Lab 4 - Joining Two or More Tables and SQL Sub-queries

1 JOINING TWO OR MORE TALBES

All the previous query examples that you have worked on are related to only one table. The join feature of SQL lets you select data from two or more tables and combine the selected data into a single query result.

1.1 Relationships Between Tables

The relationships between rows in one table and rows in another table are represented by the values stored in fields of those rows. For example, the EMPLOYEE table and the DEPARTMENT table each have a column DEPTNO that contains department numbers. It is the department numbers stored in both these columns that allows you to relate rows in the department table to rows in the employee table.

1.2 Selecting Data From Two Or More Tables

Let's say you want to know the location of the employee named ALLEN. Note that the EMP table does not contain a location (LOC) column but the DEPT table does. By looking at the employee table, you can see that ALLEN works for department 30. By looking at the DEPT table you can see that the department 30 is located in CHICAGO. Thus, you were able to relate one row in the EMP table to another row in the DEPT table by using data in a column present in both tables: the DEPTNO column.
Similarly, Oracle can "look" at data values stored in tables, and use those values to relate or *join* rows in one table to a row in other tables. You list the tables to be joined in the FROM clause and the relationship between the tables in the WHERE clause.

- Find Allen's name from the EMP table and location of Allen's department from the DEPT table.(join.1)

```
SQL > SELECT ENAME, LOC
      2 FROM EMP, DEPT
      3 WHERE ENAME = 'ALLEN'
      4 AND EMP.DEPTNO = DEPT.DEPTNO;
```

The WHERE clause says to retrieve only the rows for employee ALLEN. The WHERE clause also specifies that if the department number of a row in the employee table is equal to the department number of a row in the department table (EMP.DEPTNO = DEPT.DEPTNO), then join the rows together. That is, join the ALLEN row from the EMP table to the department 30 row from the DEPT table. The SELECT clause directs that only the ENAME field from the EMP table and the LOC field from the DEPT table are to be retrieved from the joined row.

1.3 Prefixing Column Names With Table Names

Notice that the DEPTNO column name is prefixed with the table name EMP or DEPT (EMP.DEPTNO = DEPT.DEPTNO). This is because both the EMP table and the DEPT table have a column named DEPTNO. If a column name is unique among the tables listed in the FROM clause (ENAME for example) the column name need not be prefixed.

1.4 Equi-join

- Join the DEPT table to the EMP table.(join.2)

```
SQL > SELECT DEPT.DEPTNO,DNAME,JOB,ENAME
      2 FROM DEPT, EMP
```

```
3 WHERE DEPT.DEPTNO = EMP.DEPTNO
4 ORDER BY DEPT.DEPTNO;
```

There is only one search-condition in the WHERE clause in the above example and it specifies the relationship between the EMP table and the DEPT table. This special type of search-condition is called a *join-condition*. Specifically, this join is called an *equi-join* because the comparison operator in the join-condition is *equals*. In addition to the equi-join, there are several types of join-conditions, but most joins that you will do will be either equi-joins or *outer-joins*.

1.5 Using Table Labels To Abbreviate Table Names

Join queries can become rather tedious to type when column names have to be prefixed with table names.

- List the department name and all the fields from the employee table for employees that work in Chicago. (join.3)

```
SQL > SELECT DNAME, EMPNO, ENAME, JOB, MGR, HIREDATE,
2 SAL, COMM, EMP.DEPTNO
3 FROM EMP, DEPT
4 WHERE EMP.DEPTNO = DEPT.DEPTNO
5 AND LOC = 'CHICAGO'
6 ORDER BY EMP.DEPTNO;
```

SQL allows you to define a temporary label in the FROM clause by placing the label after the table name separated by a blank. You may then use these labels in place of the full table names within the query.

