

Prac 1: Distributed Databases (5%)

Semester 1, 2021

Introduction

Learning Objectives:

- Learn how to use Oracle DBMS through SQL Plus and SQL Developer. Oracle will be used in both Pracs 1 & 2.
- Get familiar with the basic SQL queries and keywords. Write your SQL queries for data retrieval.
- Simulate horizontal and vertical fragmentation using centralized Oracle. Understand how to update records on a distributed database with data replications.
- Apply semi-join algorithm to simulate data transmission cost reduction over computer networks. Understand why semi-join could be faster sometimes.

Assessment:

Due in week 6 (online submission)

This prac carries 5 marks.

Marking scheme:

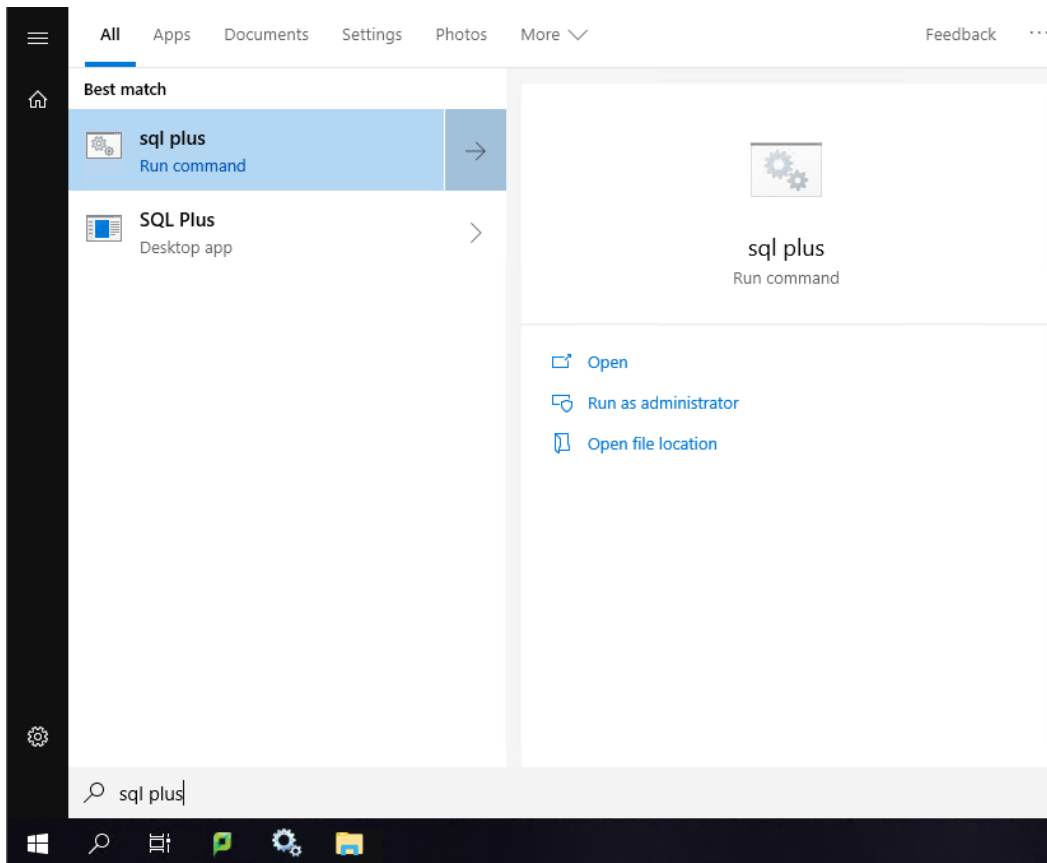
- 2 marks: Receive one mark for completing two of Task 1 questions, full marks for completing all three questions;
- 1 mark: Answer the question of Task 2 correctly, receive 0.5 mark when unnecessary updates or inappropriate explanation appears;
- 1 mark: Finish Task 3;
- 1 mark: Finish Task 4;

Screenshot your results and provide necessary scripts/explanations according to task requirements. Please make sure your screenshots contain your student ID (your student ID will be included in the name of the users you created) as the proof of originality. Put all your content in a word/pdf document or leave scripts in separate files and pack all files into a zip/rar package. Please format your document and code nicely to help tutor's marking process. A poorly formatted document may receive a reduced mark. Submit your work to the Blackboard site by 11:59pm, September 13th. Late submission will receive 2 marks' penalty every 12 hours.

