# Lab 3. Working with Multiple Tables

This lab introduces the use of joins and set operations to combine data from multiple tables. Joins are the foundation of SQL. Set operations are also very important. If you want to master the complex queries found in the later labs, you must start here, with joins and set operations.

## 3.1. Stacking One Rowset atop Another

### PROBLEM

You want to return data stored in more than one table, conceptually stacking one result set atop the other. The tables do not necessarily have a common key, but their columns do have the same data types. For example, you want to display the name and department number of the employees in department 10 in table EMP, along with the name and department number of each department in table DEPT. You want the result set to look like the following:
ENAME_AND_DNAME DEPTNO
--------------- ----------
CLARK 10
KING 10
MILLER 10
----------
ACCOUNTING 10
RESEARCH 20
SALES 30
OPERATIONS 40

### SOLUTION

Use the set operation UNION ALL to combine rows from multiple tables:

```
 select ename as ename_and_dname, deptno         from emp        where deptno = 10       union all       select '----------', null
```

### DISCUSSION

UNION ALL combines rows from multiple row sources into one result set. As with all set operations, the items in all the SELECT lists must match in number and data type. For example, both of the following queries will fail:

```
 select deptno    from dept       union all      select ename     from emp
 select deptno, dname from dept union select deptno from emp
```

It is important to note, UNION ALL will include duplicates if they exist. If you wish to filter out duplicates, use the UNION operator. For example, a UNION between EMP.DEPTNO and DEPT.DEPTNO returns only four rows:

```
 select deptno    from emp        union  select deptno    from dept


     DEPTNO
   ---------
          10
          20
          30
          40
```

Specifying UNION rather than UNION ALL will most likely result in a sort operation in order to eliminate duplicates. Keep this in mind when working with large result sets. Using UNION is roughly equivalent to the following query, which applies DISTINCT to the output from a UNION ALL:

```
 select distinct deptno          from (         select deptno    from emp        union all      select deptno    from dept       )


     DEPTNO
   ---------
          10
          20
          30
          40
```

You wouldn’t use DISTINCT in a query unless you had to, and the same rule applies for UNION; don’t use it instead of UNION ALL unless you have to.

## 3.2. Combining Related Rows

### PROBLEM

You want to return rows from multiple tables by joining on a known common column or joining on columns that share common values. For example, you want to display the names of all employees in department 10 along with the location of each employee's department, but that data is stored in two separate tables. You want the result set to be the following:
ENAME LOC
---------- ----------
CLARK NEW YORK
KING NEW YORK
MILLER NEW YORK

### SOLUTION

Join table EMP to table DEPT on DEPTNO:

```
select e.ename, d.loc   from emp e, dept d      where e.deptno = d.deptno       and e.deptno = 10
```

### DISCUSSION

The solution is an example of a join, or more accurately an equi-join, which is a type of inner join. A join is an operation that combines rows from two tables into one. An equi-join is one in which the join condition is based on an equality condition (e.g., where one department number equals another). An inner join is the original type of join; each row returned contains data from each table.
Conceptually, the result set from a join is produced by first creating a Cartesian product (all possible combinations of rows) from the tables listed in the FROM clause, as seen below:

```
select e.ename, d.loc,          e.deptno as emp_deptno,         d.deptno as dept_deptno         from emp e, dept d      where e.deptno
```

```
ENAME      LOC            EMP_DEPTNO DEPT_DEPTNO
---------- -------------  ---------- -----------
CLARK      NEW YORK               10          10
KING       NEW YORK               10          10
MILLER     NEW YORK               10          10
CLARK      DALLAS                 10          20

KING       DALLAS                 10          20
MILLER     DALLAS                 10          20
CLARK      CHICAGO                10          30
KING       CHICAGO                10          30
MILLER     CHICAGO                10          30
CLARK      BOSTON                 10          40
KING       BOSTON                 10          40
MILLER     BOSTON                 10          40
```

Every employee in table EMP (in department 10) is returned along with every department in the table DEPT. Then, the expression in the WHERE clause involving e.deptno and d.deptno (the join) restricts the result set such that the only rows returned are the ones where EMP.DEPTNO and DEPT.DEPTNO are equal:

```
select e.ename, d.loc,          e.deptno as emp_deptno,         d.deptno as dept_deptno         from emp e, dept d      where e.deptno
```

```
ENAME       LOC             EMP_DEPTNO  DEPT_DEPTNO
----------  --------------  ----------  -----------
CLARK       NEW YORK                10           10
KING        NEW YORK                10           10
MILLER      NEW YORK                10           10
```

An alternative solution makes use of an explicit JOIN clause (the "INNER" keyword is optional):

```
select e.ename, d.loc   from emp e inner join dept d    on (e.deptno = d.deptno)        where e.deptno = 10
```

Use the JOIN clause if you prefer to have the join logic in the FROM clause rather than the WHERE clause. Both styles are ANSI compliant and work on all the latest versions of the RDBMSs.

## 3.3. Finding Rows in Common Between Two Tables

### PROBLEM

You want to find common rows between two tables but there are multiple columns on which you can join. For example, consider the following view V:

```
create view V   as      select ename,job,sal     from emp        where job = 'CLERK'
```

```
select * from V

ENAME       JOB              SAL
----------  --------- ----------
SMITH       CLERK            800
ADAMS       CLERK           1100
JAMES       CLERK            950
MILLER      CLERK           1300
```

Only clerks are returned from view V. However, the view does not show all possible EMP columns. You want to return the EMPNO, ENAME, JOB, SAL, and DEPTNO of all employees in EMP that match the rows from view V. You want the result set to be the following:

```
EMPNO ENAME JOB SAL DEPTNO
-------- ---------- --------- ---------- ---------
7369 SMITH CLERK 800 20
7876 ADAMS CLERK 1100 20
7900 JAMES CLERK 950 30
7934 MILLER CLERK 1300 10
```

# SOLUTION

Join the tables on all the columns necessary to return the correct result. Alternatively, use the set operation INTERSECT to avoid performing a join and instead return the intersection (common rows) of the two tables.

**MySQL and SQL Server**

Join table EMP to view V using multiple join conditions:

```
select e.empno,e.ename,e.job,e.sal,e.deptno     from emp e, V   where e.ename = v.ename         and e.job = v.job       and e.sal = v.
```

Alternatively, you can perform the same join via the JOIN clause:

```
select e.empno,e.ename,e.job,e.sal,e.deptno     from emp e join V       on ( e.ename = v.ename          and e.job = v.job       and e.
```

**DB2, Oracle, and PostgreSQL**

The MySQL and SQL Server solution also works for DB2, Oracle, and PostgreSQL. It’s the solution you should use if you need to return values from view V.

