CS-450, CS-550, INFS-614: How to Use Oracle

1. Preliminaries

Oracle is installed on the VSE Oracle Server `apollo.vse.gmu.edu`, and is accessible from the VSE server `zeus.vse.gmu.edu`. To access Oracle you need 2 computer accounts:

1. VSE UNIX account: This account is created automatically for all students who are enrolled in any course offered by an VSE department. This account can be used to log into zeus.vse.gmu.edu using SSH. For issues regarding computer accounts see: http://labs.ite.gmu.edu/reference/faq_iteaccount.htm.
2. Oracle account: To become a user of the Oracle Database Management System you must create an Oracle account. For problems regarding your Oracle account you may contact Oracle administrator.

Note that a firewall prevents you from connecting directly to `zeus.vse.gmu.edu` from off campus. In such cases you should first connect to the GMU VPN (http://vpn.gmu.edu/). Then you should be able to use the `ssh` command to connect to `zeus.vse.gmu.edu`.

The VSE server `zeus.vse.gmu.edu` runs the Unix operating system, and you will need basic familiarity with this operating system. Some of the basic commands that you will want to learn are `ls`, `pwd`, `mkdir`, `cd`, `cp`, `rm`, `cat`, `more`, and `lp`, as well as one of the text editors (e.g., `emacs`, `vi`, `ed` or `pico`). If you are not familiar with any of these editors, you should consider using pico. Type `pico` and press return. A simple menu will appear. Once you have logged to the system, you may use the command `man` to display the manual pages for any of these commands. For example, `man man` will display the manual page of the command `man`.

2. SQL*Plus

1. **Start SQL*Plus.** At the Unix prompt, type the command `sqlplus`, and then enter your Oracle user-name and password to login to Oracle. You will get the Oracle prompt `SQL>` and you can now issue SQL commands. If you don't get the `SQL>` prompt, send email to the Oracle Database Administrator.

2. **Quit SQL*Plus.** The SQL commands `quit` or `exit` will log you out of Oracle and return you to the Unix prompt. If you are done, you should now log out of `zeus.ite.gmu.edu` as well.

3. **Help.** At the `SQL>` prompt, you can type the command `help` to get on-line help messages. Type

   ```
   SQL> help index
   ```

   to get a list of help topics. The topics concern the `sqlplus` tool, not the SQL language.

4. **Pause output.** To instruct Oracle to pause after each page when text is displayed, assuming the size of your window is 20 lines, at the `SQL>` prompt, type

   ```
   SQL> set pagesize 20

   SQL> set pause on
   ```

   Press return to continue after a pause, and Control-C to stop the display of text.

5. **Print.** Use the command

   ```
   SQL> spool filename
   ```

   to log the entire session (your input and Oracle's output) in a Unix file called *file-name*`.lst`. Use `spool off` to turn off the logging. Later, at the Unix prompt, you can print this file using the commmad

   ```
   UNIX> enscript -P pfp file-name.lst
   ```

   If you are working remotely, you can move this file to your local computer (using, for example, `ftp` or `sftp`) and print it there. To save paper, you may use the Unix command

   ```
   UNIX> mpage -2 file-name.lst > file-name.ps
   ```

   to convert your spool file to a Postscript file in which two pages are printed on a single sheet of paper, and then print with

   ```
   UNIX> lp file-name.ps
   ```

   If you substitute "-2" with "-4" in the above command, you will print four pages per sheet.

6. **Demo database.** To build the demo database type `demobld` at the Unix prompt. You may then start SQL*Plus and attempt various SQL commands against this database. To remove the demo tables, type `demodrop` at the Unix

prompt.

7. **Edit your input.** Your most recent input to SQL*Plus is saved in a buffer, which you may edit and resubmit. To edit SQL*Plus buffers, use the command

   ```
   SQL> edit
   ```

   This command launches a text editor with the contents of the buffer. When you finish your editing, save, exit the editor, and resubmit using the command

   ```
   SQL> run
   ```

   The commands `edit` and `run` may be abbreviated `ed` and `r`, respectively.
   The default Unix editor is `ed`. To change this default, set the EDITOR environment variable. For example, if you want pico to be your editor, at the Unix prompt type

   ```
   UNIX> EDITOR=pico; export EDITOR
   ```

   before you start SQL*Plus. (This assumes that you are using bash as your command interpreter.) You may also include the above line in your Unix configuration file `.bash_profile` so that it is executed automatically every time you login.

