Chapter 11
SUBQUERIES

- What is a subquery?
- Multiple Subqueries
- Nested subquery
- Using subquery in DML commands
- Correlated subquery
- EXISTS, NOT EXISTS, ANY and ALL operators

What Is A Subquery

A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.

In this chapter, we will see how subqueries are used to retrieve the required data and also how to use subqueries in DML and DDL commands.

The general syntax of subquery will be as follows:

```
Query
 (subquery)
```

The following is a simple example of a subquery.

```
select name, qual from faculty
where  fcode in
       ( select fcode from course_faculty
         where  ccode = 'ora')
```

```
NAME                           QUAL
------------------------------ ------------------------------
George Koch                    MS Computer Science
Jason Couchman                 OCP DBA
Kevin Loney                    MS Electronics
```

In the above example we will take the details of faculty members who can handle course *ora*. COURSE_FACULTY table contains information about which faculty members can take course ora. So we first use a subquery to get the list of the codes of the faculty members who can handle Oracle course. Then we send the list of faculty codes to outer query, which will then display the details of those faculty members.

Subquery is always executed first and the result is passed to the main query. Main query is executed by taking the result from subquery.

IN operator plays a very important role in subqueries as generally subqueries generate a list of values and main query is to compare a single value against the list of values supplied by subquery.

In the above example, subquery supplies the list of faculty codes to main query. Then main query compares each faculty code of the FACULTY table with the list supplied by subquery. If the faculty code exists in the list then it will display the details of the faculty.

The following are a few other examples of subqueries.

Get the details of students who have paid today.

```
select * from students  where  rollno in
   ( select rollno from payments where trunc(dp) = trunc(sysdate));
```

Display the details of batches handled by faculty name 'Kevin Loney'.

```
select * from batches
where fcode =
        ( select fcode from faculty where  name = 'Kevin Loney');
```

The following query displays the details of the faculty members who have not taken any batch in the last three months.

```
select * from faculty
where  fcode NOT IN
  (select fcode from batches
   where months_between(sysdate,stdate) <= 3)
```

For example, the following is invalid.

```
SQL> select * from  subjects
  2  where fulldur
  3      between 25
  4      and  (select max(fulldur) from subjects);

         and  (select max(fulldur) from subjects)
              *
ERROR at line 4:
ORA-00936: missing expression
```

Multiple Subqueries

It is possible for a main query to receive values from more than one subquery. The following example displays the details of batches that are taken by faculty with qualification MS or the course fee is more than 5000.

```
select  from batches
where fcode in
      (select  fcode from faculty where qual like '%MS%')
   or  ccode in
      ( select ccode from courses where fee > 5000);

BCODE CCODE FCODE STDATE    ENDDATE      TIMING
----- ----- ----- --------- --------- ---------
b1    ora   gk    12-JAN-01 20-FEB-01         1
b3    c     hs    20-JAN-01 27-FEB-01         3
b4    xml   dh    02-MAR-01 30-MAR-01         3
b5    java  hs    05-APR-01 10-MAY-01         1
b6    vbnet da    12-JUL-01                   1
```

In the above query first the subquery – `select fcode from faculty where qual like '%MS%'` – is executed. It then retrieves faculty codes where qualification of the faculty contains MS. Then it executes second subquery and then returns course codes of the courses where FEE is more than 5000. After both the subqueries are executed then the main query is executed with the data that is passed by subqueries to main query.

Nesting Subquery

It is also possible to nest subqueries. So far we have seen examples where a single subquery is executed and sends values to main query. It is also possible for a subquery to depend on another subquery and that subquery on another and so on.

The following example displays the details of the students who belong to batches that are taken by faculty with qualification MS.

```
select rollno,name, bcode  from students
where  bcode in
 ( select  bcode from batches
   where  fcode in ( select fcode from faculty where qual like '%MS%')
 );
```

```
   ROLLNO NAME                           BCODE
--------- ------------------------------ -----
        1 George Micheal                 b1
        2 Micheal Douglas                b1
        6 Chirs Evert                    b3
        7 Ivan Lendal                    b3
        8 George Micheal                 b4
        9 Richard Marx                   b5
       10 Tina Turner                    b5
       11 Jody Foster                    b5
```

Note: *Subqueries can be nested up to 16 levels. But that limit is seldom reached. Moreover it is not recommended to use more than 3 levels of nes*ting considering the performance.

