Lab 8. Date Arithmetic

This Lab introduces techniques for performing simple date arithmetic. Recipes cover common tasks like adding days to dates, finding the number of business days between dates, and finding the difference between dates in days.

Being able to successfully manipulate dates with your RDBMS's built-in functions can greatly improve your productivity. For all the recipes in this Lab, I try to take advantage of each RDBMS's built-in functions. In addition, I have chosen to use one date format for all the recipes, "DD-MON-YYYY". I chose to do this because I believe it will benefit those of you who work with one RDBMS and want to learn others. Seeing one standard format will help you focus on the different techniques and functions provided by each RDBMS without having to worry about default date formats.

TIP
This Lab focuses on basic date arithmetic. You'll find more advanced date recipes in the following Lab. The recipes presented in this Lab use simple date data types. If you are using more complex date data types you will need to adjust the solutions accordingly.

8.1. Adding and Subtracting Days, Months, and Years

PROBLEM

You need to add or subtract some number of days, months, or years from a date. For example, using the HIREDATE for employee CLARK you want to return six different dates: five days before and after CLARK was hired, five months before and after CLARK was hired, and, finally, five years before and after CLARK was hired. CLARK was hired on "09-JUN-1981", so you want to return the following result set:

```
HD_MINUS_5D HD_PLUS_5D  HD_MINUS_5M HD_PLUS_5M  HD_MINUS_5Y HD_PLUS_5Y
----------- ----------- ----------- ----------- ----------- -----------
04-JUN-1981 14-JUN-1981 09-JAN-1981 09-NOV-1981 09-JUN-1976 09-JUN-1986
12-NOV-1981 22-NOV-1981 17-JUN-1981 17-APR-1982 17-NOV-1976 17-NOV-1986
18-JAN-1982 28-JAN-1982 23-AUG-1981 23-JUN-1982 23-JAN-1977 23-JAN-1987
```

SOLUTION

DB2
Standard addition and subtraction is allowed on date values, but any value that you add to or

subtract from a date must be followed by the unit of time it represents:

```
select hiredate -5 day   as hd_minus_5D,
       hiredate +5 day   as hd_plus_5D,
       hiredate -5 month as hd_minus_5M,
       hiredate +5 month as hd_plus_5M,
       hiredate -5 year  as hd_minus_5Y,
       hiredate +5 year  as hd_plus_5Y
  from emp
 where deptno = 10
```

Oracle

Use standard addition and subtraction for days, and use the ADD_MONTHS function to add and subtract months and years:

```
select hiredate-5                 as hd_minus_5D,
       hiredate+5                 as hd_plus_5D,
       add_months(hiredate,-5)    as hd_minus_5M,
       add_months(hiredate,5)     as hd_plus_5M,
       add_months(hiredate,-5*12) as hd_minus_5Y,
       add_months(hiredate,5*12)  as hd_plus_5Y
  from emp
 where deptno = 10
```

PostgreSQL

Use standard addition and subtraction with the INTERVAL keyword specifying the unit of time to add or subtract. Single quotes are required when specifying an INTERVAL value:

```
select hiredate - interval '5 day'   as hd_minus_5D,
       hiredate + interval '5 day'   as hd_plus_5D,
       hiredate - interval '5 month' as hd_minus_5M,
       hiredate + interval '5 month' as hd_plus_5M,
       hiredate - interval '5 year'  as hd_minus_5Y,
       hiredate + interval '5 year'  as hd_plus_5Y
  from emp
 where deptno=10
```

MySQL

Use standard addition and subtraction with the INTERVAL keyword specifying the unit of time to add or subtract. Unlike the PostgreSQL solution, you do not place single quotes around the INTERVAL value:

```
select hiredate - interval 5 day   as hd_minus_5D,
       hiredate + interval 5 day   as hd_plus_5D,
       hiredate - interval 5 month as hd_minus_5M,
       hiredate + interval 5 month as hd_plus_5M,
       hiredate - interval 5 year  as hd_minus_5Y,
       hiredate + interval 5 year  as hd_plus_5Y
  from emp
 where deptno=10
```

Alternatively, you can use the DATE_ADD function, which is shown below:

```
select date_add(hiredate,interval -5 day)   as hd_minus_5D,
       date_add(hiredate,interval  5 day)   as hd_plus_5D,
       date_add(hiredate,interval -5 month) as hd_minus_5M,
       date_add(hiredate,interval  5 month) as hd_plus_5M,
       date_add(hiredate,interval -5 year)  as hd_minus_5Y,
       date_add(hiredate,interval  5 year)  as hd_plus_5DY
  from emp
 where deptno=10
```

SQL Server
Use the DATEADD function to add or subtract different units of time to/from a date:

```
select dateadd(day,-5,hiredate)   as hd_minus_5D,
       dateadd(day,5,hiredate)    as hd_plus_5D,
       dateadd(month,-5,hiredate) as hd_minus_5M,
       dateadd(month,5,hiredate)  as hd_plus_5M,
       dateadd(year,-5,hiredate)  as hd_minus_5Y,
       dateadd(year,5,hiredate)   as hd_plus_5Y
  from emp
 where deptno = 10
```

DISCUSSION

The Oracle solution takes advantage of the fact that integer values represent days when performing date arithmetic. However, that's true only of arithmetic with DATE types. Oracle9 i Database introduced TIMESTAMP types. For those, you should use the INTERVAL solution shown for PostgreSQL. Beware too, of passing TIMESTAMPs to old-style date functions such as ADD_MONTHS. By doing so, you can lose any fractional seconds that such TIMESTAMP values may contain.

The INTERVAL keyword and the string literals that go with it represent ISO-standard SQL syntax. The standard requires that interval values be enclosed within single quotes. PostgreSQL (and Oracle9 i

Database and later) complies with the standard. MySQL deviates somewhat by omitting support for the quotes.

