INFS3200/7907 Advanced Database Systems

Prac 1: Distributed Databases (5%)

Semester 2, 2020

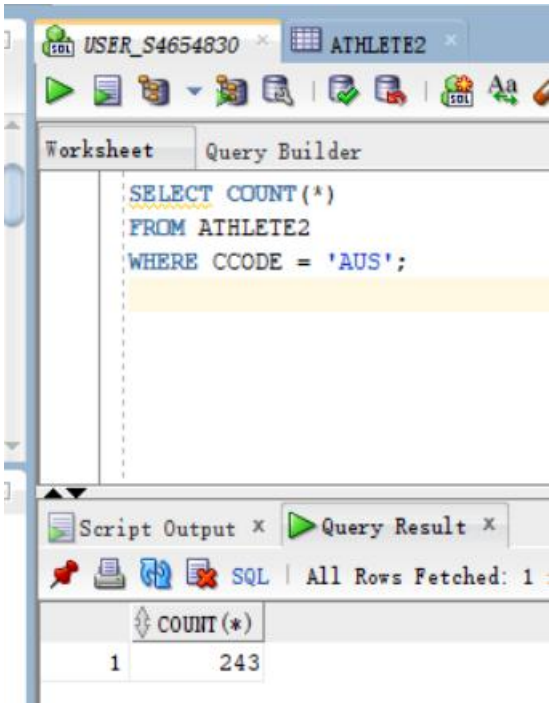
Student ID: 17998819
Student Name: Xuanche.Liu
Date: 15/09/2020

Answer book(After install and create database and user) :

Task 1: Write SQL queries to answer the following questions. *Your queries and the result screenshots should be included in your submission.*

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

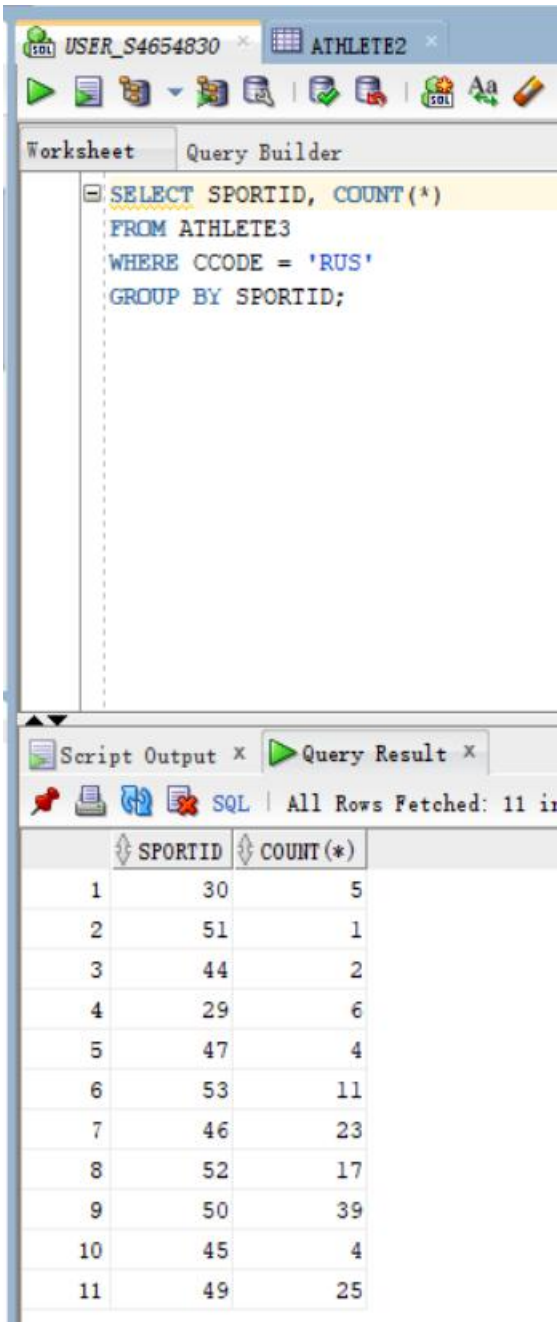
Screenshot and answer:



(2) For all Russian (RUS) players in table Athlete3, count the number of players participating in each sport. The result should be a list containing records like:

	Sport ID	Count
1		
2		

Screenshot and answer:



(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.