- Issue the same query as the last example but use temporary labels to abbreviate the table names. (join.4)

```
SQL > SELECT DNAME, E.*
2 FROM EMP E, DEPT D
3 WHERE E.DEPTNO = D.DEPTNO
4 AND LOC = 'CHICAGO'
5 ORDER BY E.DEPTNO;
```

In the example above the letter E refers to the EMP table and the letter D refers to the DEPT table. Also notice the use of E.* in the SELECT clause to retrieve all of the columns of the EMP table.

1.6 Joining A Table To Itself

You can use a table label for more than just abbreviating a table name in a query. It also lets you "join" a table to itself as thought it were two separate tables.

- For each employee whose salary exceeds his manager's salary, list the employees' names and salary and the manager's name and salary. (join.5)

```
SQL > SELECT WORKER.ENAME, WORKER.SAL, MANAGER.ENAME,
2 MANAGER.SAL
3 FROM EMP WORKER, EMP MANAGER
4 WHERE WORKER.MGR = MANAGER.EMPNO
5 AND WORKER.SAL > MANAGER.SAL;
```

In the above query, the EMP table is treated as if it were two separated tables named WORKER and MANAGER. Firstly all the WORKERs are joined to their

MANAGERs using the WORKER's manager's employee number (WORKER.MGR) and the MANAGER's employee number (MANAGER.EMPNO). For example, SCOTT's manager is JONES because SCOTT has a MGR column with a value of 7566 and JONES' EMPNO is 7566. The WHERE clause then eliminates all WORKER MANAGER pairs except those where the WORKER earn more than the manager (WORKER.SAL > MANAGER.SAL).
Note that the EMP table is *not physically* duplicated into two separate tables during the query processing.

1.7 Other Join Operator
All the examples so far show tables being joined using the equal comparison operator (including outer-join). As we said earlier, equal is by far the most common join-condition, but tables can be joined to another using any comparison operator including:

=	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
BETWEEN	
LIKE	

- Find all the employees that earn more then JONES.(join.6)

```
SQL > SELECT X.ENAME, X.SAL, X.JOB, Y.ENAME, Y.SAL, Y.JOB
  2 FROM EMP X, EMP Y
  3 WHERE X.SAL > Y.SAL
  4 AND Y.ENAME = 'JONES';
```

In the example above, the EMP table is joined to itself using the comparison operator greater than (>).

To demonstrate a BETWEEN join we will use a new table that contain salary grades.

- List the SALGRADE table. (join.7)

```
SQL > SELECT * FROM SALGRADE;
```

- Now find the salary grade of each employee by joining the EMP table to the SALGRADE table.(join.8)

```
SQL > SELECT GRADE, JOB, ENAME, SAL
  2 FROM EMP, SALGRADE
  3 WHERE SAL BETWEEN LOSAL AND HISAL
  4 ORDER BY GRADE, JOB;
```

In the BETWEEN join, the SAL field of each row of the employee table is "tested" to make sure it is both greater than or equal to LOSAL and less than or equal to HISAL of the SALOGRADE table.

1.8 Selecting All Possible Combinations of Rows
If the WHERE clause contains no join-condition at all, then all possible combinations of rows from tables listed in the FROM clause are displayed. This result, called a cartesian product, is normally not desired so that a join-condition is usually specified.

- Join the ALLEN row from the EMP table with all the rows of the DEPT table. (join. 9)

SQL > SELECT ENAME, LOC
2 FROM EMP, DEPT
3 WHERE ENAME = 'ALLEN';

2 SQL SUBQUERIES

One of the reasons why SQL is so powerful is that you can build complex queries out of several simple queries. This is possible because the WHERE clause of one query may contain another query (called a *sub-query*). Oracle uses the sub-query to "dynamically build" a search-condition from values stored in the database.