8.2. Determining the Number of Days Between Two Dates

PROBLEM

You want to find the difference between two dates and represent the result in days. For example, you want to find the difference in days between the HIREDATEs of employee ALLEN and employee WARD.

SOLUTION

DB2
Use two inline views to find the HIREDATEs for WARD and ALLEN. Then subtract one HIREDATE from the other using the DAYS function:

```
select days(ward_hd) - days(allen_hd)
  from (
select hiredate as ward_hd
  from emp
 where ename = 'WARD'
       ) x,
       (
select hiredate as allen_hd
  from emp
 where ename = 'ALLEN'
       ) y
```

Oracle and PostgreSQL
Use two inline views to find the HIREDATEs for WARD and ALLEN, and then subtract one date from the other:

```
select ward_hd - allen_hd
  from (
select hiredate as ward_hd
  from emp
 where ename = 'WARD'
       ) x,
       (
select hiredate as allen_hd
  from emp
 where ename = 'ALLEN'
       ) y
```

MySQL and SQL Server
Use the function DATEDIFF to find the number of days between two dates. MySQL's version of DATEDIFF requires only two parameters (the two dates you want to find the difference in days between), and the smaller of the two dates should be passed first to avoid negative values (opposite in SQL Server). SQL Server's version of the function allows you to specify what you want the return value to represent (in this example you want to return the difference in days). The solution following uses the SQL Server version:

```
select datediff(day,allen_hd,ward_hd)
  from (
select hiredate as ward_hd
  from emp
 where ename = 'WARD'
       ) x,
       (
select hiredate as allen_hd
  from emp
 where ename = 'ALLEN'
       ) y
```

MySQL users can simply remove the first argument of the function and flip-flop the order in which ALLEN_HD and WARD_HD is passed.

DISCUSSION

For all solutions, inline views X and Y return the HIREDATEs for employees WARD and ALLEN respectively. For example:

```
select ward_hd, allen_hd
    from (
select hiredate as ward_hd
  from emp
 where ename = 'WARD'
       ) y,
       (
select hiredate as allen_hd
  from emp
 where ename = 'ALLEN'
       ) x

WARD_HD     ALLEN_HD
----------- ----------
22-FEB-1981 20-FEB-1981
```

You'll notice a Cartesian product is created, because there is no join specified between X and Y. In this case, the lack of a join is harmless as the cardinalities for X and Y are both 1, thus the result set will ultimately have one row (obviously, because 1x1=1). To get the difference in days, simply subtract one of the two values returned from the other using methods appropriate for your database.

8.3. Determining the Number of Business Days Between Two Dates

PROBLEM

Given two dates, you want to find how many "working" days are between them, including the two dates themselves. For example, if January 10th is a Tuesday and January 11th is a Monday, then the number of working days between these two dates is two, as both days are typical work days. For this recipe, "business days" is defined as any day that is not Saturday or Sunday.

SOLUTION

The solution examples find the number of business days between the HIREDATEs of BLAKE and JONES. To determine the number of business days between two dates, you can use a pivot table to return a row for each day between the two dates (including the start and end dates). Having done that, finding the number of business days is simply counting the dates returned that are not Saturday or Sunday.

TIP

If you want to exclude holidays as well, you can create a HOLIDAYS table. Then include a simple NOT IN predicate to exclude days listed in HOLIDAYS from the solution.

DB2

Use the pivot table T500 to generate the required number of rows (representing days) between the two dates. Then count each day that is not a weekend. Use the DAYNAME function to return the weekday name of each date. For example:

```
select sum(case when dayname(jones_hd+t500.id day -1 day)
                  in ( 'Saturday','Sunday' )
                then 0 else 1
           end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
           end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= blake_hd-jones_hd+1
```

MySQL

Use the pivot table T500 to generate the required number of rows (days) between the two dates. Then count each day that is not a weekend. Use the DATE_ADD function to add days to each date. Use the DATE_FORMAT function to obtain the weekday name of each date:

```
 select sum(case when date_format(
                         date_add(jones_hd,
                                  interval t500.id-1 DAY),'%a')
                  in ( 'Sat','Sun' )
                then 0 else 1
           end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
            end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= datediff(blake_hd,jones_hd)+1
```

Oracle

Use the pivot table T500 to generate the required number of rows (days) between the two dates, and then count each day that is not a weekend. Use the TO_CHAR function to obtain the weekday name of each date:

```
select sum(case when to_char(jones_hd+t500.id-1,'DY')
                  in ( 'SAT','SUN' )
                then 0 else 1
           end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
           end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= blake_hd-jones_hd+1
```

PostgreSQL

Use the pivot table T500 to generate the required number of rows (days) between the two dates. Then count each day that is not a weekend. Use the TO_CHAR function to obtain the weekday name of each date:

```
select sum(case when trim(to_char(jones_hd+t500.id-1,'DAY'))
                  in ( 'SATURDAY','SUNDAY' )
                then 0 else 1
           end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
           end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= blake_hd-jones_hd+1
```

SQL Server

Use the pivot table T500 to generate the required number of rows (days) between the two dates, and then count each day that is not a weekend. Use the DATENAME function to obtain the weekday name of each date:

```
select sum(case when datename(dw,jones_hd+t500.id-1)
                  in ( 'SATURDAY','SUNDAY' )
                 then 0 else 1
           end) as days
  from (
selectmax(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
           end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= datediff(day,jones_hd-blake_hd)+1
```

DISCUSSION

While each RDBMS requires the use of different built-in functions to determine the name of a day, the overall solution approach is the same for each. The solution can be broken into two steps:

Return the days between the start date and end date (inclusive).

