Lab 6. Working with Strings

This lab focuses on string manipulation in SQL. Keep in mind that SQL is not designed to perform complex string manipulation and you can (and will) find working with strings in SQL to be very cumbersome and frustrating at times. Despite SQL's limitations, there are some very useful built-in functions provided by the different DBMSs, and I've tried to use them in creative ways. This lab in particular is very representative of the message I tried to convey in the introduction; SQL is the good, the bad, and the ugly. I hope that you take away from this lab a better appreciation for what can and can't be done in SQL when working with strings. In many cases you'll be surprised by how easy parsing and transforming of strings can be, while at other times you'll be aghast by the kind of SQL that is necessary to accomplish a particular task.

The first recipe in this lab is critically important, as it is leveraged by several of the subsequent solutions. In many cases, you'd like to have the ability to traverse a string by moving through it a character at a time. Unfortunately, SQL does not make this easy. Because there is no loop functionality in SQL (Oracle's MODEL clause excluded), you need to mimic a loop to traverse a string. I call this operation "walking a string" or "walking through a string" and the very first recipe explains the technique. This is a fundamental operation in string parsing when using SQL, and is referenced and used by almost all recipes in this lab. I strongly suggest becoming comfortable with how the technique works.

6.1. Walking a String

PROBLEM

You want to traverse a string to return each character as a row, but SQL lacks a loop operation. For example, you want to display the ENAME "KING" from table EMP as four rows, where each row contains just characters from "KING".

SOLUTION

Use a Cartesian product to generate the number of rows needed to return each character of a string on its own line. Then use your DBMS's built-in string parsing function to extract the characters you are interested in (SQL Server users will use SUBSTRING instead of SUBSTR and DATALENGTH instead of LENGTH):

```
 select substr(e.ename,iter.pos,1) as C
   from (select ename from emp where ename = 'KING') e,
        (select id as pos from t10) iter
  where iter.pos <= length(e.ename)

C
-
K
I
N
G
```

DISCUSSION

The key to iterating through a string's characters is to join against a table that has enough rows to produce the required number of iterations. This example uses table T10, which contains 10 rows (it has one column, ID, holding the values 1 through 10). The maximum number of rows that can be returned from this query is 10.

The following example shows the Cartesian product between E and ITER (i.e., between the specific name and the 10 rows from T10) without parsing ENAME:

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter

ENAME             POS
---------- ----------
KING                1
KING                2
KING                3
KING                4
KING                5
KING                6
KING                7
KING                8
KING                9
KING               10
```

The cardinality of inline view E is 1, and the cardinality of inline view ITER is 10. The Cartesian product is then 10 rows. Generating such a product is the first step in mimicking a loop in SQL.

TIP

It is common practice to refer to table T10 as a "pivot" table.

The solution uses a WHERE clause to break out of the loop after four rows have been returned. To restrict the result set to the same number of rows as there are characters in the name, that WHERE clause specifies ITER.POS <= LENGTH(E. ENAME) as the condition:

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter
 where iter.pos <= length(e.ename)

ENAME             POS
---------- ----------
KING                1
KING                2
KING                3
KING                4
```

Now that you have one row for each character in E.ENAME, you can use ITER.POS as a parameter to SUBSTR, allowing you to navigate through the characters in the string. ITER.POS increments with each row, and thus each row can be made to return a successive character from E.ENAME. This is how the solution example works.

Depending on what you are trying to accomplish you may or may not need to generate a row for every single character in a string. The following query is an example of walking E.ENAME and exposing different portions (more than a single character) of the string:

```
select substr(e.ename,iter.pos) a,
       substr(e.ename,length(e.ename)-iter.pos+1) b
  from (select ename from emp where ename = 'KING') e,
       (select id pos from t10) iter
 where iter.pos <= length(e.ename)

A          B
---------- ------
KING       G
ING        NG
NG         ING
G          KING
```

The most common scenarios for the recipes in this lab involve walking the whole string to generate a row for each character in the string, or walking the string such that the number of rows generated reflects the number of particular characters or delimiters that are present in the string.

