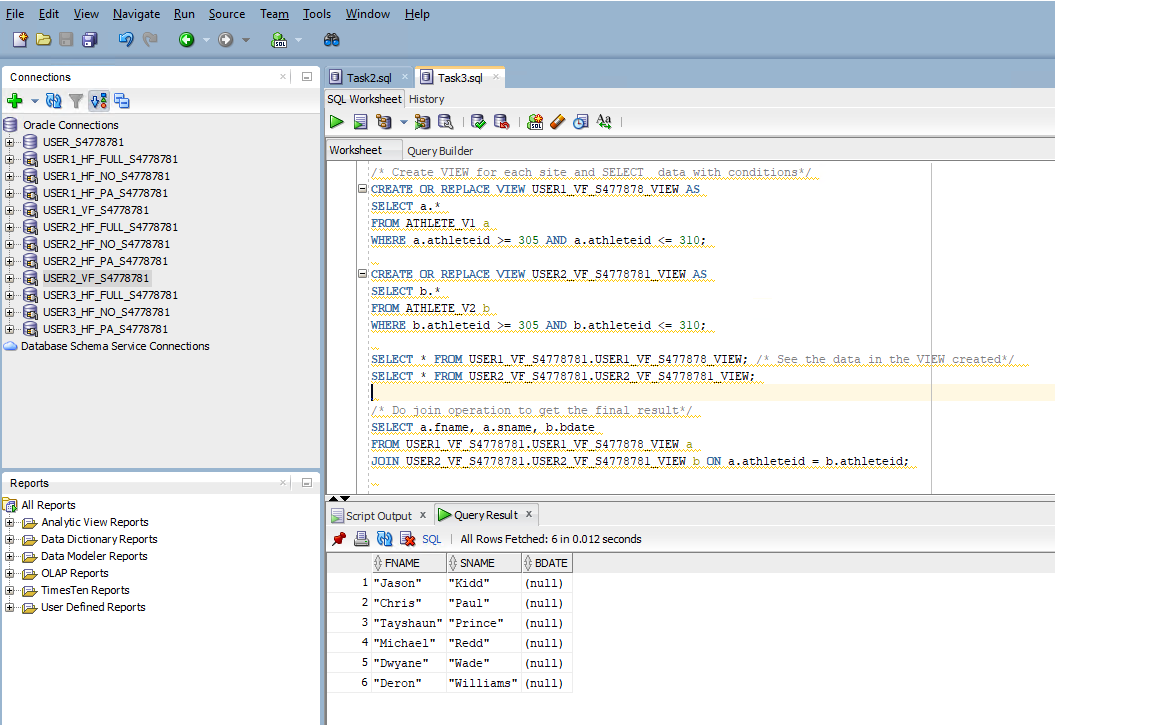
No Replication strategy:



Explanation: I just only need to update ATHLETE1\_REPLICA1 only because of no replication in other tables.

Comparison: The differences in updating transactions among full replication, partial replication, and no replication in horizontal fragmentation depend on the number of copies of a fragment, the location of the fragments, and the type of updates. Full replication involves updating all copies of a fragment, while partial replication updates only some copies, and no replication updates only the local fragment copy. Therefore, in term of performance of update operation, I think full replication strategy is the best because it eliminates the need for communication between fragments, however, the space to store data needs to be larger than other strategies.