Count how many days (i.e., rows) there are, excluding weekends.

Inline view X performs step 1. If you examine inline view X, you'll notice the use of the aggregate function MAX, which the recipe uses to remove NULLs. If the use of MAX is unclear, the following output might help you understand. The output shows the results from inline view X without MAX:

```
select case when ename = 'BLAKE'
            then hiredate
       end as blake_hd,
       case when ename = 'JONES'
            then hiredate
       end as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )

BLAKE_HD    JONES_HD
----------- -----------
            02-APR-1981
01-MAY-1981
```

Without MAX, two rows are returned. By using MAX you return only one row instead of two, and the NULLs are eliminated:

```
select max(case when ename = 'BLAKE'
                then hiredate
       end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
       end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )

BLAKE_HD    JONES_HD
----------- -----------
01-MAY-1981 02-APR-1981
```

The number of days (inclusive) between the two dates here is 30. Now that the two dates are in one row, the next step is to generate one row for each of those 30 days. To return the 30 days (rows), use table T500. Since each value for ID in table T500 is simply 1 greater than the one before it, add each row returned by T500 to the earlier of the two dates (JONES_HD) to generate consecutive days starting from JONES_HD up to and including BLAKE_HD. The result of this addition is shown below (using Oracle syntax):

```
select x.*, t500.*, jones_hd+t500.id-1
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
           end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= blake_hd-jones_hd+1
```

```
BLAKE_HD    JONES_HD            ID JONES_HD+T5
----------- ----------- ---------- -----------
01-MAY-1981 02-APR-1981          1 02-APR-1981
01-MAY-1981 02-APR-1981          2 03-APR-1981
01-MAY-1981 02-APR-1981          3 04-APR-1981
01-MAY-1981 02-APR-1981          4 05-APR-1981
01-MAY-1981 02-APR-1981          5 06-APR-1981
01-MAY-1981 02-APR-1981          6 07-APR-1981
01-MAY-1981 02-APR-1981          7 08-APR-1981
01-MAY-1981 02-APR-1981          8 09-APR-1981
01-MAY-1981 02-APR-1981          9 10-APR-1981
01-MAY-1981 02-APR-1981         10 11-APR-1981
01-MAY-1981 02-APR-1981         11 12-APR-1981
01-MAY-1981 02-APR-1981         12 13-APR-1981
01-MAY-1981 02-APR-1981         13 14-APR-1981
01-MAY-1981 02-APR-1981         14 15-APR-1981
01-MAY-1981 02-APR-1981         15 16-APR-1981
01-MAY-1981 02-APR-1981         16 17-APR-1981
01-MAY-1981 02-APR-1981         17 18-APR-1981
01-MAY-1981 02-APR-1981         18 19-APR-1981
01-MAY-1981 02-APR-1981         19 20-APR-1981
01-MAY-1981 02-APR-1981         20 21-APR-1981
01-MAY-1981 02-APR-1981         21 22-APR-1981
01-MAY-1981 02-APR-1981         22 23-APR-1981
01-MAY-1981 02-APR-1981         23 24-APR-1981
01-MAY-1981 02-APR-1981         24 25-APR-1981
01-MAY-1981 02-APR-1981         25 26-APR-1981
01-MAY-1981 02-APR-1981         26 27-APR-1981
01-MAY-1981 02-APR-1981         27 28-APR-1981
01-MAY-1981 02-APR-1981         28 29-APR-1981
01-MAY-1981 02-APR-1981         29 30-APR-1981
01-MAY-1981 02-APR-1981         30 01-MAY-1981
```

If you examine the WHERE clause, you’ll notice that you add 1 to the difference between BLAKE_HD and JONES_HD to generate the required 30 rows (otherwise, you would get 29 rows). You’ll also notice that you subtract 1 from T500.ID in the SELECT list of the outer query, since the values for ID start at 1 and adding 1 to JONES_HD would cause JONES_HD to be excluded from the final count.

Once you generate the number of rows required for the result set, use a CASE expression to “flag” whether or not each of the days returned are weekdays or weekends (return a 1 for a weekday and a 0 for a weekend). The final step is to use the aggregate function SUM to tally up the number of 1s to get the final answer.

8.4. Determining the Number of Months or Years Between Two Dates

PROBLEM

You want to find the difference between two dates in terms of either months or years. For example, you want to find the number of months between the first and last employees hired, and you also wish to express that value as some number of years.

SOLUTION

Since there are always 12 months in a year, you can find the number of months between two dates, and then divide by 12 to get the number of years. After getting comfortable with the solution, you’ll want to round the results up or down depending on what you want for the year. For example, the first HIREDATE in table EMP is “17-DEC-1980” and the last is “12-JAN-1983”. If you do the math on the years (1983 minus 1980) you get three years, yet the difference in months is approximately 25 (a little over two years). You should tweak the solution as you see fit. The solutions below will return 25 months and ~2 years.