8. **Customize the SQL*Plus environment.** To customize your SQL*Plus environment, create a Unix file called `login.sql`. The commands in this file will be executed every time you start SQL*Plus. For example, you could include in this file the commands for pausing the output described earlier.

9. **Examples.** Below are examples of three popular operations. In each case, the commands are entered at the `SQL>` prompt.
   - To create a relation, use the `create table` command. For example:

     ```
     create table DEPT
     ( DEPTNO number(2),
       DNAME varchar2(14),
       LOC varchar2(13) );
     ```

   - To insert rows into a relation, use the `insert` command. For example,

     ```
     insert into DEPT values (20,'research','Dallas');
     ```

   - To display the rows of a relation, use the command `select`. For example,

     ```
     select * from DEPT;
     ```

     You may use the spool command to turn on the spooling before issuing the `select` command, and then print the spool file as described earlier.

10. **Scripts.** When a sequence of SQL*Plus commands has to be repeated, script files may be used. Using a text editor, create a file of SQL*Plus commands. The file name must end with `.sql`, for example, `myscript.sql`. Then, at the Unix prompt type

    ```
    UNIX> sqlplus @myscript
    ```

    Or, at the `SQL` prompt type

    ```
    SQL> @myscript
    ```

    As an example, consider this script file called `table_dept.sql`:

    ```
    drop table EMP;
    create table EMP
    ( EMPNO number(4) not null,
      ENAME varchar2(10),
      JOB varchar2(9),
      MGR number(4),
      HIREDATE date,
      SAL number(7,2),
      COMM number(7,2),
      DEPTNO number(2) );
    insert into EMP values (7369,'Smith','clerk',7902,'17-DEC-80',800,NULL,20);
    insert into EMP values (7499,'Allen','salesman',7698,'20-FEB-81',1600,300,30);
    ```

    The commands in this script file (1) delete the present `EMP` table, (2) create a new `EMP` table with the specified columns, and (3) insert two rows into the `EMP` table. To execute these commands, type

    ```
    UNIX> sqlplus @table_dept
    ```

    at the Unix prompt, or

    ```
    SQL> @table_dept
    ```

at the `SQL>` prompt. This method makes it easier to initialize a database with a large number of rows. And if definitions need to be changed, it may be easier to edit the script file and re-execute it. Unless you include the SQL*Plus command `quit` at the end of the script, when the script is done, you will be at the SQL prompt.

11. **Finding out what's in your database.**
    - To find out the tables in your database:

      ```
      select TABLE_NAME
      from USER_TABLES;
      ```

    - To find out the columns in a table:

      ```
      describe table-name;
      ```

    - To find out the rows in a table:

      ```
      select *
      from tablename;
      ```

12. **Circular constraints.** Suppose we want to define two tables: R(A,B) and S(C,D), in which A and C are primary keys, B is a foreign key that references S.C, and D is a foreign key that references R.A. There are two different problems.

    First, it is impossible to include in the definition of R(A,B) the foreign key reference to S, beacuase S has not been created yet. Similarly, if we attempt to create S(C,D) first, it cannot include the foreign key reference to R, because R has not been created yet. The solution is to create the first table without the foreign key constraint, and add it later, as follows:

    ```
    create table R (
    A number primary key,
    B number
    );

    create table S (
    C number primary key,
    D number references R deferrable
    );

    alter table R
    add constraint BREFC foreign key (B) references S deferrable
    ;
    ```

    Note three important things:
    1. We could, of course, choose to create *both* tables without the foreign keys and add both later.
    2. The `alter table` command must include a *name* for the new constraint (in this case, we chose BREFC).
    3. Both foreign key constraints were defined as *deferrable*.

    Now that the tables with the circular constraints have been created, there is a second problem: how to insert rows. If we attempt to insert a row such as (1,10) into R, Oracle will complain about a foreign key violation because the table S should already have the value 10 in column C. But if we address this by inserting first a row such as (10,1) into S, Oracle will complain about a foreign key violation because the table R should already have the value 1 in column A.

