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.

```
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.

```
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.

```
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 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 though 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.

```
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 than JONES.

```
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.

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

- Now find the salary grade of each employee by joining the `EMP` table to the `SALGRADE` table.

```
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.

```
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.

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 managers.

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.

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.

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 subquery, 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 than `ALL` employees in department 30.

```
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.

```
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 employee in department 10 to see whether 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 30.

```
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 than 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.

```
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 than 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.

```
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.

```
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.

```
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;
```

4. PARAMETERIZED SQL

A database schema consisting of the following four relations is given below:

EMPLOYEE(*EmpNo, SurName, FirstName, Birthdate, Salary, DeptNo, SuperNo)
DEPARTMENT(*DeptNo, DName, Location, DBudget, DHeadNo) PROJECT(*PName, PBudget, AdminDeptNo)
ASSIGNMENT(*EmpNo, *PName, HoursPerWeek)

Birthdate is of type `DATE`. Salary is an annual salary and is of type `NUMBER`. `DBudget`, `PBudget` and `HoursPerWeek` are also type of `NUMBER`. All other fields are of type `CHAR` (or `VARCHAR`). `SuperNo` is the `EmpNo` of an employee's immediate supervisor. `DHeadNo` is the EmpNo of the head of the department. `AdminDeptNo` is the `DeptNo` of the department which administrates the project. The asterisk(*) indicates a field of the primary key.

Formulate the following queries in ORACLE SQL. Is there any query that cannot be formulated in a single SQL statement?

1. List the `EmpNo` of employees whose immediate supervisor is not the same as the head of his department.
2. For each department, determine by how much the total departmental budget exceeds the cost of employee salaries.
3. For each project give each department's weekly contribution in salary dollars. Assume the annual salary is based on a 52-week-year and a 40-hour-week.

4. Which project has the greatest number of employee-hours/week spent on it?

The following two queries require SQL statements which are "parameterized", i.e., using substitution variables which are user variables preceded by one or two ampersands (& or &&). See the attached pages for the way to use substitution variables, in conjunction with the `Connect By` and `Start With` clauses of SQL queries for data items that are of hierarchical relationships.

1. List the employees who will be at least 60 years old on a given date.
2. Find the `EmpNo` and Surname of all employees directly or indirectly supervised by an employee with a given `EmpNo`. Format the output to show the tree structure of the hierarchy.

Remarks: It is helpful to set up the database schema first, and insert some data tuples into the tables so that you can actually test your resultant queries against your database.

Using Substitution Variables

Suppose you want to write a query to list the employees with various jobs, not just those whose job is `SALESMAN`. You could do that by editing a different `CHAR` value into the `WHERE` clause each time you run the command, but there is an easier way.

By using a substitution variable in place of the value `SALESMAN` in the `WHERE` clause, you can get the same results you would get if you had written the values into the command itself.

A substitution variable is a user variable name preceded by one or two ampersands (&). When SQL*Plus encounters a substitution variable in a command, SQL*Plus executes the command as though it contained the value of the substitution variable, rather than the variable itself.

Where and How to Use Substitution Variables?

You can use substitution variables anywhere in SQL and SQL*Plus commands, except as the first word entered at the command prompt. When SQL*Plus encounters an undefined substitution variable in a command, SQL*Plus prompts you for the value.

You can enter any string at the prompt, even one containing blanks and punctuation. If the SQL command containing the reference should have quotation marks around the variable and you do not include them there, the user must include the quotes when prompted.

SQL*Plus reads your response from the keyboard, even if you have redirected terminal input or output to a file. If a terminal is not available (*if, for example, you run the command file in batch mode*), SQL*Plus uses the redirected file.

Example

Create a command file named `STATS`, to be used to calculate a subgroup statistic (*the maximum value*) on a numeric column: SQL> CLEAR BUFFER

```
buffer cleared
  SQL> INPUT
      1  SELECT &GROUP_COL,
      2  MAX(&NUMBER_COL) MAXIMUM
      3  FROM &TABLE
      4  GROUP BY &GROUP_COL
      5
  SQL> SAVE STATS
  Created file STATS.sql
  SQL>
```

Now run the command file STATS and respond as shown below to the prompts for values:

```
SQL> @STATS
Enter value for group_col: JOB
old   1: SELECT &GROUP_COL,
new   1: SELECT JOB,
Enter value for number_col: SAL
old   2: MAX(&NUMBER_COL) MAXIMUM
new   2: MAX(SAL) MAXIMUM
Enter value for table: EMP
old   3: FROM &TABLE
new   3: FROM EMP
Enter value for group_col: JOB
old   4: GROUP BY &GROUP_COL
```