DB2 and MySQL
Use the functions YEAR and MONTH to return the four-digit year and the two-digit month for the dates supplied:

```
select mnth, mnth/12
  from (
select (year(max_hd) - year(min_hd))*12 +
       (month(max_hd) - month(min_hd)) as mnth
  from (
select min(hiredate) as min_hd, max(hiredate) as max_hd
  from emp
       ) x
       ) y
```

Oracle
Use the function MONTHS_BETWEEN to find the difference between two dates in months (to get years, simply divide by 12):

```
select months_between(max_hd,min_hd),
       months_between(max_hd,min_hd)/12
  from (
select min(hiredate) min_hd, max(hiredate) max_hd
  from emp
       ) x
```

PostgreSQL
Use the function EXTRACT to return the four-digit year and two-digit month for the dates supplied:

```
select mnth, mnth/12
  from (
select ( extract(year from max_hd)
         extract(year from min_hd) ) * 12
       +
       ( extract(month from max_hd)
         extract(month from min_hd) ) as mnth
  from (
select min(hiredate) as min_hd, max(hiredate) as max_hd
  from emp
       ) x
       ) y
```

SQL Server
Use the function DATEDIFF to find the difference between two dates in months (to get years, simply divide by 12):

```
select datediff(month,min_hd,max_hd),
       datediff(month,min_hd,max_hd)/12
  from (
select min(hiredate) min_hd, max(hiredate) max_hd
  from emp
       ) x
```

DISCUSSION

DB2, MySQL, and PostgreSQL
Once you extract the year and month for MIN_HD and MAX_HD in the PostgreSQL solution, the method for finding the months and years between MIN_HD and MAX_HD is the same for all three

RDBMs. This discussion will cover all three solutions. Inline view X returns the earliest and latest HIREDATEs in table EMP and can be seen below:

```
select min(hiredate) as min_hd,
       max(hiredate) as max_hd
  from emp

MIN_HD      MAX_HD
----------- -----------
17-DEC-1980 12-JAN-1983
```

To find the months between MAX_HD and MIN_HD, multiply the difference in years between MIN_HD and MAX_HD by 12, then add the difference in months between MAX_HD and MIN_HD. If you are having trouble seeing how this works, return the date component for each date. The numeric values for the years and months are show below:

```
select year(max_hd) as max_yr, year(min_hd) as min_yr,
       month(max_hd) as max_mon, month(min_hd) as min_mon
  from (
select min(hiredate) as min_hd, max(hiredate) as max_hd
  from emp
       ) x

MAX_YR     MIN_YR    MAX_MON    MIN_MON
------ ---------- ---------- ----------
  1983       1980          1         12
```

Looking at the results above, finding the months between MAX_HD and MIN_HD is simply (1983–1980)*12 + (1–12). To find the number of years between MIN_HD and MAX_HD, divide the number of months by 12. Again, depending on the results you are looking for you will want to round the values.

Oracle and SQL Server
Inline view X returns the earliest and latest HIREDATEs in table EMP and can be seen below:

```
select min(hiredate) as min_hd, max(hiredate) as max_hd
  from emp

MIN_HD      MAX_HD
----------- -----------
17-DEC-1980 12-JAN-1983
```

The functions supplied by Oracle and SQL Server (MONTHS_BETWEEN and DATEDIFF, respectively) will return the number of months between two given dates. To find the year, divide the number of months by 12.

8.5. Determining the Number of Seconds, Minutes, or Hours Between Two Dates

PROBLEM

You want to return the difference in seconds between two dates. For example, you want to return the difference between the HIREDATEs of ALLEN and WARD in seconds, minutes, and hours.

SOLUTION

If you can find the number of days between two dates, you can find seconds, minutes, and hours as they are the units of time that make up a day.

DB2
Use the function DAYS to find the difference between ALLEN_HD and WARD_HD in days. Then multiply to find each unit of time:

```
select dy*24 hr, dy*24*60 min, dy*24*60*60 sec
  from (
select ( days(max(case when ename = 'WARD'
                  then hiredate
             end)) -
         days(max(case when ename = 'ALLEN'
                  then hiredate
             end))
       ) as dy
  from emp
       ) x
```

MySQL and SQL Server
Use the DATEDIFF function to return the number of days between ALLEN_HD and WARD_HD. Then multiply to find each unit of time:

```
select datediff(day,allen_hd,ward_hd)*24 hr,
       datediff(day,allen_hd,ward_hd)*24*60 min,
       datediff(day,allen_hd,ward_hd)*24*60*60 sec
  from (
select max(case when ename = 'WARD'
                 then hiredate
           end) as ward_hd,
       max(case when ename = 'ALLEN'
                then hiredate
           end) as allen_hd
  from emp
       ) x
```

Oracle and PostgreSQL

Use subtraction to return the number of days between ALLEN_HD and WARD_ HD. Then multiply to find each unit of time:

```
select dy*24 as hr, dy*24*60 as min, dy*24*60*60 as sec
  from (
select (max(case when ename = 'WARD'
                then hiredate
           end) -
       max(case when ename = 'ALLEN'
                then hiredate
           end)) as dy
  from emp
       ) x
```

DISCUSSION

Inline view X for all solutions returns the HIREDATEs for WARD and ALLEN, as can be seen below:

```
select max(case when ename = 'WARD'
                 then hiredate
           end) as ward_hd,
       max(case when ename = 'ALLEN'
                then hiredate
           end) as allen_hd
  from emp

WARD_HD     ALLEN_HD
----------- -----------
22-FEB-1981 20-FEB-1981
```

Multiply the number of days between WARD_HD and ALLEN_HD by 24 (hours in a day), 1440 (minutes in a day), and 86400 (seconds in a day).

8.6. Counting the Occurrences of Weekdays in a Year

PROBLEM

You want to count the number of times each weekday occurs in one year.

SOLUTION

To find the number of occurrences of each weekday in a year, you must:

Generate all possible dates in the year.

Format the dates such that they resolve to the name of their respective weekdays.

Count the occurrence of each weekday name.