The following is another example where we will take details of payments made by students of the batch that started on 12-jul-2001.

```
select * from payments
where rollno in
  (select rollno from students
   where  bcode in
     (select bcode from batches
      where stdate = '12-jul-01'
     )
  );
```

Comparing more than one value

A subquery can return multiple columns. These multiple columns must be compared with multiple values. The following query displays the details of the batches that have taken maximum duration among the batches of the same course.

```
select * from batches
where (ccode, enddate-stdate) in
        (select  ccode, max(enddate-stdate)
         from batches
         group by ccode);
```

```
BCODE CCODE FCODE STDATE    ENDDATE      TIMING
----- ----- ----- --------- --------- ---------
b2    asp   da    15-JAN-01 05-MAR-01         2
b3    c     hs    20-JAN-01 27-FEB-01         3
b5    java  hs    05-APR-01 10-MAY-01         1
b1    ora   gk    12-JAN-01 20-FEB-01         1
b4    xml   dh    02-MAR-01 30-MAR-01         3
```

First subquery returns the course and maximum duration for that course from BATCHES table using MAX function and GROUP BY clause. Then the values are sent to main query where they are compared with CCODE and duration of each batch. If in a row of BATCHES table the CCODE and the duration are equivalent to CCODE and maximum duration of any of rows returned by subquery then the row of BATCHES table is selected.

Now let us see few more examples of subqueries.

Get the details of course that has highest course fee.

```
select ccode,name,fee  from courses
where fee =
         (select max(fee) from courses);
```

```
CCODE NAME                              FEE
----- ------------------------------ ---------
cs    C Sharp                         7000
```

Get the details of students who have made a payment in the last month but no in the current month.

```
select * from students
where rollno not in
  (select  rollno from payments
   where to_char(dp,'mmyyyy') = to_char(sysdate,'mmyyyy')
```

```
  )
and rollno in
  ( select rollno from payments
    where to_char(dp,'mmyyyy')=to_char(add_months(sysdate,-1),'mmyyyy')
  );
```

First we take roll numbers of students who have made payment in the current month and roll numbers of students who have made payment in the previous month. Then the outer query selects students who are not part of the first list (who made payment in current month) and part of second list (who made payment in previous month).

Subqueries in DML and DDL commands

Subqueries can also be used with DML commands. WHERE clause of UPDATE and DELETE can always contain a subquery. The following UPDATE command increases the FEE of the course if more than 5 batches have started for that course.

```
update courses set fee = fee * 1.1
where  ccode in
 (  select  ccode  from  batches
    group by ccode
    having count(*) > 5);
```

In the above example, subquery returns course codes for which more than 5 batches have started and then UPDATE will update only those courses.

The following DELETE command uses subquery in WHERE clause to find out batches for which there are no students.

```
delete from batches
where  bcode not in ( select bcode from students);
```

The examples above are the cases where we used subquery in WHERE clause and not precisely in DML command. The following UPDATE command updates FCODE of batch **b7** to the faculty code of the batch **b1**.

```
update batches set fcode =
  (select fcode from batches where bcode = 'b1')
where bcode = 'b7';
```

Subquery returns a single value that is to be copied to FCODE of UPDATE.

Similarly the following INSERT inserts rows into a table called COMP_BATCH. The data is taken from BATCHES table.

The following subquery creates a new table from an existing table.

```
create table new_batches
as select bcode, ccode,fcode, stdate ,timing from batches
where stdate > sysdate;
```

The subquery is used to retrieve the data using which the new table is created. The structure of the new table will be same as the structure of the query. In the above query as query selects BCODE,CCODE,FCODE, STDATE , and TIMING columns the table is also created with same columns.