Part 1: Oracle 12c Enterprise Basics

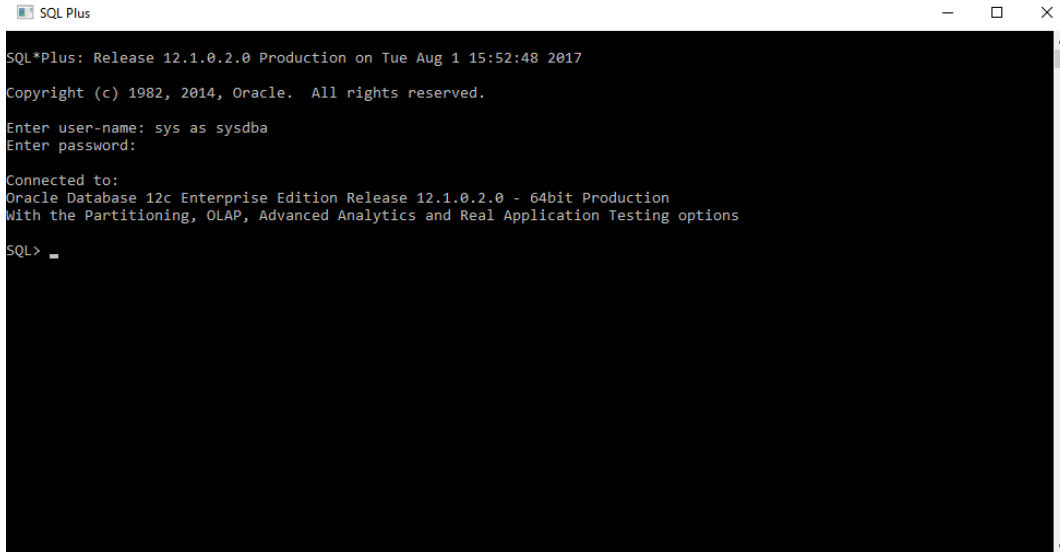
1. Find the Oracle software

From the Windows 10 start menu, you can type “SQL Plus” and “SQL Developer” to search the tools, as shown below. We will use “SQL Plus” to create database users, and use “SQL Developer” to connect as these users and interact with the database.



2. Login and create users

In SQL Plus Command Line window, first log in with username “SYS AS SYSDBA” and password “Password1!”, a successful login should be similar as follows (see troubleshooting below if you cannot log in).



```
SQL*Plus: Release 12.1.0.2.0 Production on Tue Aug 1 15:52:48 2017
Copyright (c) 1982, 2014, Oracle. All rights reserved.
Enter user-name: sys as sysdba
Enter password:
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
SQL> _
```

Execute the commands below to create users. You can copy them, excluding comments (words in green), and right click your mouse in “SQL Plus” to paste.

NOTE: All “S1234567” mentioned in the document should be replaced by your student ID to distinguish your work from others, case insensitive.

```
/*Enable user creation*/
ALTER SESSION SET "_ORACLE_SCRIPT"=TRUE;

/* Create a user named “USER_S1234567” with password “w” */
CREATE USER USER_S1234567 IDENTIFIED BY w ACCOUNT UNLOCK
DEFAULT TABLESPACE "USERS" TEMPORARY TABLESPACE "TEMP"
PROFILE "DEFAULT";

/* Grant DBA privilege to “USER_S1234567” */
GRANT DBA TO USER_S1234567;

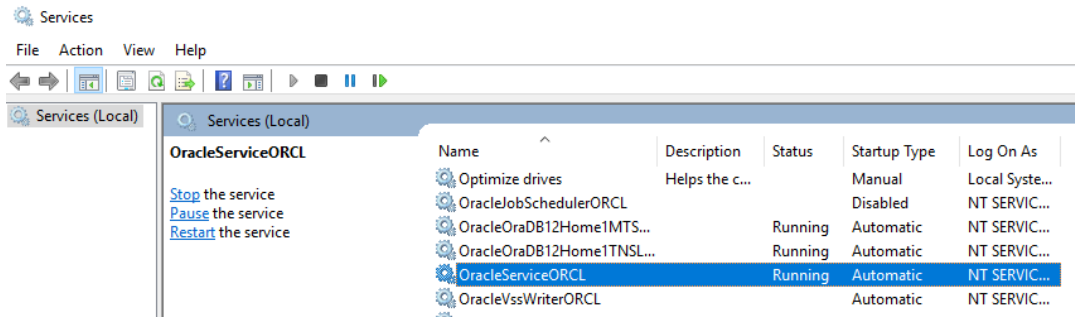
/* Check if “USER_S1234567” has been created */
SELECT USERNAME FROM DBA_USERS;
```

Proceed to step 3 if you complete the above processes successfully. Otherwise, we provide several solutions to the problems you may encounter.