DB2
Use recursive WITH to avoid the need to SELECT against a table with at least 366 rows. Use the function DAYNAME to obtain the weekday name for each date, and then count the occurrence of each:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (current_date
        dayofyear(current_date) day)
        +1 day as start_date
  from t1
       ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
)
select dayname(start_date),count(*)
  from x
 group by dayname(start_date)
```

MySQL
Select against table T500 to generate enough rows to return every day in the year. Use the DATE_FORMAT function to obtain the weekday name of each date, and then count the occurrence of each name:

```
select date_format(
          date_add(
              cast(
            concat(year(current_date),'-01-01')
                   as date),
                   interval t500.id-1 day),
                   '%W') day,
       count(*)
  from t500
 where t500.id <= datediff(
                        cast(
                      concat(year(current_date)+1,'-01-01')
                             as date),
                        cast(
                      concat(year(current_date),'-01-01')
                             as date))
group by date_format(
            date_add(
                cast(
              concat(year(current_date),'-01-01')
                     as date),
                     interval t500.id-1 day),
                     '%W')
```

Oracle
If you are on Oracle9 i Database or later, you can use the recursive CONNECT BY to return each day in a year. If you are on Oracle8 i Database or earlier, select against table T500 to generate enough rows to return every day in a year. In either case, use the TO_CHAR function to obtain the weekday name of each date, and then count the occurrence of each name.

First, the CONNECT BY solution:

```
with x as (
select level lvl
  from dual
 connect by level <= (
   add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
 )
)
select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
  from x
 group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')
```

and next, the solution for older releases of Oracle:

```
select to_char(trunc(sysdate,'y')+rownum-1,'DAY'),
       count(*)
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                     - trunc(sysdate,'y'))
 group by to_char(trunc(sysdate,'y')+rownum-1,'DAY')
```

PostgreSQL

Use the built-in function GENERATE_SERIES to generate one rows for every day in the year. Then use the TO_CHAR function to obtain the weekday name of each date. Finally, count the occurrence of each weekday name. For example:

```
select to_char(
          cast(
    date_trunc('year',current_date)
               as date) + gs.id-1,'DAY'),
       count(*)
  from generate_series(1,366) gs(id)
 where gs.id <= (cast
                    ( date_trunc('year',current_date) +
                         interval '12 month' as date) -
cast(date_trunc('year',current_date)
                      as date))
 group by to_char(
             cast(
       date_trunc('year',current_date)
          as date) + gs.id-1,'DAY')
```

SQL Server

Use the recursive WITH to avoid the need to SELECT against a table with at least 366 rows. If you are on a version of SQL Server that does not support the WITH clause, see the alternative Oracle solution as a guideline for using a pivot table. Use the DATENAME function to obtain the weekday name of each date, and then count the occurrence of each name. For example:

```
with x (start_date,end_date)
as (
select start_date,
       dateadd(year,1,start_date) end_date
  from (
select cast(
       cast(year(getdate()) as varchar) + '-01-01'
            as datetime) start_date
  from t1
       ) tmp
union all
select dateadd(day,1,start_date), end_date
  from x
 where dateadd(day,1,start_date) < end_date
)
select datename(dw,start_date),count(*)
  from x
 group by datename(dw,start_date)
OPTION (MAXRECURSION 366)
```

DISCUSSION

DB2

Inline view TMP, in the recursive WITH view X, returns the first day of the current year and is shown below:

```
select (current_date
        dayofyear(current_date) day)
        +1 day as start_date
  from t1

START_DATE
-------------
01-JAN-2005
```

The next step is to add one year to START_DATE, so that you have the beginning and end dates. You need to know both because you want to generate every day in a year. START_DATE and END_DATE are shown below:

```
select start_date,
       start_date + 1 year end_date
  from (
select (current_date
        dayofyear(current_date) day)
        +1 day as start_date
  from t1
       ) tmp

START_DATE  END_DATE
----------- ------------
01-JAN-2005 01-JAN-2006
```

The next step is to recursively increment START_DATE by one day, stopping before it equals END_DATE. A portion of the rows returned by the recursive view X is shown below:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (current_date -
        dayofyear(current_date) day)
        +1 day as start_date
  from t1
       ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
)
select * from x

START_DATE  END_DATE
----------- -----------
01-JAN-2005 01-JAN-2006
02-JAN-2005 01-JAN-2006
03-JAN-2005 01-JAN-2006
…
29-JAN-2005 01-JAN-2006
30-JAN-2005 01-JAN-2006
31-JAN-2005 01-JAN-2006
…
01-DEC-2005 01-JAN-2006
02-DEC-2005 01-JAN-2006
03-DEC-2005 01-JAN-2006
…
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
31-DEC-2005 01-JAN-2006
```

The final step is to use the function DAYNAME on the rows returned by the recursive view X, and count how many times each weekday occurs. The final result is shown below:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (
                current_date -
        dayofyear(current_date) day)
        +1 day as start_date
  from t1
       ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
)
select dayname(start_date),count(*)
  from x
 group by dayname(start_date)

START_DATE   COUNT(*)
---------- ----------
FRIDAY             52
MONDAY             52
SATURDAY           53
SUNDAY             52
THURSDAY           52
TUESDAY            52
WEDNESDAY          52
```

MySQL

This solution selects against table T500 to generate one row for every day in the year. The command on line 4 returns the first day of the current year. It does this by returning the year of the date returned by the function CURRENT_DATE, and then appending a month and day (following MySQL's default date format). The result is shown below:

```
select concat(year(current_date),'-01-01')
  from t1

START_DATE
-----------
01-JAN-2005
```

Now that you have the first day in the current year, use the DATEADD function to add each value from T500.IDto generate each day in the year. Use the function DATE_FORMAT to return the weekday for each date. To generate the required number of rows from table T500, find the difference in days between the first day of the current year and the first day of the next year, and return that many rows (will be either 365 or 366). A portion of the results is shown below:

```
select date_format(
          date_add(
              cast(
            concat(year(current_date),'-01-01')
                   as date),
                   interval t500.id-1 day),
                   '%W') day
  from t500
 where t500.id <= datediff(
                         cast(
                       concat(year(current_date)+1,'-01-01')
                              as date),
                         cast(
                       concat(year(current_date),'-01-01')
                             as date))