6.2. Embedding Quotes Within String Literals

PROBLEM

You want to embed quote marks within string literals. You would like to produce results such as the following with SQL:

```
QMARKS
--------------
g'day mate
beavers' teeth
'
```

SOLUTION

The following three SELECTs highlight different ways you can create quotes: in the middle of a string and by themselves:

```
select 'g''day mate' qmarks from t1 union all
select 'beavers'' teeth'    from t1 union all
select ''''                 from t1
```

DISCUSSION

When working with quotes, it's often useful to think of them like parentheses. When you have an opening parenthesis, you must always have a closing parenthesis. The same goes for quotes. Keep in mind that you should always have an even number of quotes across any given string. To embed a single quote within a string you need to use two quotes:

```
select 'apples core', 'apple''s core',
        case when '' is null then 0 else 1 end
  from t1

'APPLESCORE 'APPLE''SCOR CASEWHEN''ISNULLTHEN0ELSE1END
----------- ------------ ----------------------------
apples core apple's core                            0
```

Following is the solution stripped down to its bare elements. You have two outer quotes defining a string literal, and, within that string literal you have two quotes that together represent just one quote in the string that you actually get:

```
select '''' as quote from t1

Q
-
'
```

When working with quotes, be sure to remember that a string literal comprising two quotes alone, with no intervening characters, is NULL.

6.3. Counting the Occurrences of a Character in a String

PROBLEM

You want to count the number of times a character or substring occurs within a given string. Consider the following string:

```
10,CLARK,MANAGER
```

You want to determine how many commas are in the string.

SOLUTION

Subtract the length of the string without the commas from the original length of the string to determine the number of commas in the string. Each DBMS provides functions for obtaining the length of a string and removing characters from a string. In most cases, these functions are LENGTH and REPLACE, respectively (SQL Server users will use the built-in function LEN rather than LENGTH):

```
select (length('10,CLARK,MANAGER')-
       length(replace('10,CLARK,MANAGER',',','')))/length(',')
       as cnt
  from t1
```

DISCUSSION

You arrive at the solution by using simple subtraction. The call to LENGTH on line 1 returns the original size of the string, and the first call to LENGTH on line 2 returns the size of the string without the commas, which are removed by REPLACE.

By subtracting the two lengths you obtain the difference in terms of characters, which is the number of commas in the string. The last operation divides the difference by the length of your search string.

This division is necessary if the string you are looking for has a length greater than 1. In the following example, counting the occurrence of “LL” in the string “HELLO HELLO” without dividing will return an incorrect result:

```
select
       (length('HELLO HELLO')-
       length(replace('HELLO HELLO','LL','')))/length('LL')
       as correct_cnt,
       (length('HELLO HELLO')-
       length(replace('HELLO HELLO','LL',''))) as incorrect_cnt
  from t1

CORRECT_CNT INCORRECT_CNT
----------- -------------
          2             4
```

6.4. Removing Unwanted Characters from a String

PROBLEM

You want to remove specific characters from your data. Consider this result set:

```
ENAME             SAL
---------- ----------
SMITH              800
ALLEN             1600
WARD              1250
JONES             2975
MARTIN            1250
BLAKE             2850
CLARK             2450
SCOTT             3000
KING              5000
TURNER            1500
ADAMS             1100
JAMES              950
FORD              3000
MILLER            1300
```

You want to remove all zeros and vowels as shown by the following values in columns STRIPPED1 and STRIPPED2:

```
ENAME      STRIPPED1         SAL STRIPPED2
---------- ---------- ---------- ---------
SMITH      SMTH              800 8
ALLEN      LLN              1600 16
WARD       WRD              1250 125
JONES      JNS              2975 2975
MARTIN     MRTN             1250 125
BLAKE      BLK              2850 285
CLARK      CLRK             2450 245
SCOTT      SCTT             3000 3
KING       KNG              5000 5
TURNER     TRNR             1500 15
ADAMS      DMS              1100 11
JAMES      JMS               950 95
FORD       FRD              3000 3
MILLER     MLLR             1300 13
```

SOLUTION

Each DBMS provides functions for removing unwanted characters from a string. The functions REPLACE and TRANSLATE are most useful for this problem.