    This is why we defined both foreign key constraints as *deferrable*. Normally constraints are checked immediately upon each update. However, if a constraint has been defined as *deferrable*, we may request that constraint checking will be deferred. This is done with the command

    ```
    set constraint all deferred;
    ```

    Now we can add rows. For example:

    ```
    insert into R values (1, 10);
    insert into R values (2, 20);
    insert into S values (10, 1);
    insert into S values (20, 2);
    ```

    When we are done, we should validate the content of the database by activating the constraints. This is done with the command

    ```
    set constraint all immediate;
    ```

    In this case the checking will succeed, and at the end of the transaction (when the SQL*Plus session ends or when we issue `commit`) the updates will be saved. However, if we also added

    ```
    insert into S values (30, 3);
    ```

then the checking of constraints will fail. The table S may show this additional row during this session, but when the session ends (or when we issue `commit`), the transaction will be *rolled back*, and all its updates ignored!

The entire sequence of commands is repeated here:

```
create table R (
A number primary key,
B number
);
create table S (
C number primary key,
D number references R deferrable
);
alter table R
add constraint BREFC foreign key (B) references S deferrable
;
set constraint all deferred;
insert into R values (1, 10);
insert into R values (2, 20);
insert into S values (10, 1);
insert into S values (20, 2);
set constraint all immediate;
```

On a final note, when you use a script to create your database, and you include `drop table` commands at its beginning, and there are foreign key constraints, then you must drop the tables in the correct order! For example if table R has a foreign key that references table S, then you must drop table R first! However, if tables R and S are "locked" in circular constraints, then you must use the `cascade constraints` option of the `drop table`command!

3. C-Embedded SQL

Write your C-embedded SQL programs using a text editor. The program file name must end with `.pc`. To pre-compile an embedded C/SQL program `myprog.pc`, use the command

```
UNIX> occ iname=myprog.pc oname=myprog.c
```

which will generate a C program by the name `myprog.c`. To compile this program, use the command:

```
UNIX> cc -L /usr/local/oracle/OraHome1/lib -l clntsh myprog.c
```

This will generate an executable file by the name `a.out`. Finally, to execute your program, use the command

```
UNIX> ./a.out
```

Note that syntax errors in your Pro*C program may result in a confusing "Segmentation error" message.

An example C-embedded SQL program

```
/*************************************************************
This program accesses relations created by the program "demobld".
It connects to ORACLE, declares and opens a cursor, fetches the names
and salaries of all employees, displays the results, then closes the
cursor.
*************************************************************/

#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;

    VARCHAR    username[20];    /* necessary structures for Oracle login */
    VARCHAR    password[20];    /* do not change these lines */
    VARCHAR    empname[11];     /* structures specific to this program */
    float      salary;          /* note that empname has an extra */
                                /* position for string terminator */
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca;         /* include SQL "communication area" for */
                                /* reporting the results of SQL execution */
                                /* after each statement */

void sqlerror();                /* handle unrecoverable errors */

main()
{
  /* Connect to ORACLE. */

  EXEC SQL WHENEVER SQLERROR DO sqlerror();  /* upon error, goto this */
                                             /* user-defined function */
                                             /* (implicit error-handling) */

  EXEC SQL CONNECT :"john" IDENTIFIED BY :"xyz"; /* substitute your name and password */
```

```
                                        /* for "john" and "xyz" */

  printf("\nConnected to ORACLE!\n");

  /* begin work */

  EXEC SQL DECLARE employee CURSOR FOR  /* declare a query and a cursor */
  SELECT ename, sal
  FROM emp;

  EXEC SQL OPEN employee;               /* do the query (creates temp table) */

  printf("\nEmployee_Name      Salary\n");
  printf("-------------      ------ \n");

  /* Loop, fetching all employees and their salaries */

  for ( ; ; )
  {
    EXEC SQL WHENEVER NOT FOUND DO break;     /* exit loop when no more rows */

    EXEC SQL FETCH employee INTO :empname, :salary; /* fetch a row (2 values)*/

    /* The Pro*C declaration "VARCHAR empname" created
       a structure with two fields:
       empname.arr holds the string
       empname.len holds the length of the string
       the next statement terminates the string with a null byte */

       empname.arr[empname.len] = '\0';
       printf("%-17s%9.2f%\n", empname.arr, salary);
  }

EXEC SQL CLOSE employee;                /* delete temporary relation */

printf("\nHave a good day!\n\n");
                                        /* transaction succeeded */
EXEC SQL COMMIT WORK RELEASE;           /* commit the changes, if any, */
                                        /* and release all locks */
exit(0);
}

void sqlerror()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;

  printf("\nORACLE error detected:\n");
  printf("\n% .70s \n", sqlca.sqlerrm.sqlerrmc);
                                        /* transaction failed */
  EXEC SQL ROLLBACK WORK RELEASE;       /* undo the changes, if any, */
                                        /* and release all locks */
  exit(1);
}
```