Troubleshooting:

(1) TNS: Protocol adapter error: Oracle services are closed, check the service state:

From the Windows 10 “Start” menu, search for “services”. In Services, check if “OracleOraDB12Home2MTSRecoveryService”, “OracleOraDB12Home2TNSListener” and “OracleServiceORCL” are running. If not, right-click and start them.



(2) Logon denied: Incorrect password, **retry** the password or reset it as follows:

Open C:\app\ntadmin\virtual\product\12.2.0\dbhome_2\network\admin\sqlnet.ora, and change the 8th line to:

SQLNET.AUTHENTICATION_SERVICES= (NONE)

Then open a command **prompt** and type the following (you can simply copy and paste):

```
orapwd
file=C:\app\ntadmin\virtual\product\12.2.0\dbhome_2\database\PWDorcl.ora
password=Password1! force=y
```

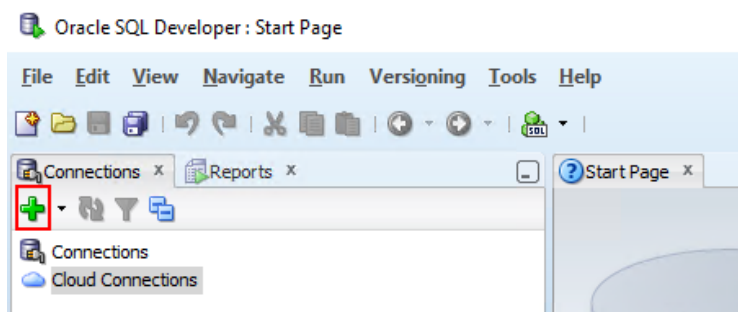
(3) “USER_S1234567” conflicts with another user: When creating user, drop the existing user by running:

```
/* WARNING: this will drop everything under that user */
DROP USER USER_S1234567 CASCADE;
```

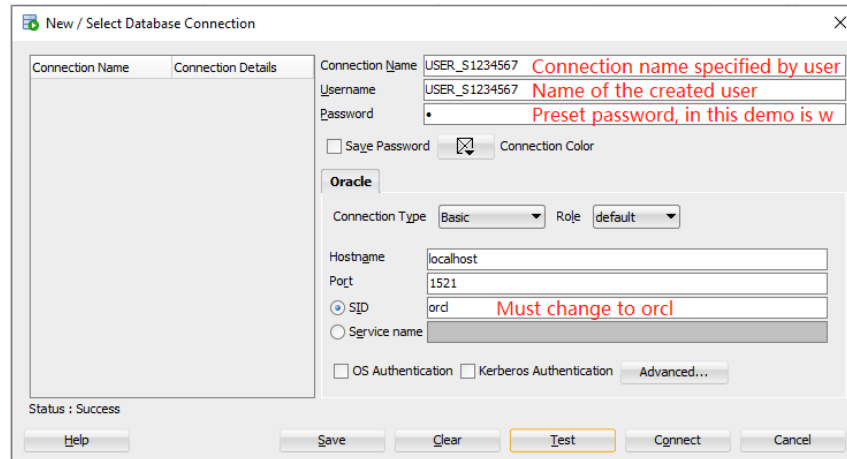
Then redo the **CREATE** and **GRANT** commands.

3. Use Oracle SQL Developer

Open SQL Developer in the start menu. We will use it to connect as the user we just created. Click the green “+” button as shown below.