If you do not actually need to return columns from view V, you may use the set operation INTERSECT along with an IN predicate:

```
select empno,ename,job,sal,deptno       from emp        where (ename,job,sal) in (      select ename,job,sal from emp  intersect
```

## DISCUSSION

When performing joins, you must consider the proper columns to join on in order to return correct results. This is especially important when rows can have common values for some columns while having different values for others.

The set operation INTERSECT will return rows common to both row sources. When using INTERSECT, you are required to compare the same number of items, having the same data type, from two tables. When working with set operations keep in mind that, by default, duplicate rows will not be returned.

# 3.4. Retrieving Values from One Table That Do Not Exist in Another

## PROBLEM

You wish to find those values in one table, call it the source table, that do not also exist in some target table. For example, you want to find which departments (if any) in table DEPT do not exist in table EMP. In the example data, DEPTNO 40 from table DEPT does not exist in table EMP, so the result set should be the following:

```
DEPTNO
----------
40
```

## SOLUTION

Having functions that perform set difference is particularly useful for this problem. DB2, PostgreSQL, and Oracle support set difference operations. If your DBMS does not support a set difference function, use a subquery as shown for MySQL and SQL Server.

**DB2 and PostgreSQL**

Use the set operation EXCEPT:

```
select deptno from dept         except          select deptno from emp
```

__Oracle

Use the set operation MINUS:

```
 select deptno from dept        minus   select deptno from emp
```

**MySQL and SQL Server**

Use a subquery to return all DEPTNOs from table EMP into an outer query that searches table DEPT for rows that are not amongst the rows returned from the subquery:

```
 select deptno   from dept      where deptno not in (select deptno from emp)
```

## DISCUSSION

**DB2 and PostgreSQL**

The built-in functions provided by DB2 and PostgreSQL make this operation quite easy. The EXCEPT operator takes the first result set and removes from it all rows found in the second result set. The operation is very much like a subtraction.

There are restrictions on the use of set operators, including EXCEPT. Data types and number of values to compare must match in both SELECT lists. Additionally, EXCEPT will not return duplicates and, unlike a subquery using NOT IN, NULLs do not present a problem (see the discussion for MySQL and SQL Server). The EXCEPT operator will return rows from the upper query (the query before the EXCEPT) that do not exist in the lower query (the query after the EXCEPT).

**Oracle**

The Oracle solution is identical to that for DB2 and PostgreSQL, except that Oracle calls its set difference operator MINUS rather than EXCEPT. Otherwise, the preceding explanation applies to Oracle as well.

**MySQL and SQL Server**

The subquery will return all DEPTNOs from table EMP. The outer query returns all DEPTNOs from table DEPT that are “not in” or “not included in” the result set returned from the subquery.

Duplicate elimination is something you’ll want to consider when using the MySQL and SQL Server solutions. The EXCEPT- and MINUS-based solutions used for the other platforms eliminate duplicate rows from the result set, ensuring that each DEPTNO is reported only one time. Of course, that can only be the case anyway, as DEPTNO is a key field in my example data. Were DEPTNO not a key field, you could use DISTINCT as follows to ensure that each DEPTNO value missing from EMP is reported only once:

```
 select distinct deptno          from dept      where deptno not in (select deptno from emp)
```

Be mindful of NULLs when using NOT IN. Consider the following table, NEW_ DEPT:

```
 create table new_dept(deptno integer)  insert into new_deptvalues (10)        insert into new_dept values (50)       insert into new
```

If you try to find the DEPTNOs in table DEPT that do not exist in table NEW_DEPT and use a subquery with NOT IN, you’ll find that the query returns no rows:

```
 select *        from dept      where deptno not in (select deptno from new_dept)
```

DEPTNOs 20, 30, and 40 are not in table NEW_DEPT, yet were not returned by the query. Why? The reason is the NULL value present in table NEW_DEPT. Three rows are returned by the subquery, with DEPTNOs of 10, 50, and NULL. IN and NOT IN are essentially OR operations, and will yield different results because of how NULL values are treated by logical OR evaluations. To understand this, examine the truth tables below (Let T=true, F=false, N=null):

```
OR | T | F | N |
+----+---+---+----+
| T | T | T | T |
| F | T | F | N |
| N | T | N | N |
+----+---+---+----+

NOT |
+-----+---+
| T | F |
| F | T |
| N | N |
+-----+---+

AND | T | F | N |
+-----+---+---+---+
| T | T | F | N |
| F | F | F | F |
| N | N | F | N |
+-----+---+---+---+
```

Now consider the following example using IN and its equivalent using OR:

**select deptno**
**from dept**
**where deptno in ( 10,50,null )**

# DEPTNO

```
  10
```

*select deptno*
*from dept*
*where (deptno=10 or deptno=50 or deptno=null)*

# DEPTNO

```
  10
```

Why was only DEPTNO 10 returned? There are four DEPTNOs in DEPT, (10,20,30,40), each one is evaluated against the predicate (deptno=10 or deptno=50 or deptno=null). According to the truth tables above, for each DEPTNO (10,20,30,40), the predicate yields:
DEPTNO=10
(deptno=10 or deptno=50 or deptno=null)
= (10=10 or 10=50 or 10=null)
= (T or F or N)
= (T or N)
= (T)

DEPTNO=20
(deptno=10 or deptno=50 or deptno=null)
= (20=10 or 20=50 or 20=null)
= (F or F or N)
= (F or N)
= (N)

DEPTNO=30
(deptno=10 or deptno=50 or deptno=null)
= (30=10 or 30=50 or 30=null)
= (F or F or N)
= (F or N)
= (N)

DEPTNO=40
(deptno=10 or deptno=50 or deptno=null)
= (40=10 or 40=50 or 40=null)
= (F or F or N)
= (F or N)
= (N)
Now it is obvious why only DEPTNO 10 was returned when using IN and OR. Now consider the same example using NOT IN and NOT OR:

```
 select deptno from dept where deptno not in ( 10,50,null )
```

( no rows )

```
 select deptno from dept where not (deptno=10 or deptno=50 or deptno=null)
```

( no rows )
Why are no rows returned? Let’s check the truth tables:
DEPTNO=10
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (10=10 or 10=50 or 10=null)
= NOT (T or F or N)
= NOT (T or N)
= NOT (T)
= (F)

DEPTNO=20
NOT (deptno=10 or deptno=50 or deptno=null)

= NOT (20=10 or 20=50 or 20=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)

DEPTNO=30
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (30=10 or 30=50 or 30=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)