4. Java-SQL

The JDBC package: Java provides relational database access via the package *java.sql* (this library implements JDBC, the Java DataBase Connectivity subsystem). Your Java program should include the line

```
import java.sql.*;
```

Oracle drivers: To access Oracle, you need to load Oracle-specific drivers. Your Java program should include the line

```
String driverName = "oracle.jdbc.driver.OracleDriver";
Class.forName(driverName);
```

Connecting to your database: To connect your Java program to your Oracle database, your program should include the line

```
String url = "jdbc:oracle:thin:@apollo.ite.gmu.edu:1521:ite10g";
Connection conn = DriverManager.getConnection(url,"username","password");
```

`conn` is a connection object that points to your database. For instance, for a student who was assigned the username "john" and the password "xyz" to connect to the oracle database, the statement would be

```
Connection conn = DriverManager.getConnection(url,"john","xyz");
```

Executing SQL statements:

- To execute SQL statements, you must first issue

```
Statement stmt = conn.createStatement();
```

`stmt` is a statement object that holds your future statements.

- The `executeUpdate` command is used for SQL statements that return no results (e.g., create or drop tables, insert or delete rows). For example, the command

  ```
  stmt.executeUpdate("CREATE TABLE MYTABLE (A VARCHAR[25], B INTEGER)");
  ```

  creates a new table called `MYTABLE` with columns `A` and `B`.

- The `executeQuery` command is used for SQL queries. For example, the command

  ```
  ResultSet myresults = stmt.executeQuery("SELECT * FROM MYTABLE");
  ```

  retrieves the content of the table `MYTABLE`.

Your Java-SQL program file name should end with `.java`. To compile a program called `myprog.java`, issue

```
UNIX> javac MyProg.java
```

This will create a file called `MyProg.class`. To execute your program, issue

```
UNIX> java MyProg
```

An example Java-SQL program

```
/*********************************************************************
This program accesses relations created by the program "demobld".
It connects to ORACLE, fetches the names and salaries of all employees,
and displays the results.
*********************************************************************/

import java.sql.*;  //Import the java SQL library

class MyProg     //Create a new class to encapsulate the program
{

 public static void SQLError (Exception e)   //Our function for handling SQL errors
 {
        System.out.println("ORACLE error detected:");
        e.printStackTrace();
 }

public static void main (String args[])  //The main function

{

 try {                                        //Keep an eye open for errors

       String driverName = "oracle.jdbc.driver.OracleDriver";
       Class.forName(driverName);

       System.out.println("Connecting   to Oracle...");

       String url = "jdbc:oracle:thin:@apollo.ite.gmu.edu:1521:ite10g";
       Connection conn = DriverManager.getConnection(url,"john","xyz");

       System.out.println("Connected!");

       Statement stmt = conn.createStatement();   //Create a new statement

       //Now we execute our query and store the results in the myresults object:
       ResultSet myresults = stmt.executeQuery("SELECT ename, sal FROM emp");

       System.out.println("Employee_Name\tSalary");
       System.out.println("-------------\t------"); //Print a header

       while (myresults.next()) //pass to the next row and loop until the last
        {
          System.out.println(myresults.getString("ename") + "\t\t" +  myresults.getString("sal")); //Print the current row
        }

        conn.close();  // Close our connection.

      }
       catch (Exception e) {SQLError(e);} //if any error occurred in the try..catch block, call the SQLError function

}
}
```

Last updated: Friday, 19 October 2012, 3:00pm
Amihai Motro