2.1 Sub-queries That Return Only One Value

Suppose you want to find all the employees that have the same job as JONES. To answer this question, you can issue the following two queries:

- Find JONES' job.(subq.1)

SQL > SELECT JOB
2 FROM EMP
3 WHERE ENAME = 'JONES';

- Now that the first query has informed you that JONES is a MANAGER, you can issue a second query to find all the manager. (subq.2)

SQL > SELECT ENAME, JOB
2 FROM EMP
3 WHERE JOB = 'MANAGER';

You can arrive at the same result that required the previous two queries by using a single query with an embedded sub-query.

- List the name and job of employees who have the same job as JONES. (subq.3)

SQL > SELECT ENAME, JOB FROM EMP
2 WHERE JOB =
3 (SELECT JOB FROM EMP WHERE ENAME = 'JONES');

The second SELECT command returns the value MANAGER. Oracle uses this value to build the search-condition WHERE JOB = 'MANAGER'. This dynamically constructed search-condition is then used by the first SELECT command (called the main query) to select the desired rows. Note that sub-queries must be enclosed in parentheses but need not be indented.

2.2 Sub-queries That Return A Set Of Values

If a sub-query may return a set of values (more than one) you must attach the word ANY or ALL to the comparison operator (=,!=,>,>=,<,<=) preceding the sub-query to clarify the meaning of your query.

- Find the employees that earn more than ANY employee in department 30. (subq.4)

SQL > SELECT DISTINCT SAL, JOB, ENAME, DEPTNO FROM EMP
2 WHERE SAL > ANY
3 (SELECT SAL FROM EMP
4 WHERE DEPTNO = 30)
5 ORDER BY SAL DESC;

If an employee's salary is more than ANY of the set of salaries returned by the sub-query, then that employee is included in the query result. The lowest salary in department 30 is 950(JAMES). Thus the main query returns only those employees that earn more than 950.

The next query returns only those employees that earn more than ALL employees in department 30.

- Find the employees that earn more then ALL employees in department 30. (subq.5)

```
SQL > SELECT SAL, JOB, ENAME, DEPTNO FROM EMP
  2 WHERE SAL > ALL
  3 (SELECT SAL FROM EMP
  4 WHERE DEPTNO = 30)
  5 ORDER BY SAL DESC;
```

Blake, the manager of department 30, earns 2,850. Thus the query above returns only those employees that earn more than 2,850.

The operator IN has the same meaning and may be substituted for =ANY and the operator NOT IN is the same as !=ALL.

- Find all the employees in department 10 that have a job that is the same as anyone in department 30. (subq.6)

```
SQL > SELECT ENAME, JOB FROM EMP
  2 WHERE DEPTNO = 10
  3 AND JOB IN
  4 (SELECT JOB FROM EMP
  5 WHERE DEPTNO = 30);
```

Another way to state the above query is, "Find all employees with a job that is IN the set of jobs returned by the sub-query." The sub-query returns a set of all the jobs held by employees in department 30. The main query "tests" each employees in department 10 to see if his job is in the set of jobs returned by the sub-query.

- Find all the employees in department 10 that have a job that is NOT the same as anyone in department 10. (subq.7)

```
SQL > SELECT ENAME, JOB FROM EMP
  2 WHERE DEPTNO = 10
  3 AND JOB NOT IN
  4 (SELECT JOB FROM EMP
  5 WHERE DEPTNO = 30);
```

Another way to state the above query is, "Find all employees with a job that is NOT IN the set of jobs returned by the sub-query".

2.3 Sub-queries That Return More Than One Column

Oracle allows you to have more then one column in the SELECT list of the sub-query.

- List the name, job title, and salary of employees who have the same job and salary as Ford. (subq.8)

```
SQL > SELECT ENAME, JOB, SAL
  2 FROM EMP
```

```
3 WHERE (JOB,SAL)=
4 (SELECT JOB, SAL FROM EMP
5 WHERE ENAME = 'FORD');
```

Parentheses must be used to enclose the SELECT list of a sub-query when it contains more then one column.