DB2

Use the built-in functions TRANSLATE and REPLACE to remove unwanted characters and strings:

```
select ename,
       replace(translate(ename,'aaaaa','AEIOU'),'a','') stripped1,
       sal,
       replace(cast(sal as char(4)),'0','') stripped2
  from emp
```

MySQL and SQL Server

MySQL and SQL Server do not offer a TRANSLATE function, so several calls to REPLACE are needed:

```
select ename,
       replace(
       replace(
       replace(
       replace(
       replace(ename,'A',''),'E',''),'I',''),'O',''),'U','')
       as stripped1,
       sal,
       replace(sal,0,'') stripped2
  from emp
```

Oracle and PostgreSQL
Use the built-in functions TRANSLATE and REPLACE to remove unwanted characters and strings:

```
select ename,
       replace(translate(ename,'AEIOU','aaaaa'),'a')
       as stripped1,
       sal,
       replace(sal,0,'') as stripped2
  from emp
```

DISCUSSION

The built-in function REPLACE removes all occurrences of zeros. To remove the vowels, use TRANSLATE to convert all vowels into one specific character (I used “a”; you can use any character), then use REPLACE to remove all occurrences of that character.

6.5. Separating Numeric and Character Data

PROBLEM

You have (unfortunately) stored numeric data along with character data together in one column. You want to separate the character data from the numeric data. Consider the following result set:

```
DATA
---------------
SMITH800
ALLEN1600
WARD1250
JONES2975
MARTIN1250
BLAKE2850
CLARK2450
SCOTT3000
KING5000
TURNER1500
ADAMS1100
JAMES950
FORD3000
MILLER1300
```

You would like the result to be:

```
ENAME             SAL
---------- ----------
SMITH              800
ALLEN             1600
WARD              1250
JONES             2975
MARTIN            1250
BLAKE             2850
CLARK             2450
SCOTT             3000
KING              5000
TURNER            1500
ADAMS             1100
JAMES              950
FORD              3000
MILLER            1300
```

SOLUTION

Use the built-in functions TRANSLATE and REPLACE to isolate the character from the numeric data. Like other recipes in this lab, the trick is to use TRANSLATE to transform multiple characters into a single character you can reference. This way you are no longer searching for multiple numbers or characters, rather one character to represent all numbers or one character to represent all characters.

DB2

Use the functions TRANSLATE and REPLACE to isolate and separate the numeric from the character data:

```
select replace(
     translate(data,'0000000000','0123456789'),'0','') ename,
       cast(
     replace(
   translate(lower(data),repeat('z',26),
          'abcdefghijklmnopqrstuvwxyz'),'z','') as integer) sal
   from (
 select ename||cast(sal as char(4)) data
   from emp
        ) x
```

Oracle

Use the functions TRANSLATE and REPLACE to isolate and separate the numeric from the character data:

```
select replace(
     translate(data,'0123456789','0000000000'),'0') ename,
     to_number(
       replace(
       translate(lower(data),
                 'abcdefghijklmnopqrstuvwxyz',
                  rpad('z',26,'z')),'z')) sal
  from (
select ename||sal data
  from emp
       )
```

PostgreSQL

Use the functions TRANSLATE and REPLACE to isolate and separate the numeric from the character data:

```
 1 select replace(
 2      translate(data,'0123456789','0000000000'),'0','') as ename,
 3           cast(
 4        replace(
 5      translate(lower(data),
 6                'abcdefghijklmnopqrstuvwxyz',
 7                rpad('z',26,'z')),'z','') as integer) as sal
 8   from (
 9 select ename||sal as data
10   from emp
11        ) x
```

DISCUSSION

The syntax is a bit different for each DBMS, but the technique is the same. I will use the solution for Oracle in the discussion section. The key to solving this problem is to isolate the numeric and character data. You can use TRANSLATE and REPLACE to do this. To extract the numeric data, first isolate all character data using TRANSLATE:

```
select data,
       translate(lower(data),
                 'abcdefghijklmnopqrstuvwxyz',
                 rpad('z',26,'z')) sal
  from (select ename||sal data from emp)

DATA                 SAL
-------------------- --------------------
SMITH800             zzzzz800
ALLEN1600            zzzzz1600
WARD1250             zzzz1250
JONES2975            zzzzz2975
MARTIN1250           zzzzzz1250
BLAKE2850            zzzzz2850
CLARK2450            zzzzz2450
SCOTT3000            zzzzz3000
KING5000             zzzz5000
TURNER1500           zzzzzz1500
ADAMS1100            zzzzz1100
JAMES950             zzzzz950
FORD3000             zzzz3000
MILLER1300           zzzzzz1300
```