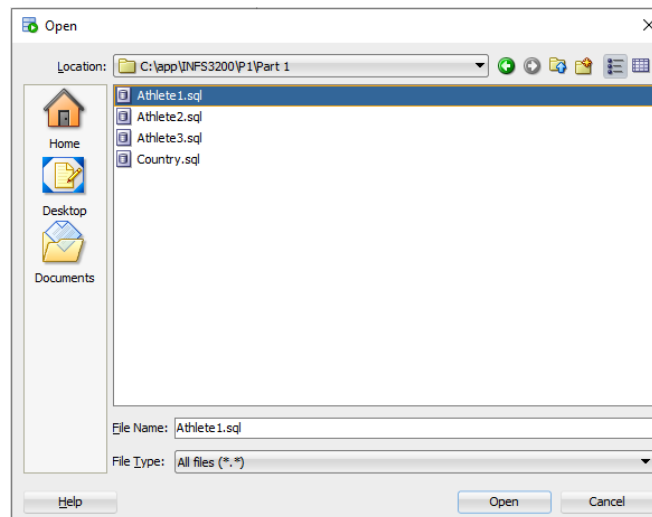


Fill in the connection information in the prompted **dialog** window as shown below. The connection name is specified by the user. Username should be a user already existing in the database. In this example, it is the “USER_S1234567” we just created. Password is the password for that user, namely “w” in this case. You should also change SID to “orcl”. Then press the “Connect” button to connect to the user.



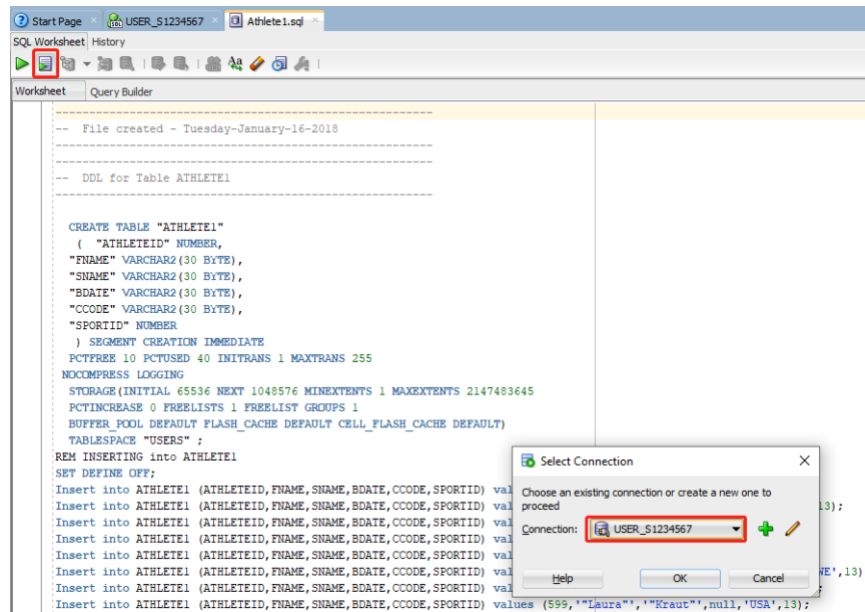
4. Import data to database

Select “Open” command in the “File” menu. Open the SQL scripts we provide for Part 1 as shown below (folder: “...\P1\Part 1”).

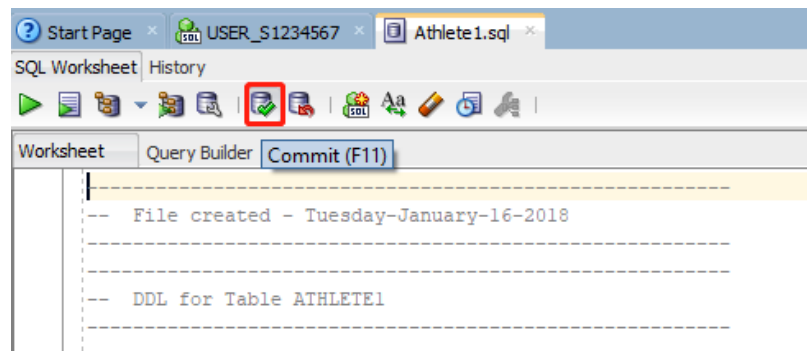


It will show the script in the window. The script contains a table creation command (**CREATE TABLE**) and a list of record insertions (**INSERT INTO**). Click the second button (run all the scripts in the current tab) on its menu bar. A dialog will pop up asking

for connection details. You should choose which connection you want to run the script. In this step, we choose the USER_S1234567.



After running a script, press the sixth button on the menu bar, as shown below, to **commit** the insertion. We perform the same process for all four scripts in the folder to complete the insertion.