DAY
-----------
01-JAN-2005
02-JAN-2005
03-JAN-2005
…
29-JAN-2005
30-JAN-2005
31-JAN-2005
…
01-DEC-2005
02-DEC-2005
03-DEC-2005
…
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

Now that you can return every day in the current year, count the occurrences of each weekday returned by the function DAYNAME. The final results are shown below:

```
select date_format(
          date_add(
              cast(
            concat(year(current_date),'-01-01')
                   as date),
                   interval t500.id-1 day),
                   '%W') day,
       count(*)
  from t500
 where t500.id <= datediff(
                        cast(
                      concat(year(current_date)+1,'-01-01')
                             as date),
                        cast(
                      concat(year(current_date),'-01-01')
                             as date))
 group by date_format(
             date_add(
                 cast(
               concat(year(current_date),'-01-01')
                      as date),
                     interval t500.id-1 day),
                     '%W')

DAY         COUNT(*)
--------- ----------
FRIDAY            52
MONDAY            52
SATURDAY          53
SUNDAY            52
THURSDAY          52
TUESDAY           52
WEDNESDAY         52
```

Oracle

The solutions provided either select against table T500 (a pivot table), or use the recursive CONNECT BY and WITH, to generate a row for every day in the current year. The call to the function TRUNC truncates the current date to the first day of the current year.

If you are using the CONNECT BY/WITH solution, you can use the pseudo-column LEVEL to generate sequential numbers beginning at 1. To generate the required number of rows needed for this solution, filter ROWNUM or LEVEL on the difference in days between the first day of the current year and the first day of the next year (will be 365 or 366 days). The next step is to increment each day by adding ROWNUM or LEVEL to the first day of the current year. Partial results are shown below:

```
/* Oracle 9i and later */
with x as (
select level lvl
  from dual
 connect by level <= (
   add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
 )
)
select trunc(sysdate,'y')+lvl-1   from x
```

If you are using the pivot-table solution, you can use any table or view with at least 366 rows in it. And since Oracle has ROWNUM, there's no need for a table with incrementing values starting from 1. Consider the following example, which uses pivot table T500 to return every day in the current year:

```
/* Oracle 8i and earlier */
select trunc(sysdate,'y')+rownum-1 start_date
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                     - trunc(sysdate,'y'))

START_DATE
-----------
01-JAN-2005
02-JAN-2005
03-JAN-2005
…
29-JAN-2005
30-JAN-2005
31-JAN-2005
…
01-DEC-2005
02-DEC-2005
03-DEC-2005
…
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

Regardless of which approach you take, you eventually must use the function TO_ CHAR to return the weekday name for each date, and then count the occurrence of each name. The final results are shown below:

```
/* Oracle 9i and later */
with x as (
select level lvl
  from dual
 connect by level <= (
   add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
 )
)
select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
  from x
 group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')

/* Oracle 8i and earlier */
select to_char(trunc(sysdate,'y')+rownum-1,'DAY') start_date,
       count(*)
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                    - trunc(sysdate,'y'))
 group by to_char(trunc(sysdate,'y')+rownum-1,'DAY')

START_DATE   COUNT(*)
---------- ----------
FRIDAY             52
MONDAY             52
SATURDAY           53
SUNDAY             52
THURSDAY           52
TUESDAY            52
WEDNESDAY          52
```

PostgreSQL

The first step is to use the DATE_TRUNC function to return the year of the current date (shown below, selecting against T1 so only one row is returned):

```
select cast(
         date_trunc('year',current_date)
         as date) as start_date
  from t1

 START_DATE
 ----------
 01-JAN-2005
```

The next step is to select against a row source (any table expression, really) with at least 366 rows. The solution uses the function GENERATE_SERIES as the row source. You can, of course, use table T500 instead. Then add one day to the first day of the current year until you return every day in the year (shown below):

```
select cast( date_trunc('year',current_date)
               as date) + gs.id-1 as start_date
  from generate_series (1,366) gs(id)
 where gs.id <= (cast
                     ( date_trunc('year',current_date) +
                          interval '12 month' as date) -
     cast(date_trunc('year',current_date)
                      as date))

START_DATE
-----------
01-JAN-2005
02-JAN-2005
03-JAN-2005
…
29-JAN-2005
30-JAN-2005
31-JAN-2005
…
01-DEC-2005
02-DEC-2005
03-DEC-2005
…
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

The final step is to use the function TO_CHAR to return the weekday name for each date, and then count the occurrence of each name. The final results are shown below:

```
select to_char(
          cast(
    date_trunc('year',current_date)
                as date) + gs.id-1,'DAY') as start_dates,
       count(*)
  from generate_series(1,366) gs(id)
 where gs.id <= (cast
                     ( date_trunc('year',current_date) +
                        interval '12 month' as date) -
     cast(date_trunc('year',current_date)
                      as date))
 group by to_char(
             cast(
       date_trunc('year',current_date)
          as date) + gs.id-1,'DAY')

START_DATE   COUNT(*)
---------- ----------
FRIDAY             52
MONDAY             52
SATURDAY           53
SUNDAY             52
THURSDAY           52
TUESDAY            52
WEDNESDAY          52
```

SQL Server

Inline view TMP, in the recursive WITH view X, returns the first day of the current year and is shown below:

```
select cast(
       cast(year(getdate()) as varchar) + '-01-01'
            as datetime) start_date
  from t1

START_DATE
-----------
01-JAN-2005
```

Once you return the first day of the current year, add one year to START_DATE so that you have the beginning and end dates. You need to know both because you want to generate every day in a year. START_DATE and END_DATE are shown below:

```
select start_date,
        dateadd(year,1,start_date) end_date
  from (
select cast(
       cast(year(getdate()) as varchar) + '-01-01'
            as datetime) start_date
  from t1
       ) tmp

START_DATE  END_DATE
----------- -----------
01-JAN-2005 01-JAN-2006
```