By using TRANSLATE you convert every non-numeric character into a lowercase Z. The next step is to remove all instances of lowercase Z from each record using REPLACE, leaving only numerical characters that can then be cast to a number:

```
select data,
       to_number(
         replace(
       translate(lower(data),
                 'abcdefghijklmnopqrstuvwxyz',
                 rpad('z',26,'z')),'z')) sal
  from (select ename||sal data from emp)

DATA                       SAL
-------------------- ----------
SMITH800                    800
ALLEN1600                  1600
WARD1250                   1250
JONES2975                  2975
MARTIN1250                 1250
BLAKE2850                  2850
CLARK2450                  2450
SCOTT3000                  3000
KING5000                   5000
TURNER1500                 1500
ADAMS1100                  1100
JAMES950                    950
FORD3000                   3000
MILLER1300                 1300
```

To extract the non-numeric characters, isolate the numeric characters using TRANSLATE:

```
select data,
       translate(data,'0123456789','0000000000') ename
  from (select ename||sal data from emp)

DATA                 ENAME
-------------------- ----------
SMITH800             SMITH000
ALLEN1600            ALLEN0000
WARD1250             WARD0000
JONES2975            JONES0000
MARTIN1250           MARTIN0000
BLAKE2850            BLAKE0000
CLARK2450            CLARK0000
SCOTT3000            SCOTT0000
KING5000             KING0000
TURNER1500           TURNER0000
ADAMS1100            ADAMS0000
JAMES950             JAMES000
FORD3000             FORD0000
MILLER1300           MILLER0000
```

By using TRANSLATE you convert every numeric character into a zero. The next step is to remove all instances of zero from each record using REPLACE, leaving only non-numeric characters:

```
select data,
       replace(translate(data,'0123456789','0000000000'),'0') ename
  from (select ename||sal data from emp)

 DATA                 ENAME
 -------------------- -------
 SMITH800             SMITH
 ALLEN1600            ALLEN
 WARD1250             WARD
 JONES2975            JONES
 MARTIN1250           MARTIN
 BLAKE2850            BLAKE
 CLARK2450            CLARK
 SCOTT3000            SCOTT
 KING5000             KING
 TURNER1500           TURNER
 ADAMS1100            ADAMS
 JAMES950             JAMES
 FORD3000             FORD
 MILLER1300           MILLER
```

Put the two techniques together and you have your solution.

6.6. Determining Whether a String Is Alphanumeric

PROBLEM

You want to return rows from a table only when a column of interest contains no characters other than numbers and letters. Consider the following view V (SQL Server users will use the operator "+" for concatenation instead of "||"):

```
create view V as
select ename as data
  from emp
 where deptno=10
 union all
select ename||', $'|| cast(sal as char(4)) ||'.00' as data
  from emp
 where deptno=20
 union all
select ename|| cast(deptno as char(4)) as data
  from emp
 where deptno=30
```

The view V represents your table, and it returns the following:

```
DATA
--------------------
CLARK
KING
MILLER
SMITH, $800.00
JONES, $2975.00
SCOTT, $3000.00
ADAMS, $1100.00
FORD, $3000.00
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30
```

However, from the view's data you want to return only the following records:

```
DATA
-------------
CLARK
KING
MILLER
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30
```

In short, you wish to omit those rows containing data other than letters and digits.

SOLUTION

It may seem intuitive at first to solve the problem by searching for all the possible non-alphanumeric characters that can be found in a string, but, on the contrary, you will find it easier to do the exact opposite: find all the alphanumeric characters. By doing so, you can treat all the alphanumeric characters as one by converting them to one single character. The reason you want to do this is so the alphanumeric characters can be manipulated together, as a whole. Once you've generated a copy of the string in which all alphanumeric characters are represented by a single character of your choosing, it is easy to isolate the alphanumeric characters from any other characters.

DB2

Use the function TRANSLATE to convert all alphanumeric characters to a single character, then identify any rows that have characters other than the converted alphanumeric character. For DB2 users, the CAST function calls in view V are necessary; otherwise, the view cannot be created due to type conversion errors. Take extra care when working with casts to CHAR as they are fixed length (padded):

```
select data
  from V
 where translate(lower(data),
                 repeat('a',36),
                 '0123456789abcdefghijklmnopqrstuvwxyz') =
                 repeat('a',length(data))
```

MySQL

The syntax for view V is slightly different in MySQL:

```
create view V as
select ename as data
  from emp
 where deptno=10
 union all
select concat(ename,', $',sal,'.00') as data
  from emp
 where deptno=20
 union all
select concat(ename,deptno) as data
  from emp
 where deptno=30
```

Use a regular expression to easily find rows that contain non-alphanumeric data:

```
select data
  from V
 where data regexp '[^0-9a-zA-Z]' = 0
```