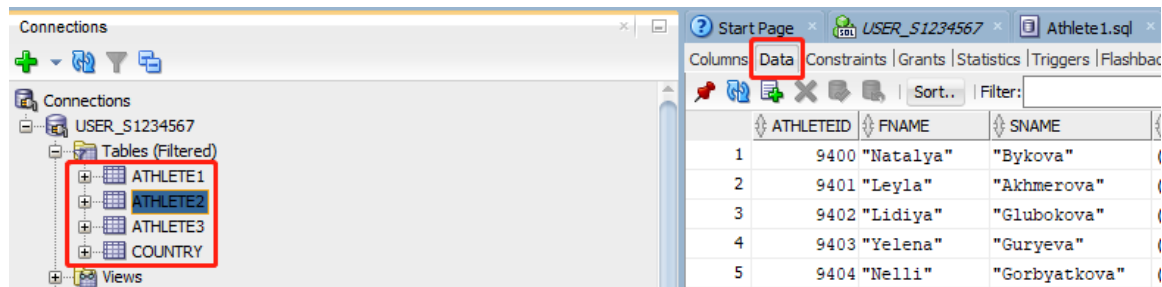
Troubleshooting:

- (1) **Maximum cursors exceed:** Disconnect the current connection by right-clicking on it in the connections panel and reconnect.
- (2) **Unique constraint:** Check the top-right dropdown list and make sure you are in the correct connection. Also check if the data is already imported by following step 5.



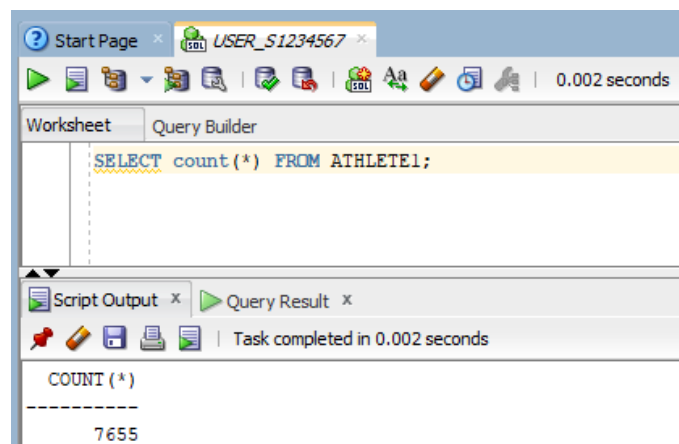
5. View the imported data

In the left panel, expand your connection and find your tables under the Table directory. The basic information will be shown in the right window. Click the second tab “Data” in the right window to find the data you just imported, as shown below.



6. Interact with database using SQL

You can write any SQL query in the corresponding connection window and run it by clicking either the first button (run a single query under current cursor) or the second one (run every script in the current tab). For example, the query below tries to find out how many records we have imported to table “Athlete1”. We will ask you a few similar questions in task 1 which requires you understand the meaning of the table attributes and help you review the basic SQL keywords ([SELECT](#), [WHERE](#), [GROUP BY](#), [JOIN](#), etc.) which are necessary for the rest of the course.



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.
- (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		

- (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.

Part 2: Distributed DB Design

In part 2, we aim to simulate a distributed database using a standalone computer. To do so, we are acting as a global site, which possesses the global conceptual schema and data replication info, and deal with all the incoming queries, then we create multiple user accounts in Oracle 12c (refer to “Part1: Oracle 12c Enterprise Basics” for how to create a user account), each of which represents a distributed local site. The data transferred among relations belonging to different user accounts corresponds to the data transferred over computer network among different sites. Given a global conceptual schema, perform the following tasks:

We provide you an Athlete table:

Athlete[AthleteID, FName, LName, DOB, CountryCode, SportID]

There are 24,591 records with AthleteID ranging from 1 to 24,591.

1. Horizontal fragmentation

There are three types of replication strategies: Full Replication, Partial Replication and No Replication. In this task, you are asked to simulate these three strategies on a distributed database which contains three local sites (USER1, 2 and 3). Each strategy requires 3 users for simulation, that is to say, you need to create 9 users in total for Task 2. After creating the users, you are asked to run the scripts in the respective folders to perform different data replications.