The following is the structure of the new table.

```
SQL> desc new_batches
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 BCODE                                              VARCHAR2(5)
 CCODE                                              VARCHAR2(5)
 FCODE                                              VARCHAR2(5)
 STDATE                                    NOT NULL DATE
 TIMING                                             NUMBER(1)



SQL> desc batches
 Name                                       Null?    Type
 ----------------------------------------- -------- ------------------
 BCODE                                      NOT NULL VARCHAR2(5)
 CCODE                                               VARCHAR2(5)
 FCODE                                               VARCHAR2(5)
 STDATE                                     NOT NULL DATE
 ENDDATE                                             DATE
 TIMING                                              NUMBER(1)
```

If you observe the above two structures, you would notice that only NOT NULL constraint of STDATE of BATCHES table is taken to NEW_BATCHES table.

```
select * from new_batches
```

Will display the details of batches where STDATE of the batch is after SYSDATE.

If you want to insert details again of new batches after some time you can issue the following INSERT command.

```
insert into new_batches
  select bcode, ccode, fcode, stdate , timing from batches
  where  stdate > sysdate;
```

Note: *The order and type of columns in the query and NEW_BATCHES should be same. Otherwise you may have to list out columns in the query according to the requirement.*

Renaming a column using subquery

The following procedure will illustrate how to use subquery with DDL to rename a column in a table. Renaming a column is not permitted in Oracle. So to rename a column, follow the given procedure. However, it is to be noted that this procedure is lengthy and not very refined. But you can consider in case of desperate need.

Assume we created a table called PRODUCTS as follows.

```
create table products
( id number(5) primary key ,
  name varchar2(30),
  qty  number(4)  check ( qty >= 0 ),
  pric number(5)
);
```

But the column PRIC is misspelt. It should have been PRICE. Now let us see how to rename the column.

First create a new table called newproducts using a subquery. Give an alias to column PRIC so that the alias becomes the column name in the new table.

```
create table newproducts
  as  select  id, name, qty, pric  price from products;

see the structure of the new table using DESC command.

desc newproducts

name                                              Null?    Type
------------------------------------------------- -------- -------------
ID                                                         NUMBER(5)
NAME                                                       VARCHAR2(30)
QTY                                                        NUMBER(4)
PRICE                                                      NUMBER(5)
```

As you can see in the output of DESC command, no constraint are defined in new table - NEWPRODUCTS. So we need to define all constraints again on this table using ALTER TABLE

command. But before that let us drop original table and rename NEWPRODUCTS to PRODUCTS.

SQL> drop table products;

Table dropped.

Note: *If the table being dropped has any dependent tables with rows then you have to drop those rows also. For example, if you have SALES table referring to PRODUCTS table then first SALES table is to be emptied before rows in PRODUCTS table can be deleted.*

SQL> rename newproducts to products;

Table renamed.

Now we have to define all constraints that we had in PRODUCTS table. This step is required as constraints of PRODUCTS table are copied to NEWPRODUCTS table.

```
alter table products
  add ( constraint products_pk  primary key(id));


alter table products
  add ( constraint  products_qty_chk check(qty >= 0));
```

Now see the structure of the new table using DESCRIBE command and constraints using USER_CONSTRAINTS view.

What's new in Oracle8i?

Orale8i has introduced to new possibilities related to subqueries.

Subquery in VALUES clause

Since Oracle8i it is possible to use a subquery in VALUES clause of INSERT command. Prior to Oracle8i it was possible to use subquery with INSERT command but not in VALUES clause of INSERT command.

The following INSERT command gets the next ROLLNO for a new student by using a subquery.

insert into students values (**(select max(rollno) + 1 from students)**,
  'b7', 'Robert Lafore', 'm', sysdate, null, null);

ORDER BY is permitted in subquery – TOPn analysis

Oracle8i has allowed the ORDER BY clause to be used with subquery. The following query will use ORDER BY clause to get courses in descending order. Then main query will take the data

sent by the subquery and selects only first two rows. As the result, the query will display the details of course with first two highest course fee.

ROWNUM pseudo column contains the row number for the retrieved rows. The query uses ROWNUM and takes only those rows that have row number less than 3.

```
select ccode,name,fee from
   (select *  from courses order by  fee desc)
    where  rownum < 3;

CCODE NAME                              FEE
----- ------------------------------ ---------
cs    C Sharp                          7000
vbnet VB.NET                           5500
```

Correlated Subquery

If there is any correlation between main query and subquery then subquery is called as ***correlated subquery***.