```
new   4: GROUP BY JOB

JOB                MAXIMUM
--------------- ----------
CLERK                  800
MANAGER               2975
SALESMAN              1600
```

Avoiding Unnecessary Prompts for Values

Suppose you want the file STATS to include the minimum, sum, and average of the “number” column. You may have noticed that SQL*Plus prompted you twice for the value of GROUP_COL and once for the value of NUMBER_COL in the example, and that each GROUP_COL or NUMBER_COL had a single ampersand in front of it. If you were to add three more functions – using a single ampersand before each – to the command file, SQL*Plus would prompt you a total of four times for the value of the number column.

You can avoid being re-prompted for the group and number columns by adding a second ampersand in front of each GROUP_COL and NUMBER_COL in STATS. SQL*Plus automatically DEFINES any substitution variable preceded by two *ampersands*, but does not DEFINE those preceded by only one ampersand. Thus, when SQL*Plus encounters a substitution variable more than once during a session, SQL*Plus uses the DEFINED values for substitution variables preceded by two *ampersands*, and prompts again for substitution variables preceded by one ampersand. This feature would also be useful if you wanted to run the file using the same GROUP_COL and NUMBER_COL in a different table.

CONNECT BY and START WITH clauses

Purpose:	Display data based on hierarchical relationships.	
Prerequisites:	None	

Syntax	`SELECT …`
	`FROM …`
	`…`
	`[CONNECT BY {PRIOR expr operator expr`
	`| expr operator PRIOR expr}`
	`[START WITH condition]]`	

Keywords and Parameters	`PRIOR`	Specifies direction of tree traversal. ORACLE will visit the `PRIOR` node before visiting the node without the `PRIOR` clause.
	`START WITH` condition	
Usage Notes	The `CONNECT BY` clause is used for querying hierarchical relations. The `CONNECT BY` clause specifies that rows are to be retrieved in a hierarchical order, and defines the relationship to be used to connect table rows into a hierarchy.	
PRIOR	The `PRIOR` clause specifies the order of retrieval (*parents first or children first*). The `PRIOR` clause must be used before one of the expressions of the `CONNECT BY` clause. The `PRIOR` side represents the parent in each parent-child relationship; the other side represents the child. For example, `CONNECT BY PRIOR EMPNO = MGR` means that `CONNECT BY` will return managers (*identified by* `EMPNO`) before employees (*identified by* `MGR`). `CONNECT BY EMPNO = PRIOR MGR` means that employees will be displayed before managers. The number of `CONNECT BY` levels is limited by available user memory. You may not use the `CONNECT BY` clause in conjunction with sub-queries or joins.	

START WITH	START WITH identifies the row(s) to be used as the root(s) of the tree by specifying a condition that they must satisfy. The omission of START WITH is identical to starting with all rows that satisfy the WHERE clause of the SELECT statement. Subqueries are permitted in a START WITH clause. When a SELECT statement contains the CONNECT BY clause, it may use the pseudo-column LEVEL. LEVEL returns a number value of 1 for a root node, 2 for a child of a root, 3 for a grandchild, and so on.	
Use of ORDER BY	The use of ORDER BY destroys the order in which CONNECT BY returns rows. There is an implicit ORDER BY when you use CONNECT BY. If a row's level is less than the previous row's level, that row is considered a child of the previous row. ORDER BY, therefore, will mask the implicit order of rows returned by CONNECT BY. Although no error is returned, the use of ORDER BY negates the effect of CONNECT BY.	
Example	SQL> SELECT LPAD(' ',2*LEVEL) || ENAME ORG_CHARG, EMPNO,MGR,JOB FROM EMP 2 CONNECT BY PRIOR EMPNO = MGR 3 START WITH ENAME = 'KING';	
Related Topics		
LPAD	**Syntax**	LPAD(char1,n [,char2])
	Purpose	Return char1, left-padded to length n with the sequence of characters in char2; char2 defaults to blanks.
	Example	SELECT LPAD('Page 1', 14,'*.') "LPAD example" FROM DUAL LPAD example -------------------- .*.*.*.* .Page 1

This example shows how the data is displayed according to hierarchical relationships.

```
SQL> SET PAGESIZE 48
SQL> SELECT LPAD(' ', 2*LEVEL)||ENAME ORG_CHARG, EMPNO, MGR,
JOB FROM EMP
2   CONNECT BY PRIOR EMPNO = MGR
3   START WITH ENAME = 'KING';

ORG_CHARG
--------------------------------------------------------------
     EMPNO        MGR JOB
---------- ---------- ---------------
  KING
      7839            PRESIDENT

    JONES
      7655       7839 MANAGER

      SCOTT
      7788       7655 ANALYST

        ADAMS
      7876       7788 CLERK

      FORD
      7902       7655 ANALYST

        SMITH
      7369       7902 CLERK

    BLAKE
      7698       7839 MANAGER

      ALLEN
      7499       7698 SALESMAN

      WARD
      7521       7698 SALESMAN

      MARTIN
      7654       7698 SALESMAN

      TURNER
      7844       7698 SALESMAN

      JAMES
      7900       7698 CLERK

    CLARK
      7782       7839 MANAGER

      MILLER
      7934       7782 CLERK

14 rows selected.
```