DEPTNO=40
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (40=10 or 40=50 or 40=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)
In SQL, “TRUE or NULL” is TRUE, but “FALSE or NULL” is NULL! You must keep this in mind when using IN predicates and when performing logical OR evaluations, and NULL values are involved.
To avoid the problem with NOT IN and NULLs, use a correlated subquery in conjunction with NOT EXISTS. The term "correlated subquery” is used because rows from the outer query are referenced in the subquery. The following example is an alternative solution that will not be affected by NULL rows (going back to the original query from the “Problem” section):

```
select d.deptno from dept d where not exists ( select 1 from emp e where d.deptno = e.deptno)
```

## DEPTNO

40

```
select d.deptno
from dept d
where not exists (
select 1
from new_dept nd
where d.deptno = nd.deptno
)
```

## DEPTNO

30
40
20
Conceptually, the outer query in this solution considers each row in the DEPT table. For each DEPT row, the following happens:

1. The subquery is executed to see whether the department number exists in the EMP table. Note the condition D.DEPTNO = E.DEPTNO, which brings together the department numbers from the two tables.
2. If the subquery returns results, then EXISTS (…) evaluates to true and NOT EXISTS (…) thus evaluates to FALSE, and the row being considered by the outer query is discarded.
3. If the subquery returns no results, then NOT EXISTS (…) evaluates to TRUE, and the row being considered by the outer query is returned (because it is for a department not represented in the EMP table).
   The items in the SELECT list of the subquery are unimportant when using a correlated subquery with EXISTS/NOT EXISTS, which is why I chose to select NULL, to force you to focus on the join in the subquery rather than the items in the SELECT list.

## 3.5. Retrieving Rows from One Table That Do Not Correspond to Rows in Another

### PROBLEM

You want to find rows that are in one table that do not have a match in another table, for two tables that have common keys. For example, you want to find which departments have no employees. The result set should be the following:

```
      DEPTNO  DNAME           LOC
     -------  --------------  -------------
          40  OPERATIONS      BOSTON
```

Finding the department each employee works in requires an equi-join on DEPTNO from EMP to DEPT. The DEPTNO column represents the common value between tables. Unfortunately, an equi-join will not show you which department has no employees. That's because by equi-joining EMP and DEPT you are returning all rows that satisfy the join condition. Instead you want only those rows from DEPT that do not satisfy the join condition.
This is a subtly different problem than in the preceding recipe, though at first glance they may seem the same. The difference is that the preceding recipe yields only a list of department numbers not represented in table EMP. Using this recipe, however, you can easily return other columns from the DEPT table; you can return more than just department numbers.

### SOLUTION

Return all rows from one table along with rows from another that may or may not have a match on the common column. Then, keep only those rows with no match.

**DB2, MySQL, PostgreSQL, SQL Server**

Use an outer join and filter for NULLs (keyword OUTER is optional):

```
select d.*      from dept d left outer join emp e       on (d.deptno = e.deptno)        where e.deptno is null
```

**Oracle**

For users on Oracle9i Database and later, the preceding solution will work. Alternatively, you can use the proprietary Oracle outer-join syntax:

```
select d.*      from dept d, emp e      where d.deptno = e.deptno (+)   and e.deptno is null
```

This proprietary syntax (note the use of the "+" in parens) is the only outer-join syntax available in Oracle8i Database and earlier.

### DISCUSSION

This solution works by outer joining and then keeping only rows that have no match. This sort of operation is sometimes called an anti-join. To get a better idea of how an anti-join works, first examine the result set without filtering for NULLs:

```
select e.ename, e.deptno as emp_deptno, d.*     from dept d left join emp e     on (d.deptno = e.deptno)
```

```
ENAME       EMP_DEPTNO      DEPTNO DNAME          LOC
----------  ----------  ---------- -------------- -------------
SMITH               20          20 RESEARCH       DALLAS
ALLEN               30          30 SALES          CHICAGO
WARD                30          30 SALES          CHICAGO
JONES               20          20 RESEARCH       DALLAS
MARTIN              30          30 SALES          CHICAGO
BLAKE               30          30 SALES          CHICAGO
CLARK               10          10 ACCOUNTING     NEW YORK
SCOTT               20          20 RESEARCH       DALLAS
KING                10          10 ACCOUNTING     NEW YORK
TURNER              30          30 SALES          CHICAGO
ADAMS               20          20 RESEARCH       DALLAS
JAMES               30          30 SALES          CHICAGO
FORD                20          20 RESEARCH       DALLAS
MILLER              10          10 ACCOUNTING     NEW YORK
                                40 OPERATIONS     BOSTON
```

Notice, the last row has a NULL value for EMP.ENAME and EMP_DEPTNO. That's because no employees work in department 40. The solution uses the WHERE clause to keep only rows where EMP_DEPTNO is NULL (thus keeping only rows from DEPT that have no match in EMP).

## 3.6. Adding Joins to a Query Without Interfering with Other Joins

### PROBLEM

You have a query that returns the results you want. You need additional information, but when trying to get it, you lose data from the original result set. For example, you want to return all employees, the location of the department in which they work, and the date they received a bonus. For this problem, the EMP_BONUS table contains the following data:

```
select * from emp_bonus
```

```
      EMPNO  RECEIVED          TYPE
 ----------  ----------- ----------
       7369  14-MAR-2005          1
       7900  14-MAR-2005          2
       7788  14-MAR-2005          3
```

The query you start with looks like this:

```
 select e.ename, d.loc    from emp e, dept d      where e.deptno=d.deptno
```

```
   ENAME      LOC
   ---------- -------------
   SMITH      DALLAS
   ALLEN      CHICAGO
   WARD       CHICAGO
   JONES      DALLAS
   MARTIN     CHICAGO
   BLAKE      CHICAGO
   CLARK      NEW YORK
   SCOTT      DALLAS
   KING       NEW YORK
   TURNER     CHICAGO
   ADAMS      DALLAS
   JAMES      CHICAGO
   FORD       DALLAS
   MILLER     NEW YORK
```

You want to add to these results the date a bonus was given to an employee, but joining to the EMP_BONUS table returns fewer rows than you wish because not every employee has a bonus:

```
 select e.ename, d.loc,eb.received       from emp e, dept d, emp_bonus eb        where e.deptno=d.deptno         and e.empno=eb.empno
```

```
 ENAME       LOC           RECEIVED
 ----------  ------------- -----------
 SCOTT       DALLAS        14-MAR-2005
 SMITH       DALLAS        14-MAR-2005
 JAMES       CHICAGO       14-MAR-2005
```

Your desired result set is the following:

```
ENAME LOC RECEIVED
---------- ------------- -----------
ALLEN CHICAGO
WARD CHICAGO
MARTIN CHICAGO
JAMES CHICAGO 14-MAR-2005
TURNER CHICAGO
BLAKE CHICAGO
SMITH DALLAS 14-MAR-2005
FORD DALLAS
ADAMS DALLAS
JONES DALLAS
SCOTT DALLAS 14-MAR-2005
CLARK NEW YORK
KING NEW YORK
MILLER NEW YORK
```