Create table:

Code:

USER_S4654830 | ATHLETE1

Worksheet | Query Builder

```
CREATE TABLE ATHLETE_FULL
  (
  ATHLETEID NUMBER,
  FNAME VARCHAR2(255),
  SNAME VARCHAR2(255),
  BDATE VARCHAR2(255),
  CCODE VARCHAR2(255),
  SPORTID NUMBER
  )
```

Create Results:

Connections

- Tables (Filtered)
  - ATHLETE_FULL
    - ATHLETEID
    - FNAME
    - SNAME
    - BDATE
    - CCODE
    - SPORTID
  - ATHLETE1

USER_S4654830 | ATHLETE_FULL

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | De

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT
1	ATHLETEID	NUMBER(38,0)	Yes	(null)
2	FNAME	VARCHAR2(255 BYTE)	Yes	(null)
3	SNAME	VARCHAR2(255 BYTE)	Yes	(null)
4	BDATE	VARCHAR2(255 BYTE)	Yes	(null)
5	CCODE	VARCHAR2(255 BYTE)	Yes	(null)
6	SPORTID	VARCHAR2(255 BYTE)	Yes	(null)

Join table together:

```
insert into ATHLETE_FULL
select * from ATHLETE1
union all
select * from ATHLETE2
union all
select * from ATHLETE3;
```

脚本输出 | 任务已完成，用时 0.098 秒

```
        that was not valid for common users or roles.  In addition to
        the usual rules for user and role names, common user and role
        names must start with C## or c## and consist only of ASCII
        characters.
*Action:   Specify a valid common user or role name.
24,591 行已插入。
```

Search results:

```
SELECT COUNT(ATHLETE_FULL.ATHLETEID)
FROM  COUNTRY , ATHLETE_FULL
WHERE COUNTRY.CCODE =  ATHLETE_FULL.CCODE
AND COUNTRY.CONTINENT = 'EU';
```

查询结果

SQL | 提取的所有行: 1, 用时 0.009 秒

	COUNT(ATHLETE_FULL.ATHLETEID)
1	16077

Job 1 - Full Replication

The data is split into three fragments:

- Athlete1: 1<= AthleteID < 7656
- Athlete2: 7657<= AthleteID < 17318
- Athlete3: 17319 <= AthleteID <= 24591

Each fragment will be a relation located on every site in the computer network (i.e. **each site has a full copy of each fragment**). You should create three sites to simulate the full replication in SQL Plus command line:

- USER1_HF_FULL_S4654830
- USER2_HF_FULL_S1234567
- USER3_HF_FULL_S1234567

To load fragments into site USER1_HF_FULL_S1234567, connect to user "USER1_HF_FULL_S1234567" in SQL Developer and run all script files in folder "…\P1\Part 2\HF\HF-Full\USER1_HF_FULL\". Repeat the same process for other sites.

Job one code:

```
ALTER SESSION SET "_ORACLE_SCRIPT"=TRUE;
CREATE USER USER1_HF_FULL_S4654830 IDENTIFIED BY w ACCOUNT UNLOCK DEFAULT TABLESPACE "USERS" TEMPORARY TABLESPACE "TEMP" PROFILE "DEFAULT";
GRANT DBA TO USER1_HF_FULL_S4654830;
SELECT USERNAME FROM DBA_USERS;
```

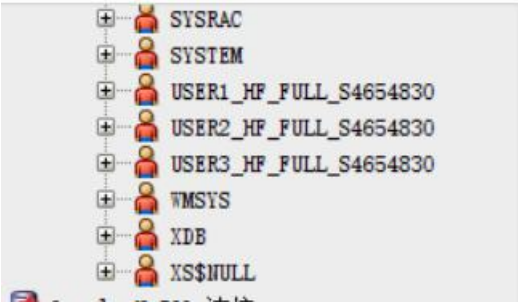
```
Table USER1_HF_FULL_S4654830.TB 已创建。

Table USER2_HF_FULL_S4654830.TB 已创建。

Table USER3_HF_FULL_S4654830.TB 已创建。
```

(Sorry about the Chinese , because my laptop system was broken, i install the system again but the language can's change to English. So sorry about this)