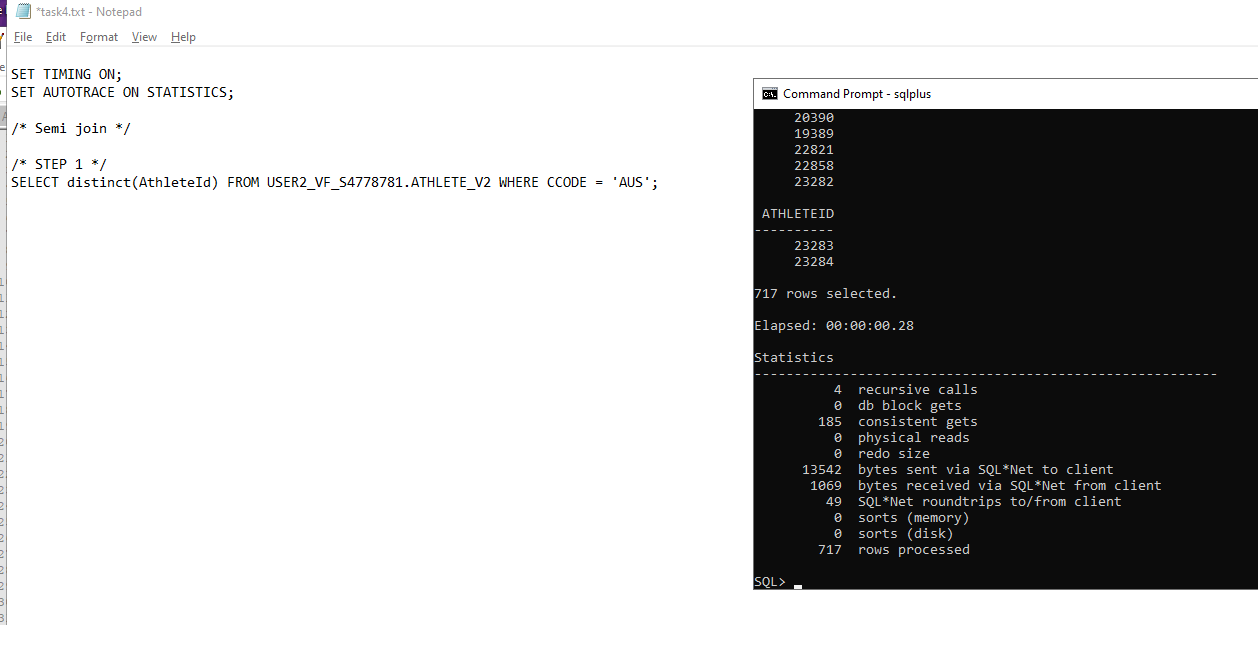
**Task 3:** *Write a SQL query to retrieve the full name and DOB of all the athletes satisfying 305<= AlthleteID <=310*



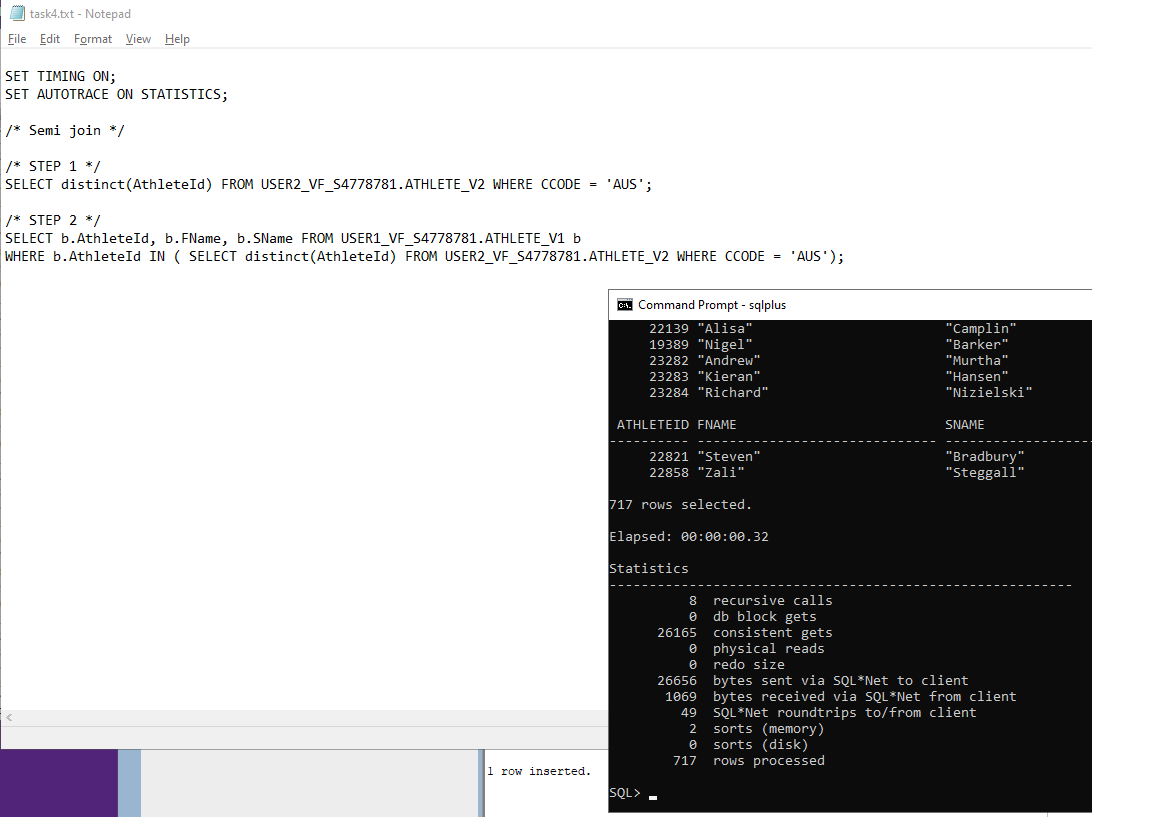
Explanation: In vertical fragmentation, I create 2 views of projection rows User1 and User2 with the conditions applied to AthleteId, then do a join to get the final result. Another method is that I can join the entire tables together, but performance will decrease due to the overhead of the join operation.