## SOLUTION

You can use an outer join to obtain the additional information without losing the data from the original query. First join table EMP to table DEPT to get all employees and the location of the department they work, then outer join to table EMP_ BONUS to return the date of the bonus if there is one. Following is the DB2, MySQL, PostgreSQL, and SQL Server syntax:

```
 select e.ename, d.loc, eb.received      from emp e join dept d          on (e.deptno=d.deptno)          left join emp_bonus eb
```

If you are using Oracle9i Database or later, the preceding solution will work for you. Alternatively, you can use Oracle's proprietary outer-join syntax, which is your only choice when using Oracle8i Database and earlier:

```
 select e.ename, d.loc, eb.received      from emp e, dept d, emp_bonus eb        where e.deptno=d.deptno         and e.empno=eb.empno (
```

You can also use a scalar subquery (a subquery placed in the SELECT list) to mimic an outer join:

```
 select e.ename, d.loc,          (select eb.received from emp_bonus eb   where eb.empno=e.empno) as received     from emp e, dept d
```

The scalar subquery solution will work across all platforms.

### DISCUSSION

An outer join will return all rows from one table and matching rows from another. See the previous recipe for another example of such a join. The reason an outer join works to solve this problem is that it does not result in any rows being eliminated that would otherwise be returned. The query will return all the rows it would return without the outer join. And it also returns the received date, if one exists.

Use of a scalar subquery is also a convenient technique for this sort of problem, as it does not require you to modify already correct joins in your main query. Using a scalar subquery is an easy way to tack on extra data to a query without compromising the current result set. When working with scalar subqueries, you must ensure they return a scalar (single) value. If a subquery in the SELECT list returns more than one row, you will receive an error.

**SEE ALSO**

See “Converting a Scalar Subquery to a Composite Subquery in Oracle” in lab 14 for a workaround to the problem of not being able to return multiple rows from a SELECT-list subquery.

## 3.7. Determining Whether Two Tables Have the Same Data

### PROBLEM

You want to know if two tables or views have the same data (cardinality and values). Consider the following view:

~~~
 create view V  as      select * from emp where deptno != 10     union all      select * from emp where ename = 'WARD'


  ```select * from V```

  EMPNO ENAME       JOB        MGR  HIREDATE      SAL  COMM DEPTNO
  ----- ----------  --------- ----- ----------- ----- ----- ------
   7369 SMITH       CLERK      7902 17-DEC-1980   800           20
   7499 ALLEN       SALESMAN   7698 20-FEB-1981  1600   300     30
   7521 WARD        SALESMAN   7698 22-FEB-1981  1250   500     30
   7566 JONES       MANAGER    7839 02-APR-1981  2975           20
   7654 MARTIN      SALESMAN   7698 28-SEP-1981  1250  1400     30
   7698 BLAKE       MANAGER    7839 01-MAY-1981  2850           30
   7788 SCOTT       ANALYST    7566 09-DEC-1982  3000           20
   7844 TURNER      SALESMAN   7698 08-SEP-1981  1500     0     30
   7876 ADAMS       CLERK      7788 12-JAN-1983  1100           20
   7900 JAMES       CLERK      7698 03-DEC-1981   950           30
   7902 FORD        ANALYST    7566 03-DEC-1981  3000           20
   7521 WARD        SALESMAN   7698 22-FEB-1981  1250   500     30
~~~

You want to determine whether or not this view has exactly the same data as table EMP. The row for employee “WARD” is duplicated to show that the solution will reveal not only different data but duplicates as well. Based on the rows in table EMP the difference will be the three rows for employees in department 10 and the two rows for employee “WARD”. You want to return the following result set:

```
EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO CNT
----- ---------- --------- ----- ----------- ----- ----- ------ ---
7521 WARD SALESMAN 7698 22-FEB-1981 1250 500 30 1
7521 WARD SALESMAN 7698 22-FEB-1981 1250 500 30 2
7782 CLARK MANAGER 7839 09-JUN-1981 2450 10 1
7839 KING PRESIDENT 17-NOV-1981 5000 10 1
7934 MILLER CLERK 7782 23-JAN-1982 1300 10 1
```

### SOLUTION

Functions that perform SET difference (MINUS or EXCEPT, depending on your DBMS) make the problem of comparing tables a relatively easy one to solve. If your DBMS does not offer such functions, you can use a correlated subquery.

DB2 and PostgreSQL

Use the set operations EXCEPT and UNION ALL to find the difference between view V and table EMP combined with the difference between table EMP and view V:

```
 (       select empno,ename,job,mgr,hiredate,sal,comm,deptno,    count(*) as cnt         from V          group by empno,ename,job,mgr,h
```

**Oracle**

Use the set operations MINUS and UNION ALL to find the difference between view V and table EMP combined with the difference between table EMP and view V:

```
 (       select empno,ename,job,mgr,hiredate,sal,comm,deptno,    count(*) as cnt         from V          group by empno,ename,job,mgr,h
```

**MySQL and SQL Server**

Use a correlated subquery and UNION ALL to find the rows in view V and not in table EMP combined with the rows in table EMP and not in view V:

```
select *
from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
e.sal,e.comm,e.deptno, count() as cnt
from emp e
group by empno,ename,job,mgr,hiredate,
sal,comm,deptno
) e
where not exists (
select null
from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
v.sal,v.comm,v.deptno, count() as cnt
from v
group by empno,ename,job,mgr,hiredate,
sal,comm,deptno
) v
where v.empno = e.empno
and v.ename = e.ename
and v.job = e.job
and coalesce(v.mgr,0) = coalesce(e.mgr,0)
and v.hiredate = e.hiredate
and v.sal = e.sal
and v.deptno = e.deptno
and v.cnt = e.cnt
and coalesce(v.comm,0) = coalesce(e.comm,0)
)
union all
select *
from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
v.sal,v.comm,v.deptno, count() as cnt
from v
group by empno,ename,job,mgr,hiredate,
sal,comm,deptno
) v
where not exists (
select null
from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
e.sal,e.comm,e.deptno, count() as cnt
from emp e
group by empno,ename,job,mgr,hiredate,
sal,comm,deptno
) e
where v.empno = e.empno
and v.ename = e.ename
and v.job = e.job
and coalesce(v.mgr,0) = coalesce(e.mgr,0)
and v.hiredate = e.hiredate
and v.sal = e.sal
and v.deptno = e.deptno
and v.cnt = e.cnt
and coalesce(v.comm,0) = coalesce(e.comm,0)
)
```

## DISCUSSION

Despite using different techniques, the concept is the same for all solutions:

1. First, find rows in table EMP that do not exist in view V.
2. Then combine (UNION ALL) those rows with rows from view V that do not exist in table EMP.
   If the tables in question are equal, then no rows are returned. If the tables are different, the rows causing the difference are returned. As an easy first step when comparing tables, you can compare the cardinalities alone rather than including them with the data comparison. The following query is a simple example of this and will work on all DBMSs:

   ```
     from emp
    union
   select count(*)
     from dept