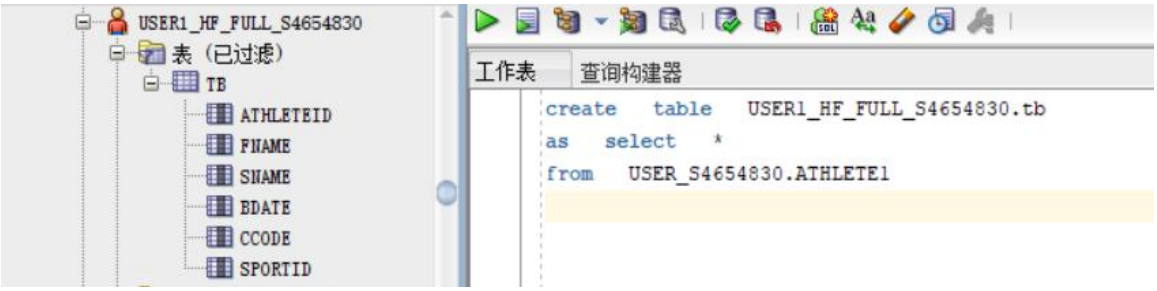
We can find the user created on other user part:

Repeat this step to create three user for job 1:

Results:



USER TABLE DETAIL:



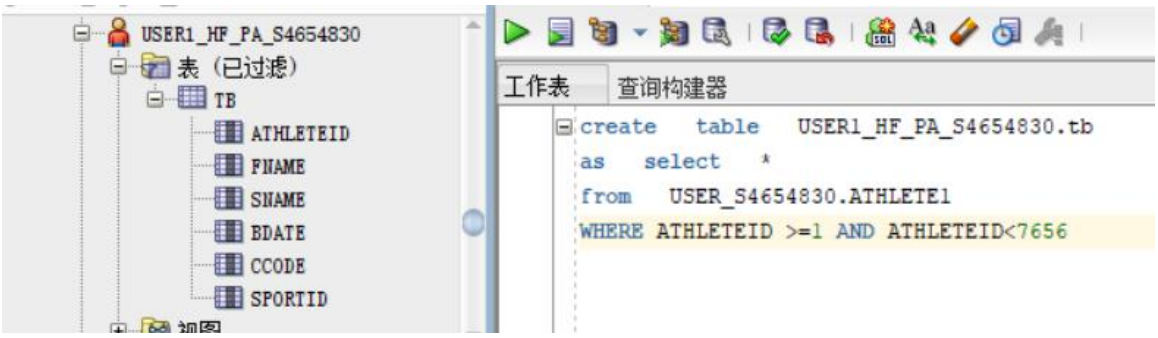
Job 2 – Partial Replication

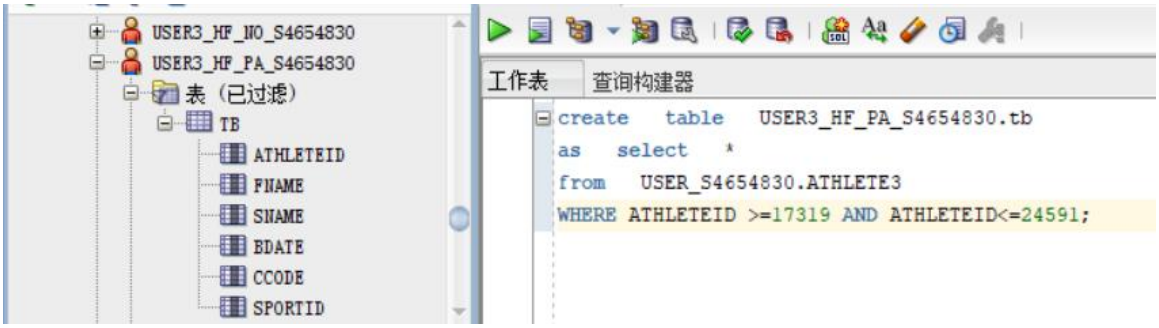
The data is split into three fragments in the same way as in Job 1.

Each fragment will be a relation located on some of the sites in the computer network (i.e., **more than one site may have a copy of this fragment, but not all of them. You should read through the scripts to understand how fragments are replicated and allocated**). You should create three sites:

- USER1_HF_PA_S1234567
- USER2_HF_PA_S1234567
- USER3_HF_PA_S1234567

User create code and results:





Job 3 – No Replication

The data is split into three fragments in the same way as in Job 1.

Each fragment will be a relation located on **only one site** in the computer network. You should create three sites:

- USER1_HF_NO_S1234567
- USER2_HF_NO_S1234567
- USER3_HF_NO_S1234567

User create code and results:

<u>Task 2</u>: 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”.