Next, recursively increment START_DATE by one day and stop before it equals END_DATE. A portion of the rows returned by the recursive view X is shown below:

```
with x (start_date,end_date)
 as (
 select start_date,
        dateadd(year,1,start_date) end_date
   from (
 select cast(
        cast(year(getdate()) as varchar) + '-01-01'
             as datetime) start_date
   from t1
        ) tmp
 union all
 select dateadd(day,1,start_date), end_date
   from x
  where dateadd(day,1,start_date) < end_date
 )
 select * from x
 OPTION (MAXRECURSION 366)

START_DATE  END_DATE
----------- -----------
01-JAN-2005 01-JAN-2006
02-JAN-2005 01-JAN-2006
03-JAN-2005 01-JAN-2006
…
29-JAN-2005 01-JAN-2006
30-JAN-2005 01-JAN-2006
31-JAN-2005 01-JAN-2006
…
01-DEC-2005 01-JAN-2006
02-DEC-2005 01-JAN-2006
03-DEC-2005 01-JAN-2006
…
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
31-DEC-2005 01-JAN-2006
```

The final step is to use the function DATENAME on the rows returned by the recursive view X and count how many times each weekday occurs. The final result is shown below:

```
with x(start_date,end_date)
 as (
 select start_date,
        dateadd(year,1,start_date) end_date
   from (
 select cast(
        cast(year(getdate()) as varchar) + '-01-01'
             as datetime) start_date
   from t1
        ) tmp
 union all
 select dateadd(day,1,start_date), end_date
   from x
  where dateadd(day,1,start_date) < end_date
 )
 select datename(dw,start_date), count(*)
   from x
  group by datename(dw,start_date)
 OPTION (MAXRECURSION 366)
```

```
START_DATE   COUNT(*)
---------  ----------
FRIDAY             52
MONDAY             52
SATURDAY           53
SUNDAY             52
THURSDAY           52
TUESDAY            52
WEDNESDAY          52
```

8.7. Determining the Date Difference Between the Current Record and the Next Record

PROBLEM

You want to determine the difference in days between two dates (specifically dates stored in two different rows). For example, for every employee in DEPTNO 10, you want to determine the number of days between the day they were hired and the day the next employee (can be in another department) was hired.

SOLUTION

The trick to this problem's solution is to find the earliest HIREDATE after the current employee was hired. After that, simply use the technique from "Determining the Number of Days between Two

Dates” to find the difference in days.

DB2

Use a scalar subquery to find the next HIREDATE relative to the current HIREDATE. Then use the DAYS function to find the difference in days:

```
select x.*,
       days(x.next_hd) - days(x.hiredate) diff
  from (
select e.deptno, e.ename, e.hiredate,
       (select min(d.hiredate) from emp d
         where d.hiredate > e.hiredate) next_hd
  from emp e
 where e.deptno = 10
       ) x
```

MySQL and SQL Server

Use a scalar subquery to find the next HIREDATE relative to the current HIREDATE. Then use the DATEDIFF function to find the difference in days. The SQL Server version of DATEDIFF is used below:

```
select x.*,
       datediff(day,x.hiredate,x.next_hd) diff
  from (
select e.deptno, e.ename, e.hiredate,
       (select min(d.hiredate) from emp d
         where d.hiredate > e.hiredate) next_hd
  from emp e
 where e.deptno = 10
       ) x
```

MySQL users can exclude the first argument (“day”) and switch the order of the two remaining arguments:

```
2         datediff(x.next_hd, x.hiredate) diff
```

Oracle

If you’re on Oracle8 i Database or later, use the window function LEAD OVER to access the next HIREDATE relative to the current row, thus facilitating subtraction:

```
select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate)over(order by hiredate) next_hd
  from emp
       )
 where deptno=10
```

If you are on Oracle8 Database or earlier, you can use the PostgreSQL solution as an alternative.

PostgreSQL
Use a scalar subquery to find the next HIREDATE relative to the current HIREDATE. Then use simple subtraction to find the difference in days:

```
select x.*,
       x.next_hd - x.hiredate as diff
  from (
select e.deptno, e.ename, e.hiredate,
       (select min(d.hiredate) from emp d
         where d.hiredate > e.hiredate) as next_hd
  from emp e
 where e.deptno = 10
       ) x
```

DISCUSSION

DB2, MySQL, PostgreSQL, and SQL Server
Despite the differences in syntax, the approach is the same for all these solutions: use a scalar subquery to find the next HIREDATE relative to the current HIREDATE, and then find the difference in days between the two using the technique described in “Determining the Number of Days Between Two Dates,” found earlier in this Lab.

Oracle
The window function LEAD OVER is extremely useful here as it allows you to access “future” rows (“future” determined by the ORDER BY clause, relative to the current row). The ability to access rows around your current row without additional joins provides for more readable and efficient code. When working with window functions, keep in mind that they are evaluated after the WHERE clause, hence the need for an inline view in the solution. If you were to move the filter on DEPTNO into the inline view, the results would change (only the HIREDATEs from DEPTNO 10 would be considered). One important note to mention about Oracle’s LEAD and LAG functions is their behavior in the presence of duplicates. In the preface I mention that these recipes are not coded “defensively” because there are too many conditions that one can’t possibly foresee that can break code. Or, even if one can

foresee every problem, sometimes the resulting SQL becomes unreadable. So in most cases, the goal of a solution is to introduce a technique: one that you can use in your production system, but that must be tested and many times tweaked to work for your particular data. In this case, though, there is a situation that I will discuss simply because the workaround may not be all that obvious, particularly for those coming from non-Oracle systems. In this example there are no duplicate HIREDATEs in table EMP, but it is certainly possible (and probably likely) that there are duplicate date values in your tables. Consider the employees in DEPTNO 10 and their HIREDATEs:

```
select ename, hiredate
  from emp
 where deptno=10
 order by 2

ENAME  HIREDATE
------ -----------
CLARK  09-JUN-1981
KING   17-NOV-1981
MILLER 23-JAN-1982
```