   COUNT(*)
   --------
          4
         14
   ```

Because UNION will filter out duplicates, only one row will be returned if the tables' cardinalities are the same. Because two rows are returned in this example, you know that the tables do not contain identical rowsets.
DB2, Oracle, and PostgreSQL
MINUS and EXCEPT work in the same way, so I will use EXCEPT for this discussion. The queries before and after the UNION ALL are very similar. So, to understand how the solution works, simply execute the query prior to the UNION ALL by itself. The following result set is produced by executing lines 1–11 in the solution section:
(
select empno,ename,job,mgr,hiredate,sal,comm,deptno,
count(*) as cnt*
*from V*
*group by empno,ename,job,mgr,hiredate,sal,comm,deptno*
*except*
*select empno,ename,job,mgr,hiredate,sal,comm,deptno,*
*count(*) as cnt
from emp
group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)

```
 EMPNO ENAME      JOB        MGR  HIREDATE      SAL  COMM DEPTNO CNT
 ----- ---------- --------- ----- ----------- ----- ----- ------ ---
 7521  WARD       SALESMAN   7698 22-FEB-1981  1250   500     30   2
```

The result set represents a row found in view V that is either not in table EMP or has a different cardinality than that same row in table EMP. In this case, the duplicate row for employee “WARD” is found and returned. If you're still having trouble understanding how the result set is produced, run each query on either side of EXCEPT individually. You'll notice the only difference between the two result sets is the CNT for employee “WARD” returned by view V.
The portion of the query after the UNION ALL does the opposite of the query preceding UNION ALL. The query returns rows in table EMP not in view V:
(
select empno,ename,job,mgr,hiredate,sal,comm,deptno,
count(*) as cnt*
*from emp*
*group by empno,ename,job,mgr,hiredate,sal,comm,deptno*
*minus*
*select empno,ename,job,mgr,hiredate,sal,comm,deptno,*
*count(*) as cnt
from v
group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)

```
EMPNO ENAME      JOB         MGR HIREDATE      SAL  COMM DEPTNO CNT
----- ---------- --------- ----- ----------- ----- ----- ------ ---
7521  WARD       SALESMAN   7698 22-FEB-1981  1250   500     30   1
7782  CLARK      MANAGER    7839 09-JUN-1981  2450           10   1
7839  KING       PRESIDENT       17-NOV-1981  5000           10   1
7934  MILLER     CLERK      7782 23-JAN-1982  1300           10   1
```

The results are then combined by UNION ALL to produce the final result set.

MySQL and SQL Server

The queries before and after the UNION ALL are very similar. To understand how the subquery-based solution works, simply execute the query prior to the UNION ALL by itself. The query below is from lines 1–27 in the solution:

```
select *
from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
e.sal,e.comm,e.deptno, count() as cnt
from emp e
group by empno,ename,job,mgr,hiredate,
sal,comm,deptno
) e
where not exists (
select null
from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
v.sal,v.comm,v.deptno, count() as cnt
from v
group by empno,ename,job,mgr,hiredate,
sal,comm,deptno
) v
where v.empno = e.empno
and v.ename = e.ename
and v.job = e.job
and v.mgr = e.mgr
and v.hiredate = e.hiredate
and v.sal = e.sal
and v.deptno = e.deptno
and v.cnt = e.cnt
and coalesce(v.comm,0) = coalesce(e.comm,0)
)
```

```
EMPNO ENAME      JOB         MGR HIREDATE      SAL  COMM DEPTNO CNT
----- ---------- --------- ----- ----------- ----- ----- ------ ---
 7521 WARD       SALESMAN   7698 22–FEB–1981  1250   500     30   1
 7782 CLARK      MANAGER    7839 09–JUN–1981  2450           10   1
 7839 KING       PRESIDENT       17–NOV–1981  5000           10   1
 7934 MILLER     CLERK      7782 23–JAN–1982  1300           10   1
```

Notice that the comparison is not between table EMP and view V, but rather between inline view E and inline view V. The cardinality for each row is found and returned as an attribute for that row. You are comparing each row and its occurrence count. If you are having trouble understanding how the comparison works, run the subqueries independently. The next step is to find all rows (including CNT) in inline view E that do not exist in inline view V. The comparison uses a correlated subquery and NOT EXISTS. The joins will determine which rows are the same, and the result will be all rows from inline view E that are not the rows returned by the join. The query after the UNION ALL does the opposite; it finds all rows in inline view V that do not exist in inline view E:

```
select *
from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
v.sal,v.comm,v.deptno, count() as cnt
from v
group by empno,ename,job,mgr,hiredate,
sal,comm,deptno
) v
where not exists (
select null
from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
e.sal,e.comm,e.deptno, count() as cnt
from emp e
group by empno,ename,job,mgr,hiredate,
sal,comm,deptno
) e
where v.empno = e.empno
and v.ename = e.ename
and v.job = e.job
and v.mgr = e.mgr
and v.hiredate = e.hiredate
and v.sal = e.sal
and v.deptno = e.deptno
and v.cnt = e.cnt
and coalesce(v.comm,0) = coalesce(e.comm,0)
)
```

```
 EMPNO ENAME      JOB         MGR HIREDATE      SAL  COMM DEPTNO CNT
 ----- ---------- --------- ----- ----------- ----- ----- ------ ---
  7521 WARD       SALESMAN   7698 22-FEB-1981  1250   500     30   2
```

The results are then combined by UNION ALL to produce the final result set.

TIP

Ales Spectic and Jonathan Gennick give an alternate solution in their book Transact-SQL Cookbook (O’Reilly). See the section "Comparing Two Sets for Equality” in lab 2.

## 3.8. Identifying and Avoiding Cartesian Products

### PROBLEM

You want to return the name of each employee in department 10 along with the location of the department. The following query is returning incorrect data:

```
select e.ename, d.loc
from emp e, dept d
where e.deptno = 10
```

```
 ENAME       LOC
 ----------  -------------
 CLARK       NEW YORK
 CLARK       DALLAS
 CLARK       CHICAGO
 CLARK       BOSTON
 KING        NEW YORK
 KING        DALLAS
 KING        CHICAGO
 KING        BOSTON
 MILLER      NEW YORK
 MILLER      DALLAS
 MILLER      CHICAGO
 MILLER      BOSTON
```

The correct result set is the following:

```
ENAME LOC
---------- ---------
CLARK NEW YORK
KING NEW YORK
MILLER NEW YORK
```

### SOLUTION

Use a join between the tables in the FROM clause to return the correct result set:

```
select e.ename, d.loc
from emp e, dept d
where e.deptno = 10
and d.deptno = e.deptno
```

### DISCUSSION

Looking at the data in the DEPT table:

```
select * from dept
```

```
     DEPTNO DNAME          LOC
 ---------- -------------- -------------
         10 ACCOUNTING     NEW YORK
         20 RESEARCH       DALLAS
         30 SALES          CHICAGO
         40 OPERATIONS     BOSTON
```

You can see that department 10 is in New York, and thus you can know that returning employees with any location other than New York is incorrect. The number of rows returned by the incorrect query is the product of the cardinalities of the two tables in the FROM clause. In the original query, the filter on EMP for department 10 will result in three rows. Because there is no filter for DEPT, all four rows from DEPT are returned. Three multiplied by four is twelve, so the incorrect query returns twelve rows. Generally, to avoid a Cartesian product you would apply the n–1 rule where n represents the number of tables in the FROM clause and n–1 represents the minimum number of joins necessary to avoid a Cartesian product. Depending on what the keys and join columns in your tables are, you may very well need more than n–1 joins, but n–1 is a good place to start when writing queries.
TIP
When used properly, Cartesian products can be very useful. The recipe, , uses a Cartesian product and is used by many other queries. Common uses of Cartesian products include transposing or pivoting (and unpivoting) a result set, generating a sequence of values, and mimicking a loop.

# 3.9. Performing Joins when Using Aggregates

## PROBLEM

You want to perform an aggregation but your query involves multiple tables. You want to ensure that joins do not disrupt the aggregation. For example, you want to find the sum of the salaries for employees in department 10 along with the sum of their bonuses. Some employees have more than one bonus and the join between table EMP and table EMP_BONUS is causing incorrect values to be returned by the aggregate function SUM. For this problem, table EMP_BONUS contains the following data:
select * from emp_bonus

```
 EMPNO RECEIVED          TYPE
 ----- ----------- ----------
  7934 17-MAR-2005          1
  7934 15-FEB-2005          2
  7839 15-FEB-2005          3
  7782 15-FEB-2005          1
```

Now, consider the following query that returns the salary and bonus for all employees in department 10. Table BONUS.TYPE determines the amount of the bonus. A type 1 bonus is 10% of an employee's salary, type 2 is 20%, and type 3 is 30%.

```
select e.empno,         e.ename,        e.sal,          e.deptno,       e.sal*case when eb.type = 1 then .1     when eb.type = 2 then
```

```
   EMPNO ENAME             SAL     DEPTNO     BONUS
 ------- ---------- ---------- ---------- ---------
    7934 MILLER           1300         10       130
    7934 MILLER           1300         10       260
    7839 KING             5000         10      1500
    7782 CLARK            2450         10       245
```

So far, so good. However, things go awry when you attempt a join to the EMP_ BONUS table in order to sum the bonus amounts:

```
select deptno,          sum(sal) as total_sal,          sum(bonus) as total_bonus       from (          select e.empno,          e.enam
```

```
 DEPTNO   TOTAL_SAL  TOTAL_BONUS
 ------ -----------  -----------
     10       10050         2135
```

While the TOTAL_BONUS is correct, the TOTAL_SAL is incorrect. The sum of all salaries in department 10 is 8750, as the following query shows:

```
select sum(sal) from emp where deptno=10
```

```
   SUM(SAL)
 ----------
       8750
```

Why is TOTAL_SAL incorrect? The reason is the duplicate rows in the SAL column created by the join. Consider the following query, which joins table EMP and EMP_ BONUS:

```
select e.ename,         e.sal   from emp e, emp_bonus eb        where e.empno = eb.empno        and e.deptno = 10
```

```
ENAME             SAL
---------- ----------
CLARK            2450
KING             5000
MILLER           1300
MILLER           1300
```

Now it is easy to see why the value for TOTAL_SAL is incorrect: MILLER’s salary is counted twice. The final result set that you are really after is:

```
DEPTNO TOTAL_SAL TOTAL_BONUS
------ --------- -----------
10 8750 2135
```

## SOLUTION

You have to be careful when computing aggregates across joins. Typically when duplicates are returned due to a join, you can avoid miscalculations by aggregate functions in two ways: you can simply use the keyword DISTINCT in the call to the aggregate function, so only unique instances of each value are used in the computation; or you can perform the aggregation first (in an inline view) prior to joining, thus avoiding the incorrect computation by the aggregate function because the aggregate will already be computed before you even join, thus avoiding the problem altogether. The solutions that follow use DISTINCT. The “Discussion” section will discuss the technique of using an inline view to perform the aggregation prior to joining.

MySQL and PostgreSQL

Perform a sum of only the DISTINCT salaries:

```
select deptno,
sum(distinct sal) as total_sal,
sum(bonus) as total_bonus
from (
select e.empno,
e.ename,
e.sal,
e.deptno,
e.sal*case when eb.type = 1 then .1
when eb.type = 2 then .2
else .3
end as bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
and e.deptno = 10
) x
group by deptno
```

**DB2, Oracle, and SQL Server**

These platforms support the preceding solution, but they also support an alternative solution using the window function SUM OVER:

```
 select distinct deptno,total_sal,total_bonus    from (          select e.empno,         e.ename,        sum(distinct e.sal) over
```

## DISCUSSION

**MySQL and PostgreSQL**

The second query in the “Problem” section of this recipe joins table EMP and table EMP_BONUS and returns two rows for employee “MILLER”, which is what causes the error on the sum of EMP.SAL (the salary is added twice). The solution is to simply sum the distinct EMP.SAL values that are returned by the query. The following query is an alternative solution—necessary if there could be duplicate values in the column you are summing. The sum of all salaries in department 10 is computed first and that row is then joined to table EMP, which is then joined to table EMP_BONUS. The following query works for all DBMSs:

```
select d.deptno,
d.total_sal,
sum(e.sal*case when eb.type = 1 then .1
when eb.type = 2 then .2
else .3 end) as total_bonus
from emp e,
emp_bonus eb,
(
select deptno, sum(sal) as total_sal
from emp
where deptno = 10
group by deptno
) d
where e.deptno = d.deptno
and e.empno = eb.empno
group by d.deptno,d.total_sal
```

```
    DEPTNO  TOTAL_SAL  TOTAL_BONUS
 --------- ---------- ------------
        10       8750         2135
```

DB2, Oracle, and SQL Server

This alternative solution takes advantage of the window function SUM OVER. The following query is taken from lines 3–14 in “Solution” and returns the following result set:

```
select e.empno,
e.ename,
sum(distinct e.sal) over
(partition by e.deptno) as total_sal,
e.deptno,
sum(e.sal*case when eb.type = 1 then .1
when eb.type = 2 then .2
else .3 end) over
(partition by deptno) as total_bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
and e.deptno = 10
```

```
 EMPNO ENAME        TOTAL_SAL   DEPTNO  TOTAL_BONUS
 ----- ----------  ----------   ------  -----------
  7934 MILLER            8750       10         2135
  7934 MILLER            8750       10         2135
  7782 CLARK             8750       10         2135
  7839 KING              8750       10         2135
```

The windowing function, SUM OVER, is called twice, first to compute the sum of the distinct salaries for the defined partition or group. In this case, the partition is DEPTNO 10 and the sum of the distinct salaries for DEPTNO 10 is 8750. The next call to SUM OVER computes the sum of the bonuses for the same defined partition. The final result set is produced by taking the distinct values for TOTAL_SAL, DEPTNO, and TOTAL_BONUS.

## 3.10. Performing Outer Joins when Using Aggregates

### PROBLEM

Begin with the same problem as in 3.9, but modify table EMP_BONUS such that the difference in this case is not all employees in department 10 have been given bonuses. Consider the EMP_BONUS table and a query to (ostensibly) find both the sum of all salaries for department 10 and the sum of all bonuses for all employees in department 10:

```
select * from emp_bonus
```

```
      EMPNO RECEIVED          TYPE
 ---------- ----------- ----------
       7934 17-MAR-2005          1
       7934 15-FEB-2005          2


 select deptno,
        sum(sal) as total_sal,
        sum(bonus) as total_bonus
   from (
 select e.empno,
        e.ename,
        e.sal,
        e.deptno,
        e.sal*case when eb.type = 1 then .1
                   when eb.type = 2 then .2
                   else .3 end as bonus
   from emp e, emp_bonus eb
  where e.empno  = eb.empno
    and e.deptno = 10
        )
  group by deptno

  DEPTNO  TOTAL_SAL TOTAL_BONUS
  ------ ---------- -----------
      10       2600         390
```

The result for TOTAL_BONUS is correct, but the value returned for TOTAL_SAL does not represent the sum of all salaries in department 10. The following query shows why the TOTAL_SAL is incorrect:

```
select e.empno,
e.ename,
e.sal,
e.deptno,
e.sal*case when eb.type = 1 then .1
when eb.type = 2 then .2
else .3 end as bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
and e.deptno = 10
```

```
     EMPNO ENAME          SAL     DEPTNO      BONUS
 --------- ---------  ------- ---------- ----------
      7934 MILLER        1300         10        130
      7934 MILLER        1300         10        260
```

Rather than sum all salaries in department 10, only the salary for "MILLER" is summed and it is erroneously summed twice. Ultimately, you would like to return the following result set:

```
DEPTNO TOTAL_SAL TOTAL_BONUS
------ --------- -----------
10 8750 390
```

## SOLUTION

The solution is similar to that of 3.9, but here you outer join to EMP_BONUS to ensure all employees from department 10 are included.

DB2, MySQL, PostgreSQL, SQL Server

Outer join to EMP_BONUS, then perform the sum on only distinct salaries from department 10:

```
select deptno,
sum(distinct sal) as total_sal,
sum(bonus) as total_bonus
from (
select e.empno,
e.ename,
e.sal,
e.deptno,
e.salcase when eb.type is null then 0
when eb.type = 1 then .1
when eb.type = 2 then .2
else .3 end as bonus
from emp e left outer join emp_bonus eb
on (e.empno = eb.empno)
where e.deptno = 10
)
group by deptno
```

*You can also use the window function SUM OVER:*

```
select distinct deptno,total_sal,total_bonus
from (
select e.empno,
e.ename,
sum(distinct e.sal) over
(partition by e.deptno) as total_sal,
e.deptno,
sum(e.salcase when eb.type is null then 0
when eb.type = 1 then .1
when eb.type = 2 then .2
else .3
end) over
(partition by deptno) as total_bonus
from emp e left outer join emp_bonus eb
on (e.empno = eb.empno)
where e.deptno = 10
) x
```

**Oracle**

If you are using Oracle9i Database or later you can use the preceding solution. Alternatively, you can use the proprietary Oracle outer-join syntax, which is mandatory for users on Oracle8i Database and earlier:

```
select deptno,
sum(distinct sal) as total_sal,
sum(bonus) as total_bonus
from (
select e.empno,
e.ename,
e.sal,
e.deptno,
e.salcase when eb.type is null then 0
when eb.type = 1 then .1
when eb.type = 2 then .2
else .3 end as bonus
from emp e, emp_bonus eb
where e.empno = eb.empno (+)
and e.deptno = 10
)
group by deptno
```

*Oracle 8i Database users can also use the SUM OVER syntaxshown for DB2 and the other databases, but must modify it to use the proprietary Oracle outer-join syntax shown in the preceding query.*

*DISCUSSION*

*The second query in the “Problem” section of this recipe joins table EMP and table EMP_BONUS and returns only rows for employee “MILLER”, which is what causes the error on the sum of EMP.SAL (the other employees in DEPTNO 10 do not have bonuses and their salaries are not included in the sum). The solution is to outer join table EMP to table EMP_BONUS so even employees without a bonus will be included in the result. If an employee does not have a bonus, NULL will be returned for EMP_BONUS.TYPE. It is important to keep this in mind as the CASE statement has been modified and is slightly different from solution 3.9. If EMP_BONUS.TYPE is NULL, the CASE expression returns zero, which has no effect on the sum.*

*The following query is an alternative solution. The sum of all salaries in department 10 is computed first, then joined to table EMP, which is then joined to table EMP_BONUS (thus avoiding the outer join). The following query works for all DBMSs:*

```
select d.deptno,
d.total_sal,
sum(e.salcase when eb.type = 1 then .1
when eb.type = 2 then .2
else .3 end) as total_bonus
from emp e,
emp_bonus eb,
(
select deptno, sum(sal) as total_sal
from emp
where deptno = 10
group by deptno
) d
where e.deptno = d.deptno
and e.empno = eb.empno
group by d.deptno,d.total_sal
```

```
    DEPTNO  TOTAL_SAL TOTAL_BONUS
 --------- ---------- -----------
        10       8750         390
```

# 3.11. Returning Missing Data from Multiple Tables

## PROBLEM

You want to return missing data from multiple tables simultaneously. Returning rows from table DEPT that do not exist in table EMP (any departments that have no employees) requires an outer join. Consider the following query, which returns all DEPTNOs and DNAMEs from DEPT along with the names of all the employees in each department (if there is an employee in a particular department):