A correlated subquery is a subquery that receives some input from main query and sends result back to main query. Unlike normal subquery, a correlated subquery receives value from main query. It uses the value (generally in condition) and sends the results of the query back to main query.

Though most of the requirements can be accomplished with normal subqueries, some requirements do need correlated subquery. For example, we have to display the details of the batches where duration of the batch is more than the average duration of all the batches of that course.

Before we write the required query, let us look at the data present in BATCHES table.

```
select * from batches;

BCODE CCODE FCODE STDATE    ENDDATE      TIMING
----- ----- ----- --------- --------- ---------
b1    ora   gk    12-JAN-01 20-FEB-01         1
b2    asp   da    15-JAN-01 05-MAR-01         2
b3    c     hs    20-JAN-01 27-FEB-01         3
b4    xml   dh    02-MAR-01 30-MAR-01         3
b5    java  hs    05-APR-01 10-MAY-01         1
b6    vbnet da    12-JUL-01 31-AUG-01         1
b7    ora   gk    15-AUG-01 04-OCT-01         2
b9    ora   kl    05-MAY-01 10-JUN-01         3
b10   c     kl    15-JUN-01 20-JUL-01         2
```

```
b11   vbnet da    15-JUN-01 30-JUL-01         2
```

Now let us write correlated query to get details of batches where duration (ENDDATE-STDATE) is more than the average duration of the course.

```
select * from batches b1
 where  enddate - stdate >=
 (select avg(enddate - stdate)
  from batches
  where  ccode = b1.ccode);

BCODE CCODE FCODE STDATE    ENDDATE      TIMING
----- ----- ----- --------- --------- ---------
b2    asp   da    15-JAN-01 05-MAR-01         2
b3    c     hs    20-JAN-01 27-FEB-01         3
b4    xml   dh    02-MAR-01 30-MAR-01         3
b5    java  hs    05-APR-01 10-MAY-01         1
b6    vbnet da    12-JUL-01 31-AUG-01         1
b7    ora   gk    15-AUG-01 04-OCT-01         2
```

In correlated subquery, subquery is executed once for each row of the main query. This is required as subquery uses a value sent from main query, which may change from row to row in main query.

A correlated subquery is identified by the use of a column of main query in the subquery. In the above example, for each row of the BATCHES table the course code (CCODE) is passed to subquery and then subquery finds out the average duration of all batches of that course. The condition *ccode = b1.ccode* is used to select batches that belong to the course to which the batch in the main query belongs.

When subquery returns the average duration then main query checks whether duration of the batch is greater than or equal to the average duration sent by subquery. If the condition is satisfied then row in the main query is selected otherwise it is not selected. The process continues with the next row in the main query and so on until all rows of the main query are processed.

Differences between normal and correlated subquery

The following are the differences between a correlated subquery and ordinary subquery.

Subquery	Correlated subquery
Executed only for once before main-query	Executed once for each row of main-query.
Sends a value to main-query.	Receives value(s) from main query and sends value(s) to main-query.

Table 1: Normal subquery Vs. Correlated subquery

The following is another example of correlated subquery where we get third highest course fee.

```
select name, fee from courses c1
where  2= ( select count(*)
from  courses
where  fee  >  c1.fee);
```

```
NAME                        FEE
--------------------------- ---------
ASP.NET                     5000
```

In the above example, subquery returns the number of courses where FEE is more than the fee of the row in the main query. If the count returned by subquery is equal to 2 then it means above that course fee there are two more. That means that course fee becomes the third highest.

The above query is pretty useful. Especially prior to Oracle8i. As Oracle8i allowed ORDER BY clause in subquery, what was achieved with this correlated subquery can to some extent be achieved with ORDER BY clause in subquery.

The following two queries, will display the details of courses with two lowest fees.