Job 1 - Full Replication

The data is split into three fragments:

- Athlete1: $1 \leq \text{AthleteID} < 7656$
- Athlete2: $7657 \leq \text{AthleteID} < 17318$
- Athlete3: $17319 \leq \text{AthleteID} \leq 24591$

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_S1234567
- 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 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

In order to load the above fragments into site USER1_HF_PA_S1234567, connect to user “USER1_HF_PA_S1234567” in SQL Developer and run all script files in folder “...\P1\Part 2\HF\HF-Partial\USER1_HF_PA\”. Repeat the same process for other sites.

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

In order to load the above fragments into site USER1_HF_NO_S1234567, connect to user “USER1_HF_NO_S1234567” in SQL Developer and run all script files in folder “...\P1\Part 2\HF\HF-No\USER1_HF_NO\”. Repeat the same process for other sites.

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”.

Put your SQL queries/transactions, your result screenshots and the explanation of the differences of update operations between three replication strategies in your submission.

2. Vertical Fragmentation

Vertical Fragmentation:

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

Create two users to simulate two sites, and load the data from folder “...\P1\Part 2\VF”

- USER1_VF_S1234567
- USER2_VF_S1234567

Task 3: Write a SQL query to retrieve the full name and DOB of all the athletes satisfying $305 \leq \text{AthleteID} \leq 310$. Include your query and the screenshot of the result in your submission.

Part 3: Distributed Query Processing

In part 3, we still simulate a distributed database using this centralised Oracle database. However, the distributed sites are now organised in a peer-to-peer architecture, which means the queries can be issued at any of the local sites and receive answer from it. In this part, you are asked to perform inner join queries in such distributed environment efficiently, which means you should find the optimal execution plan for the join queries, as mentioned in Lecture 3, the optimal execution plan refers to the one with minimum data transmission cost. Therefore, to help you trace the data transmission cost, we provide the following statistical tool:

Open **SQL Plus** and login as “SYS AS SYSDBA” (applicable to other users you created). Type in the following commands to turn on query statistics.

```
SQL> SET TIMING ON; /* enable timing */
SQL> SET AUTOTRACE ON STATISTICS; /* enable statistics */
```

We use a simple query as an example. Here is the statistical result of the following query:

```
select a.AthleteID, a.CCODE, b.CONTINENT
from "USER2_VF_S1234567"."ATHLETE_V2" a,
"USER_S1234567"."COUNTRY" b
where a.CCODE=b.CCODE;
```

```

24585 rows selected.
Elapsed: 00:00:07.82
Statistics
-----
      5 recursive calls
      0 db block gets
    1714 consistent gets
      0 physical reads
      0 redo size
    604782 bytes sent via SQL*Net to client
    18626 bytes received via SQL*Net from client
     1640 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
    24585 rows processed

```

Note that the statistics may vary among multiple runs except those marked in red, which are our main focuses. The descriptions of the statistics are shown in the appendix. In particular, the “bytes sent via SQL*Net to client” refers to the size of the query results sent from the database to user, it is irrelevant to data transmission cost. However, this statistical tool can be used to estimate the data transmission cost indirectly. For example, if we issue this query at site USER_S1234567 and run it with the semi-join execution plan, the whole query can be divided into three steps:

- (1) **Send** the projection of “CCODE” from site USER to USER2
- (2) Perform semi-join on site USER2 then **send** the result back to USER
- (3) Perform the final join locally and send the final results to user.

Therefore, the total transmission cost should be the size of step (1) projection results + step (2) results (final result is not included as you have to return the same result to user regardless of what execution plan you take). Specifically, to calculate the cost, you have to perform the following queries to obtain the size of the intermediate results:

Step One – Get the projection of “CCODE” from USER.COUNTRY

select distinct(CCODE) from "USER_S1234567"."COUNTRY";

```

136 rows selected.
Elapsed: 00:00:00.03
Statistics
-----
      1 recursive calls
      0 db block gets
      7 consistent gets
      0 physical reads
      0 redo size
     3017 bytes sent via SQL*Net to client
     707 bytes received via SQL*Net from client
      11 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
     136 rows processed

```

Step Two – Perform the semi join on USER2

select a.AthleteID, a.CCODE **from** "USER2_VF_S1234567"."ATHLETE_V2" a **where**
a.CCODE **in** (**select distinct**(CCODE) **from** "USER_S1234567"."COUNTRY");

```
24585 rows selected.
Elapsed: 00:00:04.95
Statistics
-----
      1 recursive calls
      0 db block gets
    1711 consistent gets
      0 physical reads
      0 redo size
  589666 bytes sent via SQL*Net to client
    18626 bytes received via SQL*Net from client
    1640 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
    24585 rows processed
```