For the sake of this example, let’s insert four duplicates such that there are five employees (including KING) hired on November 17:

```
insert into emp (empno,ename,deptno,hiredate)
values (1,'ant',10,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,hiredate)
values (2,'joe',10,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,hiredate)
values (3,'jim',10,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,hiredate)
values (4,'choi',10,to_date('17-NOV-1981'))

select ename, hiredate
  from emp
 where deptno=10
 order by 2

ENAME  HIREDATE
------ -----------
CLARK  09-JUN-1981
ant    17-NOV-1981
joe    17-NOV-1981
KING   17-NOV-1981
jim    17-NOV-1981
choi   17-NOV-1981
MILLER 23-JAN-1982
```

Now there are multiple employees in DEPTNO 10 hired on the same day. If you try to use the proposed solution (moving the filter into the inline view so you only are concerned with employees in DEPTNO 10 and their HIREDATEs) on this result set you get the following output:

```
select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate)over(order by hiredate) next_hd
  from emp
 where deptno=10
        )

ENAME  HIREDATE    NEXT_HD           DIFF
------ ----------- ----------- ----------
CLARK  09-JUN-1981 17-NOV-1981        161
ant    17-NOV-1981 17-NOV-1981          0
joe    17-NOV-1981 17-NOV-1981          0
KING   17-NOV-1981 17-NOV-1981          0
jim    17-NOV-1981 17-NOV-1981          0
choi   17-NOV-1981 23-JAN-1982         67
MILLER 23-JAN-1982 (null)          (null)
```

Looking at the values of DIFF for four of the five employees hired on the same day, you can see that the value is zero. This is not correct. All employees hired on the same day should have their dates evaluated against the HIREDATE of the next date on which an employee was hired, i.e., all employees hired on November 17 should be evaluated against MILLER's HIREDATE. The problem here is that the LEAD function orders the rows by HIREDATE but does not skip duplicates. So, for example, when employee ANT's HIREDATE is evaluated against employee JOE's HIREDATE, the difference is zero, hence a DIFF value of zero for ANT. Fortunately, Oracle has provided an easy workaround for situations like this one. When invoking the LEAD function, you can pass an argument to LEAD to specify exactly where the future row is (i.e., is it the next row, 10 rows later, etc.). So, looking at employee ANT, instead of looking ahead one row you need to look ahead five rows (you want to jump over all the other duplicates), because that's where MILLER is. If you look at employee JOE, he is four rows from MILLER, JIM is three rows from MILLER, KING is two rows from MILLER and, pretty boy CHOI is one row from MILLER. To get the correct answer, simply pass the distance from each employee to MILLER as an argument to LEAD. The solution is shown below:

```
select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
  from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
        )
        )

ENAME  HIREDATE    NEXT_HD           DIFF
------ ----------- ----------- ----------
CLARK  09-JUN-1981 17-NOV-1981        161
ant    17-NOV-1981 23-JAN-1982         67
joe    17-NOV-1981 23-JAN-1982         67
jim    17-NOV-1981 23-JAN-1982         67
choi   17-NOV-1981 23-JAN-1982         67
KING   17-NOV-1981 23-JAN-1982         67
MILLER 23-JAN-1982 (null)          (null)
```

Now the results are correct. All the employees hired on the same day have their HIREDATEs evaluated against the next HIREDATE, not a HIREDATE that matches their own. If the workaround isn't immediately obvious, simply break down the query. Start with the inline view:

```
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10

DEPTNO ENAME  HIREDATE           CNT         RN
------ ------ ----------- ---------- ----------
    10 CLARK  09-JUN-1981          1          1
    10 ant    17-NOV-1981          5          1
    10 joe    17-NOV-1981          5          2
    10 jim    17-NOV-1981          5          3
    10 choi   17-NOV-1981          5          4
    10 KING   17-NOV-1981          5          5
    10 MILLER 23-JAN-1982          1          1
```

The window function COUNT OVER counts the number of times each HIREDATE occurs and returns this value to each row. For the duplicate HIREDATEs, a value of 5 is returned for each row with that HIREDATE. The window function ROW_ NUMBER OVER ranks each employee by EMPNO. The ranking is partitioned by HIREDATE, so unless there are duplicate HIREDATEs each employee will have a rank of 1. At this point, all the duplicates have been counted and ranked and the ranking can serve as the distance to the next HIREDATE (MILLER's HIREDATE). You can see this by subtracting RN from CNT and adding 1 for each row when calling LEAD:

```
select deptno, ename, hiredate,
       cnt-rn+1 distance_to_miller,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
  from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
        )

DEPTNO ENAME  HIREDATE    DISTANCE_TO_MILLER NEXT_HD
------ ------ ----------- ------------------ -----------
    10 CLARK  09-JUN-1981                  1 17-NOV-1981
    10 ant    17-NOV-1981                  5 23-JAN-1982
    10 joe    17-NOV-1981                  4 23-JAN-1982
    10 jim    17-NOV-1981                  3 23-JAN-1982
    10 choi   17-NOV-1981                  2 23-JAN-1982
    10 KING   17-NOV-1981                  1 23-JAN-1982
    10 MILLER 23-JAN-1982                  1 (null)
```

As you can see, by passing the appropriate distance to jump ahead to, the LEAD function performs the subtraction on the correct dates.