Oracle and PostgreSQL

Use the function TRANSLATE to convert all alphanumeric characters to a single character, then identify any rows that have characters other than the converted alphanumeric character. The CAST function calls in view V are not needed for Oracle and PostgreSQL. Take extra care when working with casts to CHAR as they are fixed length (padded). If you decide to cast, cast to VARCHAR or VARCHAR2:

```
select data
  from V
 where translate(lower(data),
                 '0123456789abcdefghijklmnopqrstuvwxyz',
                 rpad('a',36,'a')) = rpad('a',length(data),'a')
```

SQL Server

Because SQL Server does not support a TRANSLATE function, you must walk each row and find any that contains a character that contains a non-alphanumeric value. That can be done many ways, but the following solution uses an ASCII-value evaluation:

```
select data
  from (
select v.data, iter.pos,
       substring(v.data,iter.pos,1) c,
       ascii(substring(v.data,iter.pos,1)) val
  from v,
       ( select id as pos from t100 ) iter
 where iter.pos <= len(v.data)
       ) x
 group by data
 having min(val) between 48 and 122
```

DISCUSSION

The key to these solutions is being able to reference multiple characters concurrently. By using the function TRANSLATE you can easily manipulate all numbers or all characters without having to “iterate” and inspect each character one by one.

DB2, Oracle, and PostgreSQL

Only 9 of the 14 rows from view V are alphanumeric. To find the rows that are alphanumeric only, simply use the function TRANSLATE. In this example, TRANSLATE converts characters 0–9 and a–z to “a”. Once the conversion is done, the converted row is then compared with a string of all “a” with the same length (as the row). If the length is the same, then you know all the characters are alphanumeric and nothing else.

By using the TRANSLATE function (using the Oracle syntax):

```
where translate(lower(data),
                '0123456789abcdefghijklmnopqrstuvwxyz',
                 rpad('a',36,'a'))
```

you convert all numbers and letters into a distinct character (I chose “a”). Once the data is converted, all strings that are indeed alphanumeric can be identified as a string comprising only a single character (in this case, “a”). This can be seen by running TRANSLATE by itself:

```
select data, translate(lower(data),
                  '0123456789abcdefghijklmnopqrstuvwxyz',
                   rpad('a',36,'a'))
  from V

DATA                 TRANSLATE(LOWER(DATA)
-------------------- ---------------------
CLARK                aaaaa
…
SMITH, $800.00       aaaaa, $aaa.aa
…
ALLEN30              aaaaaaa
…
```

The alphanumeric values are converted, but the string lengths have not been modified. Because the lengths are the same, the rows to keep are the ones for which the call to TRANSLATE returns all a’s. You keep those rows, rejecting the others, by comparing each original string’s length with the length of its corresponding string of a’s:

```
select data, translate(lower(data),
                  '0123456789abcdefghijklmnopqrstuvwxyz',
                   rpad('a',36,'a')) translated,
         rpad('a',length(data),'a') fixed
  from V

DATA                 TRANSLATED           FIXED
-------------------- -------------------- ----------------
CLARK                aaaaa                aaaaa
…
SMITH, $800.00       aaaaa, $aaa.aa       aaaaaaaaaaaaaa
…
ALLEN30              aaaaaaa              aaaaaaa
…
```

The last step is to keep only the strings where TRANSLATED equals FIXED.

MySQL

The expression in the WHERE clause:

```
where data regexp '[^0-9a-zA-Z]' = 0
```

causes rows that have only numbers or characters to be returned. The value ranges in the brackets, “0-9a-zA-Z”, represent all possible numbers and letters. The character “^” is for negation, so the expression can be stated as “not numbers or letters.” A return value of 1 is true and 0 is false, so the whole expression can be stated as “return rows where anything other than numbers and letters is false.”

