Oracle SQL Reference manual:

http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96540/toc.htm

Go to SELECT and Subquery part of the reference manual for details.

PART I:
(20 minutes)
In this part of the lab you will do the following:

- Add three additional tables, namely CourseOffering, taken and CourseDescription. Note that these tables and last week's tables follow from our EER diagram of lab 4 except that 1:N instead of N:M cardinality constraints have been assumed for the offer (CourseDescription-Department) relationship type.

- Load data into these three tables.

- Create a foreign key reference graph.

1. **Add three additional tables**: Copy the file schema.sql from the lab web page and run it in sqlPlus:
   
   **Start schema.sql**

2. **Load data into the three tables as follows**: 
   
   Go to **Ct1 and Dat** of the lab web page and copy all the files into the same directory you created last week.

   Loader commands are available in the **cmd.txt** file. Run each of the three commands separately to load the three tables. Note that these commands are to be run in Unix.

3. **Create a foreign key reference graph**:

   1
(a) Foreign key reference graph is a directed graph where nodes represent the referencing and the referenced tables and a directed edge represents the foreign key reference from the referencing table to the referenced table. An edge is labeled with the foreign key.

(b) A foreign key reference graph, for the relation schemas of last week’s lab, is available at Foreign Key Graph of the lab web page.

(c) Make a hard copy of the graph:
Note that this graph includes foreign key references for last week’s relation schemas only.
Include in this graph foreign key references for the three tables you have created above. This graph may provide a good reference for designing navigational queries that you will design and run next. This type of graphs are more helpful in designing queries when the database consists of a large number of tables.

PART II:  
(80 Minutes)
In this part of the lab you will design and run sql statements involving subqueries, correlated subqueries, IN, EXISTS, GROUP BY, LIKE, OUTER JOINS and M-way join.

1. SUBQUERIES (Nested Queries):

(a) Give real world meaning of the following, and then run it to verify your answer.

```sql
Select DeptName
From Department
Where DeptId IN (Select DeptId
   From DOffers X, DegreeProgram Y
   Where X.ProgId=Y.ProgId and
   ProgType='BS');
```

Note:
The above subquery corresponds to {‘CSE’, ‘EE’, ‘ME’} for
the given database. Therefore, the Where clause of the outer Select for the given database means the following:

Where DeptId ε {'CSE','EE', 'ME'}

(b) i. Give an SQL statement for the following English language query. You must use a subquery to implement it. Run the sql statement to verify your answer.
Note the tuples in table doffers.
Give Names of those Departments that offer BS and PhD programs.
ii. Write the where clause of the outer Select in terms of a set of values for this database, similar to the one in the Note above.

2. CORRELATED SUBQUERIES:

(a) Consider the following SQL statement:
Give real world meaning of the following sql statement and then run it to verify your answer.

Select Sid as StudentId, Sname StudentName
From Student X
Where 3.5<(Select AVG(Grade)
    From taken A, CourseOffering B
    Where X.Sid=A.Sid and A.SeqId=B.SeqId and
    B.Cno='CSE480');

The Where clause of the outer Select above for the given database means the following. Complete the following statements by filling in the values for ”?”:
For the first tuple of Student table:
Where 3.5 <?
For the second tuple of Student table:
Where 3.5 <?
For the third tuple of Student table:
Where 3.5 <?
and so on.
Now give real world meaning of the above sql statement and then run it to verify your answer.
(b) Give an SQL statement using correlated subqueries for the following English language query, and then run it to verify your answer.
Get ids and names of those students who are taking CSE 480 and have average GPA greater than 3.5.

3. Like:

(a) Give real world meaning of the following sql statement, and then run it to verify your answer.

```sql
Select DeptId as DepartmentId, Deptname as DepartmentName
From Department X
Where DeptId IN (Select DeptId
    From DDoffers Y, DegreeProgram Z
    Where X.DeptId=Y.DeptId and
    Y.ProgId=Z.ProgId and
    Z.ProgramName Like '%Computer%'
    and Z.ProgType='BS');
```

(b) Give an SQL statement using `Like` for the following English language query, and then run it to verify your answer.
Get Department Names and names of Department Chairs for those departments offering BS degree and program name containing 'Engineering'.

4. EXISTS and NOT EXISTS:

(a) Give real world meaning of the following sql statement and then run it to verify your answer.

```sql
Select Sname
From Student X
Where Not Exists (Select Sid
    From taken Y, CourseOffering Z
    Where X.Sid = Y.Sid and
    Y.SeqId = Z.SeqId and
    Z.Cno = 'CSE480');
```

(b) Give an SQL statement using EXISTS for the following English language query, and then run it to verify your answer.
Get ids and names of all those students who are in CSE department and taking CSE 480.

5. GROUP BY HAVING:

(a) Give real world meaning of the following sql statement, and then run it to verify your answer.

```sql
Select X.SeqId, Y.Cno, count(*) as NoStudents, AVG(Grade) as AverageGrade
From taken X, CourseOffering Y
Where X.SeqId=Y.SeqId
Group By X.SeqId, Y.Cno
Having Count(*) >= 2
Order by Cno ASC;
```

(b) Give an SQL statement using Group By Having clause for the following English language query, and then run it to verify your answer.

Get instructor id, instructor name, course seq id, course no and average grades for all courses.

6. OUTER JOIN:

(a) For this query you will first insert a few tuples into the joining tables that will not participate in the join.

Go into SqlPlus window and insert the following tuple into the Faculty table.

```
Insert into Faculty(FacName, FacSSNo) values ('JohnDoe', '0005599999');
```

Insert the following tuple into the Department table:

```
Insert into Department(DeptId,DeptName) values('Math', 'Mathematics Department');
```

i. Display the faculty table and the Department table by using select * from table.

ii. Write the resulting tables you expect from the following two outer joins.
iii. Now you run the following sql statement to verify your answer.

**Run the SQL statement to verify your answer.**

```sql
Select FacName, FacSSno, DeptName
From Faculty X, Department Y
Where X.Worksfor (+)=Y.DeptId
```

```sql
Select FacName, FacSSno, DeptName
From Faculty X, Department Y
Where X.Worksfor=Y.DeptId (+)
```

(b) Now you run the above sql statements to verify your answer.

7. Queries involving Long Join Paths (in the Foreign key Reference Graph):

Give real world meaning of the following sql statement. Run it to verify your answer.

```sql
Select DeptName, FacName
From Department D, Faculty F, CourseOffering CO,
CourseDescription CD
Where D.DeptId=F.Worksfor and F.FacSSNo=CO.Instructor and
CO.Cno=CD.Cno and CD.Ofers=D.DeptId;
```

8. Give sql statements for the following three English language queries. These queries involve only equijoins just like the sql statement of problem 7. For each of these queries you first identify the correct join path in the foreign key reference graph. After you have identified the join path for a query, you design the sql statements by using the sequence of cascaded equijoins like in the Where clause of the above sql statement.

(a) Get name of each student and her/his department name.
(b) Get name of each student and the names of those department he/she is taking at least one course from.
(c) Get student name and his/her department name if the student is taking at least one course offered by his/her department.