2.4 Compound Queries with Multiple Sub-queries

SQL lets you combine basic commands to construct more and more complex commands. It is in this way that SQL obtains its power without sacrificing simplicity. The WHERE clause of a query may contain a combination of standard search-conditions, join-conditions and multiple sub-queries as shown in the following examples. Oracle does not limit you to just one sub-query.

- List the name, job, and department of employees who have the same job as Jones, or a salary greater than or equal to Ford. (subq.9)

```
SQL > SELECT ENAME, JOB, DEPTNO, SAL FROM EMP
      2 WHERE JOB IN
      3 (SELECT JOB FROM EMP
      4 WHERE ENAME = 'JONES')
      5 OR SAL > =
      6 (SELECT SAL FROM EMP
      7 WHERE ENAME = 'FORD')
      8 ORDER BY JOB, SAL;
```

You may have up to 16 sub-queries "linked" to the next higher-level query.

- Find all the employees in department 10 that have a job that is the same as anyone in the SALES department. (subq.10)

```
SQL > SELECT ENAME, JOB FROM EMP
      2 WHERE DEPTNO = 10
      3 AND JOB IN
      4 (SELECT JOB FROM EMP
      5 WHERE DEPTNO IN
      6 (SELECT DEPTNO FROM DEPT
      7 WHERE DNAME = 'SALES'));
```

2.5 Synchronizing A Repeating Sub-query With A Main Query

Depending on how you structure your sub-query, it can operate in different ways. In the previous examples, the sub-query was executed once and the resulting value was substituted into the WHERE clause of the main query. The following example shows a sub-query that is *executed repeatedly*, once for each row considered for selection (called the *candidate row*) by the main query.

Let's say you want to find the department number, name and salary of the employees that earn more than the average salary in their department. Firstly you need a main query to select the desired data from the EMP table.

```
SELECT    DEPTNO,ENAME,SAL
FROM      EMP
WHERE     SAL > (average salary of candidate employee's department)
```

Next you need a sub-query that will calculate the average salary of an employee's department as that employee is being considered for selection by the main query.

```
SELECT   DEPTNO, ENAME, SAL
FROM     EMP X
WHERE    SAL >
         (SELCT AVG(SAL)
         FROM EMP
         WHERE DEPTNO = (department of candidate employee))
```

The sub-query does not know which department average to compute until it knows the DEPTNO of the candidate employee being processed by the main query. The main query tells the sub-query which department average to compute. The sub-query computes the average salary for the candidate employee's department. The main-query then compares the average salary with the candidate employee's salary.

- Find all the employees that earn more than the average salary of employees in their department. (subq.11)

```
SQL > SELECT DEPTNO, ENAME, SAL FROM EMP X
  2 WHERE SAL >
  3 (SELECT AVG(SAL) FROM EMP
  4 WHERE X.DEPTNO = DEPTNO)
  5 ORDER BY DEPTNO;
```

The table label X in the main query and WHERE clause in the sub-query tell Oracle to "synchronize" the sub-query with the main memory. Oracle calculates the average salary in the sub-query using the DEPTNO of the employee being considered for selection by the main query. X.DEPTNO in the WHERE clause of the sub-query refers to the candidate row's department number and specifies that it must be equal to the DEPTNO used to select rows for computing average salary in the sub-query.

3. CREATE VIEWS

A view is a window into the data in one or more tables. The view itself contains no data and takes up no space.

Syntax:

```
CREATE VIEW view_name
[(column_name[,column_name]…)]
AS SELECT_STATEMENT;
```

- Create a view with DEPTNO, DNAME, ENAME, JOB from the DEPT and EMP tables.

```
SQL > CREATE VIEW DEPT_EMP
  2 AS SELECT DEPT.DEPTNO, DNAME, ENAME, JOB
  3 FROM DEPT, EMP
  4 WHERE DEPT.DEPTNO = EMP.DEPTNO;

SQL > SELECT * FROM DEPT_EMP;
```