SQL Server
The first step is to walk each row returned by view V. Each character in the value returned for DATA will itself be returned as a row. The values returned by C represent each individual character for the values returned by DATA:

```
+----------------+------+------+------+
| data           | pos  | c    | val  |
+----------------+------+------+------+
| ADAMS, $1100.00 |    1 | A    |   65 |
| ADAMS, $1100.00 |    2 | D    |   68 |
| ADAMS, $1100.00 |    3 | A    |   65 |
| ADAMS, $1100.00 |    4 | M    |   77 |
| ADAMS, $1100.00 |    5 | S    |   83 |
| ADAMS, $1100.00 |    6 | ,    |   44 |
| ADAMS, $1100.00 |    7 |      |   32 |
| ADAMS, $1100.00 |    8 | $    |   36 |
| ADAMS, $1100.00 |    9 | 1    |   49 |
| ADAMS, $1100.00 |   10 | 1    |   49 |
| ADAMS, $1100.00 |   11 | 0    |   48 |
| ADAMS, $1100.00 |   12 | 0    |   48 |
| ADAMS, $1100.00 |   13 | .    |   46 |
| ADAMS, $1100.00 |   14 | 0    |   48 |
| ADAMS, $1100.00 |   15 | 0    |   48 |
```

Inline view Z not only returns each character in the column DATA row by row, it also provides the ASCII value for each character. For this particular implementation of SQL Server, the range 48–122 represents alphanumeric characters. With that knowledge, you can group each row in DATA and filter out any such that the minimum ASCII value is not in the 48–122 range.

6.7. Extracting Initials from a Name

PROBLEM

You want convert a full name into initials. Consider the following name:

```
Stewie Griffin
```

You would like to return:

```
S.G.
```

SOLUTION

It's important to keep in mind that SQL does not provide the flexibility of languages such as C or Python; therefore, creating a generic solution to deal with any name format is not something particularly easy to do in SQL. The solutions presented here expect the names to be either first and last name, or first, middle name/middle initial, and last name.

DB2
Use the built-in functions REPLACE, TRANSLATE, and REPEAT to extract the initials:

```
select replace(
       replace(
       translate(replace('Stewie Griffin', '.', ''),
                 repeat('#',26),
                 'abcdefghijklmnopqrstuvwxyz'),
                  '#','' ), ' ','.' )
                 ||'.'
  from t1
```

MySQL
Use the built-in functions CONCAT, CONCAT_WS, SUBSTRING, and SUBSTRING_ INDEX to extract the initials:

```
select case
         when cnt = 2 then
           trim(trailing '.' from
                concat_ws('.',
                 substr(substring_index(name,' ',1),1,1),
                 substr(name,
                        length(substring_index(name,' ',1))+2,1),
                 substr(substring_index(name,' ',-1),1,1),
                 '.'))
         else
           trim(trailing '.' from
                concat_ws('.',
                 substr(substring_index(name,' ',1),1,1),
                 substr(substring_index(name,' ',-1),1,1)
                 ))
         end as initials
  from (
select name,length(name)-length(replace(name,' ','')) as cnt
  from (
select replace('Stewie Griffin','.','') as name from t1
       )y
       )x
```

Oracle and PostgreSQL

Use the built-in functions REPLACE, TRANSLATE, and RPAD to extract the initials:

```
select replace(
       replace(
       translate(replace('Stewie Griffin', '.', ''),
                 'abcdefghijklmnopqrstuvwxyz',
                 rpad('#',26,'#') ), '#','' ),' ','.' ) ||'.'
  from t1
```

SQL Server

As of the time of this writing, neither TRANSLATE nor CONCAT_WS is supported in SQL Server.

DISCUSSION

By isolating the capital letters you can extract the initials from a name. The following sections describe each vendor-specific solution in detail.

DB2

The REPLACE function will remove any periods in the name (to handle middle initials), and the TRANSLATE function will convert all non-uppercase letters to #.

```
select translate(replace('Stewie Griffin', '.', ''),
                 repeat('#',26),
                 'abcdefghijklmnopqrstuvwxyz')
  from t1

TRANSLATE('STE
--------------
S##### G######
```

At this point, the initials are the characters that are not #. The function REPLACE is then used to remove all the # characters:

```
select replace(
       translate(replace('Stewie Griffin', '.', ''),
                 repeat('#',26),
                 'abcdefghijklmnopqrstuvwxyz'),'#','')
  from t1

REP
---
S G
```

The next step is to replace the white space with a period by using REPLACE again:

```
select replace(
replace(
translate(replace('Stewie Griffin', '.', ''),
repeat('#',26),
'abcdefghijklmnopqrstuvwxyz'),'#',''),' ','.') || '.'
from t1

REPLA
-----
S.G
```

The final step is to append a decimal to the end of the initials.