```
select d.deptno,d.dname,e.ename
from dept d left outer join emp e
on (d.deptno=e.deptno)
```

```
    DEPTNO DNAME          ENAME
 --------- -------------- ----------
        20 RESEARCH       SMITH
        30 SALES          ALLEN
        30 SALES          WARD
        20 RESEARCH       JONES
        30 SALES          MARTIN
        30 SALES          BLAKE
        10 ACCOUNTING     CLARK
        20 RESEARCH       SCOTT
        10 ACCOUNTING     KING
        30 SALES          TURNER
        20 RESEARCH       ADAMS
        30 SALES          JAMES
        20 RESEARCH       FORD
        10 ACCOUNTING     MILLER
        40 OPERATIONS
```

The last row, the OPERATIONS department, is returned despite that department not having any employees, because table EMP was outer joined to table DEPT. Now, suppose there was an employee without a department. How would you return the above result set along with a row for the employee having no department? In other words, you want to outer join to both table EMP and table DEPT, and in the same query. After creating the new employee, a first attempt may look like this:

```
insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
select 1111,'YODA','JEDI',null,hiredate,sal,comm,null
from emp
where ename = 'KING'
select d.deptno,d.dname,e.ename
from dept d right outer join emp e
on (d.deptno=e.deptno)
```

```
    DEPTNO DNAME        ENAME
---------- ------------ ----------
        10 ACCOUNTING   MILLER
        10 ACCOUNTING   KING
        10 ACCOUNTING   CLARK
        20 RESEARCH     FORD
        20 RESEARCH     ADAMS
        20 RESEARCH     SCOTT
        20 RESEARCH     JONES
        20 RESEARCH     SMITH
        30 SALES        JAMES
        30 SALES        TURNER
        30 SALES        BLAKE
        30 SALES        MARTIN
        30 SALES        WARD
        30 SALES        ALLEN
                        YODA
```

This outer join manages to return the new employee but lost the OPERATIONS department from the original result set. The final result set should return a row for YODA as well as OPERATIONS, such as the following:

```
DEPTNO DNAME ENAME
---------- ------------ --------
10 ACCOUNTING CLARK
10 ACCOUNTING KING
10 ACCOUNTING MILLER
20 RESEARCH ADAMS
20 RESEARCH FORD
20 RESEARCH JONES
20 RESEARCH SCOTT
20 RESEARCH SMITH
30 SALES ALLEN
30 SALES BLAKE
30 SALES JAMES
30 SALES MARTIN
30 SALES TURNER
30 SALES WARD
40 OPERATIONS
YODA
```

## SOLUTION

Use a full outer join to return missing data from both tables based on a common value.

DB2, MySQL, PostgreSQL, SQL Server

Use the explicit FULL OUTER JOIN command to return missing rows from both tables along with matching rows:

```
select d.deptno,d.dname,e.ename
from dept d full outer join emp e
on (d.deptno=e.deptno)
```

Alternatively, since MySQL does not yet have a FULL OUTER JOIN, union the results of the two different outer joins:

```
select d.deptno,d.dname,e.ename
from dept d right outer join emp e
on (d.deptno=e.deptno)
union
select d.deptno,d.dname,e.ename
from dept d left outer join emp e
on (d.deptno=e.deptno)
```

**Oracle**

If you are on Oracle9i Database or later, you can use either of the preceding solutions. Alternatively, you can use Oracle’s proprietary outer join syntax, which is the only choice for users on Oracle8i Database and earlier:

```
select d.deptno,d.dname,e.ename
from dept d, emp e
where d.deptno = e.deptno(+)
union
select d.deptno,d.dname,e.ename
```

```
from dept d, emp e
where d.deptno(+) = e.deptno
```

### DISCUSSION

The full outer join is simply the combination of outer joins on both tables. To see how a full outer join works “under the covers,” simply run each outer join, then union the results. The following query returns rows from table DEPT and any matching rows from table EMP (if any).

```
select d.deptno,d.dname,e.ename
from dept d left outer join emp e
on (d.deptno = e.deptno)
```

```
   DEPTNO DNAME          ENAME
   ------ -------------- ----------
       20 RESEARCH       SMITH
       30 SALES          ALLEN
       30 SALES          WARD
       20 RESEARCH       JONES
       30 SALES          MARTIN
       30 SALES          BLAKE
       10 ACCOUNTING     CLARK
       20 RESEARCH       SCOTT
       10 ACCOUNTING     KING
       30 SALES          TURNER
       20 RESEARCH       ADAMS
       30 SALES          JAMES
       20 RESEARCH       FORD
       10 ACCOUNTING     MILLER
       40 OPERATIONS
```

This next query returns rows from table EMP and any matching rows from table DEPT (if any):

```
select d.deptno,d.dname,e.ename
from dept d right outer join emp e
on (d.deptno = e.deptno)
```

```
  DEPTNO DNAME          ENAME
  ------ -------------- ----------
      10 ACCOUNTING     MILLER
      10 ACCOUNTING     KING
      10 ACCOUNTING     CLARK
      20 RESEARCH       FORD
      20 RESEARCH       ADAMS
      20 RESEARCH       SCOTT
      20 RESEARCH       JONES
      20 RESEARCH       SMITH
      30 SALES          JAMES
      30 SALES          TURNER
      30 SALES          BLAKE
      30 SALES          MARTIN
      30 SALES          WARD
      30 SALES          ALLEN
                        YODA
```

The results from these two queries are unioned to provide the final result set.

## 3.12. Using NULLs in Operations and Comparisons

### PROBLEM

NULL is never equal to or not equal to any value, not even itself, but you want to evaluate values returned by a nullable column like you would evaluate real values. For example, you want to find all employees in EMP whose commission (COMM) is less than the commission of employee “WARD”. Employees with a NULL commission should be included as well.

### SOLUTION

Use a function such as COALESCE to transform the NULL value into a real value that can be used in standard evaluation:

```
select ename,comm
from emp
```

```
where coalesce(comm,0) < ( select comm
from emp
where ename = 'WARD' )
```

## DISCUSSION

The COALESCE function will return the first non-NULL value from the list of values passed to it. When a NULL value is encountered it is replaced by zero, which is then compared with Ward’s commission. This can be seen by putting the COALESCE function in the SELECT list:

```
select ename,comm,coalesce(comm,0)
from emp
where coalesce(comm,0) < ( select comm
from emp
where ename = 'WARD' )
```

```
ENAME            COMM COALESCE(COMM,0)
---------- ---------- ----------------
SMITH                                0
ALLEN             300              300
JONES                                0
BLAKE                                0
CLARK                                0
SCOTT                                0
KING                                 0
TURNER              0                0
ADAMS                                0
JAMES                                0
FORD                                 0
MILLER                               0
```