The first query uses ORDER BY clause in subquery to retrieve courses in the ascending order of FEE. Then main query retrieves only first two rows of the subquery.

```
select * from ( select ccode, name, fee from courses order by fee)
where  rownum < 3;
```

```
CCODE NAME                             FEE
----- ------------------------------ ---------
c     C programming                     3500
xml   XML Programming                   4000
```

The second query uses a correlated subquery to get number of rows where FEE is less than the FEE of the row in main query. If the count returned by subquery is greater than or equal to 1 then row of the main query is retrieved.

```
select ccode,name, fee from courses c1
where  1 >= ( select count(*)
from courses
```

```
where fee < c1.fee);

CCODE NAME                              FEE
----- ------------------------------ ---------
c     C programming                     3500
xml   XML Programming                   4000
```

EXISTS and NOT EXISTS operators

These two operators are exclusively used in correlated subquery. EXISTS checks whether any row is returned by subquery and condition will be true if subquery returns any rows. Whereas, NOT EXISTS returns true if subquery doesn't retrieve any row.

EXISTS is different from other operators like IN,ANY etc., because it doesn't compare values of columns, instead, it checks whether any row is retrieved from subquery or not. If any row is retrieved from subquery the EXISTS returns true otherwise it returns false.

The following query displays details of courses for which at least one batch started in this month.

```
select * from courses
where  exists
( select * from batches
  where courses.ccode= ccode
  and  to_char(stdate,'mmyy') = to_char(sysdate,'mmyy')
);
```

When using EXISTS operator, what you select in the inner-query does NOT matter. What does matter is whether any row is retrieved by inner query or not.

The following example displays the details of courses for which no batch has yes started.

```
select ccode,name,fee from courses
where not exists
  ( select  * from batches
    where  ccode  = courses.ccode);
```

```
CCODE NAME                              FEE
----- ------------------------------ ---------
cs    C Sharp                           7000
```

It is possible to replace the above NOT EXISTS with NOT IN operator as shown below.

```
select ccode,name,fee from courses
where ccode not in  ( select ccode from batches);

CCODE NAME                              FEE
----- ------------------------------ ---------
cs    C Sharp                         7000
```

Let us see two other operators ALL and ANY.

ANY and ALL Operators

Both are used for comparing one value against a set of values. ALL specifies that all the values given in the list should be taken into account, whereas ANY specifies that the condition is satisfied when any of the values satisfies the condition.

```
        operator  ANY  list

        operator  ALL   list
```

The operator can be any one of the standard relational operators (=, >=, >, <, <=, !=) , and list is a series of values.

What if you want to display name of the courses where FEE is more than FEE of any course with DURATION 25? The following query will do just that.

```
select name from courses
where  fee > any ( select fee from courses where duration = 25);

NAME
------------------------------
VB.NET
ASP.NET
C Sharp
```

ANY operator specifies if FEE is more than any value in the list supplied by subquery then the condition is true. The same query can also be written as follows using MIN function.

```
select name from courses
where  fee > ( select min(fee) from courses where duration = 25);
```

```
NAME
------------------------------
VB.NET
ASP.NET
C Sharp
```

The following list illustrates the result of ANY and ALL operator.

Rate	ANY/ALL Operator	Result
10	Rate > ANY (15,20)	False
10	Rate > ANY (5,15)	True
10	Rate > ALL (10,20)	False
10	Rate > ALL (5,7)	True

Summary

Subquery is a query placed within another query. A subquery may return either one or multiple rows. Understanding how to use subqueries is very important. So, make sure you are comfortable with subqueries. Because subqueries, joining, and grouping the data are the areas which you must master.

Subqueries can be used with DML and even DDL commands. Oracle8i has allowed ORDER BY clause in subqueries. This allows top n analysis. Oracle8i has also allowed usage of subquery in VALUES clause of INSERT command.

When a subquery takes data from main query it is called as correlated subquery. And operators EXISTS and NOT EXISTS are used exclusively with correlated subquery.

Exercises

1. A correlated subquery is executed for _____ number of times.
2. Subquery nesting can be up to _____ levels.
3. What is the result of x > ANY (10,20), if x is 15? _____.
4. Subquery always passes the result to the main-query [T/F] _____
5. Subquery can be used in VALUES clause of INSERT command.[T/F] ____.
6. Display details of courses taken by students who joined in the month of june, 2001.
7. Delete the details of students who haven't paid anything so far.
8. Display the details of course for which there are more than 3 batches.
9. Display the details of course that has highest number of batches.

10. Change the ENDDATE of batch B8 to the ENDDATE of most recent batch.
11. Display the details of students who haven't paid total amount so far.
12. Display the details of payment made by students of Oracle batch started on 5-dec-2000.