**Task 4:**

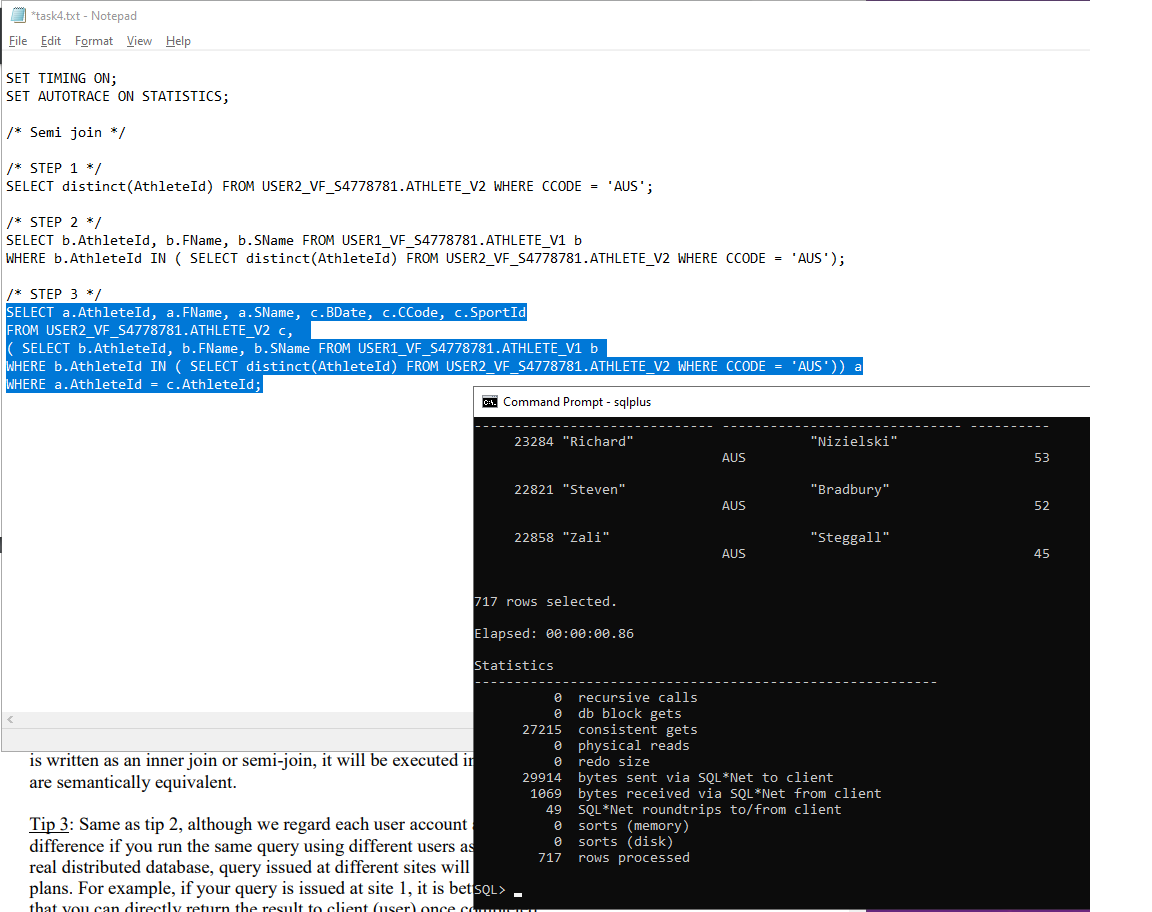
Step 1: Select a specific column (AthleteId) with the condition from User2.

Semi-join operation: 

Step 2: Send the result from the subquery above to User1 and do the semi-join.