Because the id is 305, so i just deal with user1_ ```

Update Full code:

```
update USER1_HF_FULL_S4654830.TB
set TB.CCODE = 'AUS'
WHERE TB.ATHLETEID = 305;
```

SELECT RESULTS:

```
SELECT USER1_HF_FULL_S4654830.TB.ATHLETEID,USER1_HF_FULL_S4654830.TB.CCODE
FROM USER1_HF_FULL_S4654830.TB
WHERE ATHLETEID = 305;
```

查询结果 ×

SQL | 提取的所有行: 1, 用时 0.009 秒

	ATHLETEID	CCODE
1	305	AUS

Update PA code:

```
update USER1_HF_PA_S4654830.TB
set TB.CCODE = 'AUS'
WHERE TB.ATHLETEID = 305;
```

Select results:

```
SELECT USER1_HF_PA_S4654830.TB.ATHLETEID,USER1_HF_PA_S4654830.TB.CCODE
FROM USER1_HF_PA_S4654830.TB
WHERE ATHLETEID = 305;
```

脚本输出 × 查询结果 ×

SQL | 提取的所有行: 1, 用时 0.002 秒

	ATHLETEID	CCODE
1	305	AUS

Update NO code:

```
update USER1_HF_NO_S4654830.TB
set TB.CCODE = 'AUS'
WHERE TB.ATHLETEID = 305;
```

Select result:

```
SELECT USER1_HF_NO_S4654830.TB.ATHLETEID,USER1_HF_NO_S4654830.TB.CCODE
FROM USER1_HF_NO_S4654830.TB
WHERE ATHLETEID = 305;
```

脚本输出 × 查询结果 ×

SQL | 提取的所有行: 0, 用时 0.001 秒

ATHLETEID	CCODE

Explain:

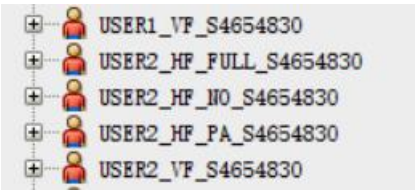
So, based on these three update methods, i can find the full replication is always select all the data, it's always cost more time, and the PA join is good at this update , because we divided the data at different level, so the time cost is more less, finally , the no replication mean no cost , also mean no update. In summary, the full replication 's query processing is much more easier than other, also have the high reliability, but the PA and No replication is more difficulty, the different between PA and No is PA is difficult to control but have higher reliability.

2. Vertical Fragmentation

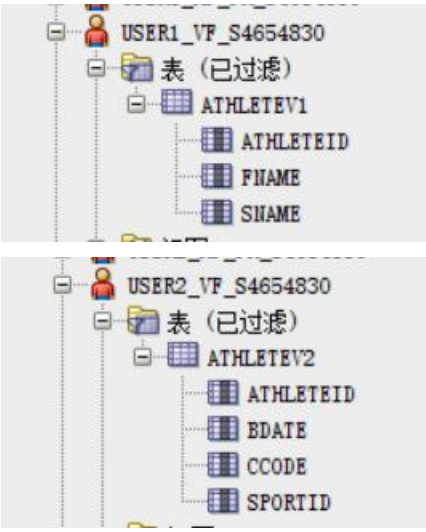
Vertical Fragmentation:

- AthleteV1[AthleteID, FName, LName]
- AthleteV2[AthleteID, DOB, CountryCode, SportID]

Create user and import data results:

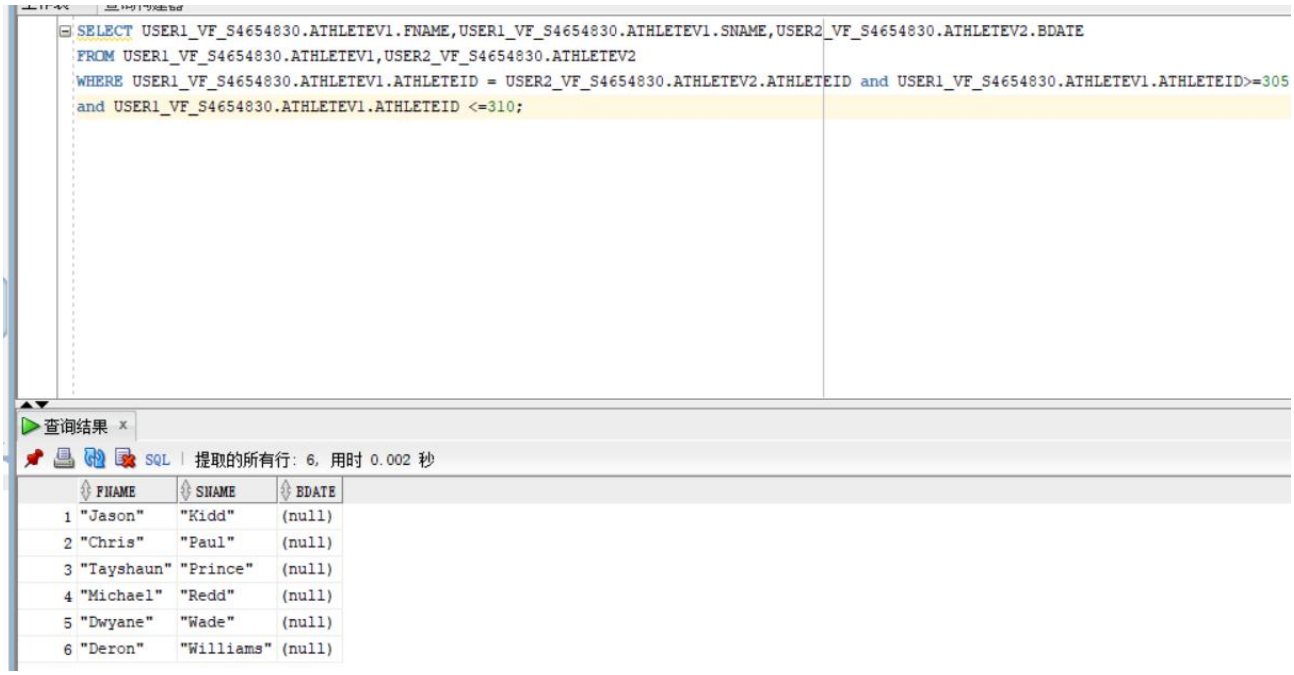


Data load results:



<u>Task 3</u>: Write a SQL query to retrieve the full name and DOB of all the athletes satisfying 305<= AlthleteID<=310. *<u>Include your query and the screenshot of the result in your submission.</u>*

<u>Code and results:</u>



```
SELECT USER1_VF_S4654830.ATHLETEV1.FNAME,USER1_VF_S4654830.ATHLETEV1.SNAME,USER2_VF_S4654830.ATHLETEV2.BDATE
FROM USER1_VF_S4654830.ATHLETEV1,USER2_VF_S4654830.ATHLETEV2
WHERE USER1_VF_S4654830.ATHLETEV1.ATHLETEID = USER2_VF_S4654830.ATHLETEV2.ATHLETEID and USER1_VF_S4654830.ATHLETEV1.ATHLETEID>=305
and USER1_VF_S4654830.ATHLETEV1.ATHLETEID <=310;
```

查询结果

SQL | 提取的所有行：6，用时 0.002 秒

	FNAME	SNAME	BDATE
1	"Jason"	"Kidd"	(null)
2	"Chris"	"Paul"	(null)
3	"Tayshaun"	"Prince"	(null)
4	"Michael"	"Redd"	(null)
5	"Dwyane"	"Wade"	(null)
6	"Deron"	"Williams"	(null)

Task 4: Suppose that we want to retrieve all the information for Australian athletes from the vertical fragments created in Task 2, which can be achieved by the following join query:

Select distinsct(CCODE) **from** USER_S4654830.COUNTRY**;**

1. Inner join :

Step 1:

Search

```
统计信息
----------------------------------------------------------
          1  recursive calls
          0  db block gets
        173  consistent gets
          0  physical reads
          0  redo size
      29930  bytes sent via SQL*Net to client
       1125  bytes received via SQL*Net from client
         49  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
        717  rows processed

SQL>
```

Sql results:

```
SELECT USER1_VF_S4654830.ATHLETEV1.FNAME,USER1_VF_S4654830.ATHLETEV1.SNAME,USER2_VF_S4
from USER2_VF_S4654830.ATHLETEV2,USER1_VF_S4654830.ATHLETEV1,USER_S4654830.COUNTRY
where USER2_VF_S4654830.ATHLETEV2.CCODE = USER_S4654830.COUNTRY.CCODE
AND USER2_VF_S4654830.ATHLETEV2.CCODE = 'AUS'
AND USER2_VF_S4654830.ATHLETEV2.ATHLETEID = USER1_VF_S4654830.ATHLETEV1.ATHLETEID;
```