Step Three – Perform the final join on USER

select a.AthleteID, a.CCode, b.CONTINENT
from "USER_S1234567"."COUNTRY" b,
(**select** a.AthleteID, a.CCODE **from** "USER2_VF_S1234567"."ATHLETE_V2" a **where**
a.CCODE **in** (**select distinct**(CCODE) **from** "USER_S1234567"."COUNTRY")) a
where a.CCODE= b.CCODE;

```
24585 rows selected.
Elapsed: 00:00:05.09
Statistics
-----
      4 recursive calls
      0 db block gets
    1714 consistent gets
      0 physical reads
      0 redo size
  604782 bytes sent via SQL*Net to client
    18626 bytes received via SQL*Net from client
    1640 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
    24585 rows processed
```

Therefore, the transmission cost of the semi-join plan is 3017(step 1) + 589666(step 2) = 592683. The example shows that by decomposing the query into a step-by-step plan, we can track the data transmission cost by aggregating the sizes of their intermediate results.

Additionally, we provide you with several tips for this task:

Tip 1: For the same query, the size of the final results should always be identical regardless of what execution plan you take. Like in the above example, the final result size is always 604782.

Tip 2: Despite different execution plans, the running time of the query should always be similar, the main reason is that we are using a centralised database to simulate distributed environment, but it is still a centralised DB technically. Therefore, the Oracle will optimise every query and, no matter if it is written as an inner join or semi-join, it will be executed in the same plan as long as those queries are semantically equivalent.

Tip 3: Same as tip 2, although we regard each user account as a distributed site, you will not see any difference if you run the same query using different users as they are all local users. However, in a real distributed database, query issued at different sites will result in completely different execution plans. For example, if your query is issued at site 1, it is better to allocate your final join at site 1 so that you can directly return the result to client (user) once completed.

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: (semi-join doesn't make sense)

```
select b.AthleteID, b.FName, b.SName, c.BDate, c.CCode, c.SportID
from "USER1_VF_S1234567"."ATHLETE_V1" b,
"USER2_VF_S1234567"."ATHLETE_V2" c
where b.AthleteID= c.AthleteID and c.CCODE='AUS';
```

```
717 rows selected.
Elapsed: 00:00:00.51
Statistics
-----
      5 recursive calls
       0 db block gets
     226 consistent gets
       0 physical reads
       0 redo size
  29922 bytes sent via SQL*Net to client
   1125 bytes received via SQL*Net from client
      49 SQL*Net roundtrips to/from client
       0 sorts (memory)
       0 sorts (disk)
     717 rows processed
```

If the join query is issued at site “USER1”, write a semi-join and an inner-join execution plans in such vertically fragmented distributed database, respectively. Follow the above example to calculate their respective data transmission cost and decide which plan is better. Include step-by-step queries, cost calculations and your choice in your submission, supported by the screenshots of the query statistics.

```

4642 rows selected.

Elapsed: 00:00:02.85

Statistics
-----
      12  recursive calls
       0  db block gets
    26691  consistent gets
       0  physical reads
       0  redo size
   192749  bytes sent via SQL*Net to client
    4007  bytes received via SQL*Net from client
     311  SQL*Net roundtrips to/from client
        1  sorts (memory)
       0  sorts (disk)
    4642  rows processed

```

Appendix. Statistics Description

Database Statistic Name	Description
recursive calls	Number of recursive calls generated at both the user and system level. Oracle maintains tables used for internal processing. When Oracle needs to make a change to these tables, it internally generates an internal SQL statement, which in turn generates a recursive call.
db block gets	Number of times a CURRENT block was requested.
consistent gets	Number of times a consistent read was requested for a block
physical reads	Total number of data blocks read from disk. This number equals the value of "physical reads direct" plus all reads into buffer cache.
redo size	Total amount of redo generated in bytes
bytes sent via SQL*Net to client	Total number of bytes sent from Oracle database to the client (user) through network, which includes the message initiation cost and the cost of sending query results.
bytes received via SQL*Net from client	Total number of bytes received from the client over Oracle Net.
SQL*Net roundtrips to/from client	Total number of Oracle Net messages sent to and received from the client
sorts (memory)	Number of sort operations that were performed completely in memory and did not require any disk writes
sorts (disk)	Number of sort operations that required at least one disk write
rows processed	Number of rows processed during the operation