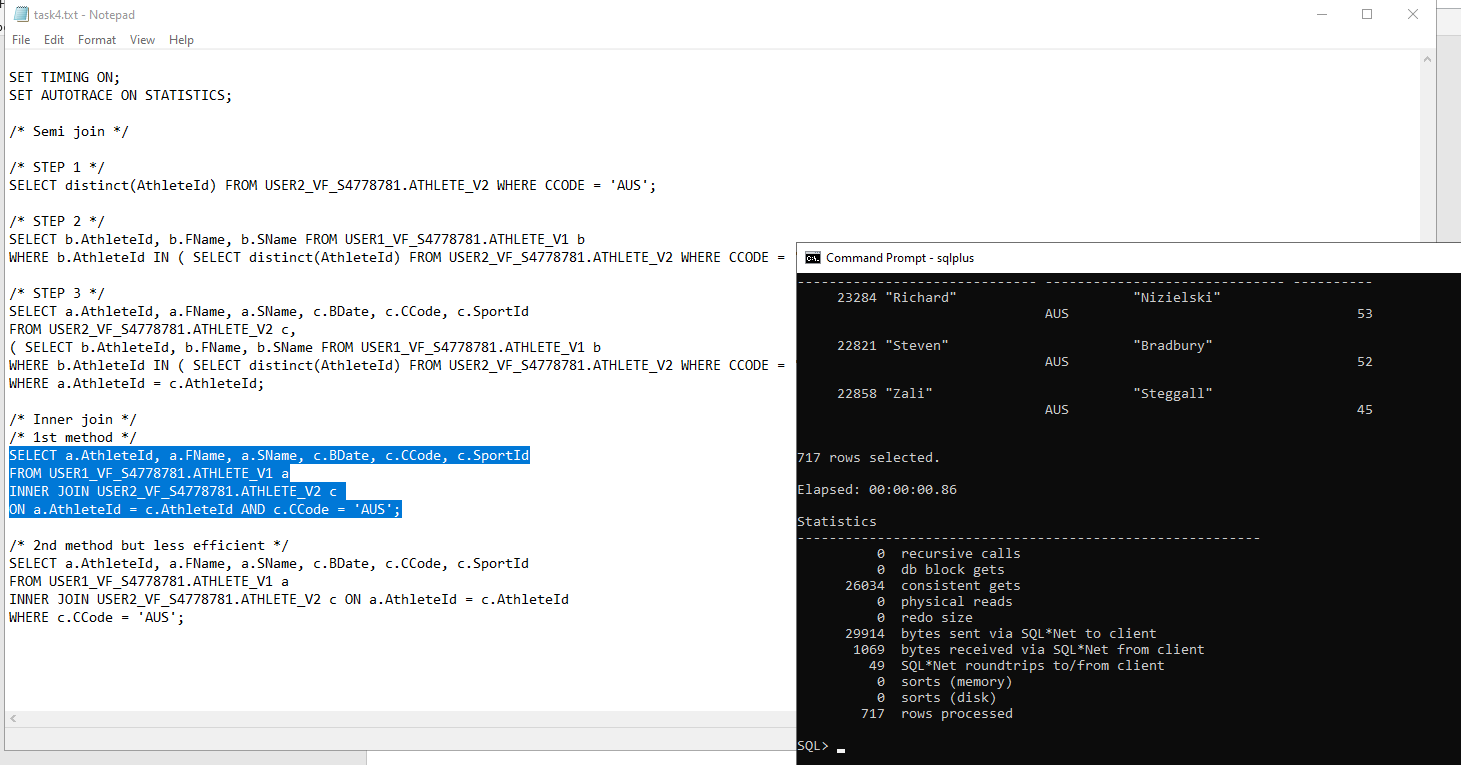


Step 3: Do the final join in User2;



Cost calculation: 13542(step 1) + 26656(step2) = 40198 > 29914 (step 3), however, adding two independent cost estimates, which may not be directly comparable, because the total cost of executing these statements together may be less than the sum of their individual costs because some blocks may be reused or cached, and there are many other that can affect to the execution cost of a query statement.

Inner-join:



There are 2 methods performing the inner join, the 1st method is slightly faster because of reducing the number of rows that need to be processed.

Comparison: I see that the execution time between Semi join and Inner join is quite similar although there is a slight difference in many runs. I think in this case, Inner join is better because it only joins the relevant rows directly based on the conditions "c.CCode = 'AUS'" and “AthleteId” in the join clause. In contrast, the semi-join first needs to find the distinct AthleteId values from User2 with condition CCode = ‘AUS’, then join two tables. So, it involves extra processing cost before the actual join.