查询结果

SQL | 已提取 50 行，用时 0.014 秒

	FNAME	SNAME	BDATE	SPORTID	ATHLETEID	CCODE
1	"Ryan"	"Bayley"	(null)	11	2188	AUS
2	"Brad"	"McGee"	(null)	11	2191	AUS
3	"Graeme"	"Brown"	(null)	11	2193	AUS
4	"Brett"	"Lancaster"	(null)	11	2194	AUS
5	"Luke"	"Roberts"	(null)	11	2195	AUS
6	"Sara"	"Carrigan"	(null)	11	2208	AUS
7	"Kate"	"Mactier"	(null)	11	2214	AUS
8	"Shane"	"Kelly"	(null)	11	2227	AUS
9	"Stuart"	"O'Grady"	(null)	11	2228	AUS
10	"Mathew"	"Helm"	(null)	12	2238	AUS
11	"Chantelle"	"Newbery"	(null)	12	2242	AUS
12	"Loudy"	"Tourky"	(null)	12	2244	AUS

Step 2: Perform Inner join

```sql
SELECT USER1_VF_S4654830.ATHLETEV1.ATHLETEID,USER1_VF_S4654830.ATHLETEV1.FNAME,USER1_
FROM USER1_VF_S4654830.ATHLETEV1
INNER JOIN USER2_VF_S4654830.ATHLETEV2
ON USER1_VF_S4654830.ATHLETEV1.ATHLETEID = USER2_VF_S4654830.ATHLETEV2.ATHLETEID
AND USER2_VF_S4654830.ATHLETEV2.CCODE = 'AUS'
ORDER BY USER1_VF_S4654830.ATHLETEV1.SNAME;
```

查询结果

SQL | 已提取 50 行，用时 0.013 秒

	ATHLETEID	FNAME	SNAME	BDATE	SPORTID	CCODE
1	11	"Des"	"Abbott"	1986/10/2 0:00:00	18	AUS
2	12	"Michael"	"Aikman"	(null)	27	AUS
3	13	"Brett"	"Aitken"	1971/1/25 0:00:00	11	AUS
4	2830	"Sandra"	"Allen"	(null)	32	AUS
5	12362	"Judith"	"Amoore"	(null)	38	AUS
6	11618	"Thomas"	"Anderson"	(null)	29	AUS
7	2002	"Craig"	"Anderson"	(null)	5	AUS
8	14532	"David"	"Anderson"	(null)	27	AUS
9	8844	"Scott"	"Anderson"	(null)	29	AUS
10	6071	"Ramon"	"Andersson"	(null)	8	AUS
11	13436	"Janice"	"Andrew"	(null)	33	AUS
12	5135	"Michelle"	"Andrews"	(null)	18	AUS
13	4425	"Kerri"	"Ann Rottharst"	(null)	41	AUS

Step 3

```
统计信息
----------------------------------------------------------
         32  recursive calls
          0  db block gets
        175  consistent gets
          0  physical reads
          0  redo size
      30423  bytes sent via SQL*Net to client
       1125  bytes received via SQL*Net from client
         49  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
        717  rows processed

SQL>
```

Cost calculate : 30423 + 1125 = 31548

2. Semi join

Step 1

search

```
统计信息
----------------------------------------------------------
          1  recursive calls
          0  db block gets
        173  consistent gets
          0  physical reads
          0  redo size
      29930  bytes sent via SQL*Net to client
       1125  bytes received via SQL*Net from client
         49  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
        717  rows processed

SQL>
```

(search code s same as inner join)

Step 2:

Sql code

```sql
SELECT USER1_VF_S4654830.ATHLETEV1.ATHLETEID,
       USER1_VF_S4654830.ATHLETEV1.FNAME,
       USER1_VF_S4654830.ATHLETEV1.SNAME
FROM USER1_VF_S4654830.ATHLETEV1
WHERE ATHLETEID IN (
    SELECT ATHLETEID FROM
USER2_VF_S4654830.ATHLETEV2 WHERE CCODE = 'AUS'
    )
ORDER BY USER1_VF_S4654830.ATHLETEV1.SNAME;
```

Step 3:

Cost screenshot:

```
统计信息
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        165  consistent gets
          0  physical reads
          0  redo size
      26631  bytes sent via SQL*Net to client
       1125  bytes received via SQL*Net from client
         49  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
        717  rows processed

SQL>
```

COST CALCULATE : 26631 + 1125 = 27756

Summary:

The SEMI join cost more less resources than inner join. Because the inner join is a loop to match every rows, and semi join just have one input and than match the information once. So inner join take much more time and have more operation on database than semi join.