Oracle and PostgreSQL
The REPLACE function will remove any periods in the name (to handle middle initials), and the TRANSLATE function will convert all non-uppercase letters to '#'.

```
select translate(replace('Stewie Griffin','.',''),
                 'abcdefghijklmnopqrstuvwxyz',
                 rpad('#',26,'#'))
  from t1

TRANSLATE('STE
--------------
S##### G######
```

At this point, the initials are the characters that are not “#”. The function REPLACE is then used to remove all the # characters:

```
select replace(
       translate(replace('Stewie Griffin','.',''),
                 'abcdefghijklmnopqrstuvwxyz',
                  rpad('#',26,'#')),'#','')
  from t1

REP
---
S G
```

The next step is to replace the white space with a period by using REPLACE again:

```
select replace(
       replace(
     translate(replace('Stewie Griffin','.',''),
               'abcdefghijklmnopqrstuvwxyz',
               rpad('#',26,'#') ),'#',''),' ','.') || '.'
  from t1

REPLA
-----
S.G
```

The final step is to append a decimal to the end of the initials.

MySQL

The inline view Y is used to remove any period from the name. The inline view X finds the number of white spaces in the name so the SUBSTR function can be called the correct number of times to extract the initials. The three calls to SUBSTRING_ INDEX parse the string into individual names based on the location of the white space. Because there is only a first and last name, the code in the ELSE portion of the case statement is executed:

```
select substr(substring_index(name, ' ',1),1,1) as a,
       substr(substring_index(name,' ',-1),1,1) as b
  from (select 'Stewie Griffin' as name from t1) x

A B
- -
S G
```

If the name in question has a middle name or initial, the initial would be returned by executing

```
substr(name,length(substring_index(name, ' ',1))+2,1)
```

which finds the end of the first name then moves two spaces to the beginning of the middle name or initial; that is, the start position for SUBSTR. Because only onecharacter is kept, the middle name or initial is successfully returned. The initials are then passed to CONCAT_WS, which separates the initials by a period:

```
select concat_ws('.',
                 substr(substring_index(name, ' ',1),1,1),
                 substr(substring_index(name,' ',-1),1,1),
                 '.' ) a
  from (select 'Stewie Griffin' as name from t1) x

A
-----
S.G..
```

The last step is to trim the extraneous period from the initials.

6.8. Ordering by Parts of a String

PROBLEM

You want to order your result set based on a substring. Consider the following records:

```
ENAME
----------
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER
```

You want the records to be ordered based on the last two characters of each name:

```
ENAME
---------
ALLEN
TURNER
MILLER
JONES
JAMES
MARTIN
BLAKE
ADAMS
KING
WARD
FORD
CLARK
SMITH
SCOTT
```

SOLUTION

The key to this solution is to find and use your DBMS's built-in function to extract the substring on which you wish to sort. This is typically done with the SUBSTR function.

DB2, Oracle, MySQL, and PostgreSQL
Use a combination of the built-in functions LENGTH and SUBSTR to order by a specific part of a string:

```
1 select ename
2   from emp
3  order by substr(ename,length(ename)-1,)
```

SQL Server

Use functions SUBSTRING and LEN to order by a specific part of a string:

```
1 select ename
2   from emp
3  order by substring(ename,len(ename)-1,2)
```

DISCUSSION

By using a SUBSTR expression in your ORDER BY clause, you can pick any part of a string to use in ordering a result set. You're not limited to SUBSTR either. You can order rows by the result of almost any expression.

6.9. Ordering by a Number in a String

PROBLEM

You want order your result set based on a number within a string. Consider the following view:

```
create view V as
select e.ename ||' '||
        cast(e.empno as char(4))||' '||
        d.dname as data
  from emp e, dept d
 where e.deptno=d.deptno
```

This view returns the following data:

```
DATA
 ---------------------------
 CLARK   7782 ACCOUNTING
 KING    7839 ACCOUNTING
 MILLER  7934 ACCOUNTING
 SMITH   7369 RESEARCH
 JONES   7566 RESEARCH
 SCOTT   7788 RESEARCH
 ADAMS   7876 RESEARCH
 FORD    7902 RESEARCH
 ALLEN   7499 SALES
 WARD    7521 SALES
 MARTIN  7654 SALES
 BLAKE   7698 SALES
 TURNER  7844 SALES
 JAMES   7900 SALES
```

You want to order the results based on the employee number, which falls between the employee name and respective department:

```
DATA
 ---------------------------
SMITH    7369 RESEARCH
ALLEN    7499 SALES
WARD     7521 SALES
JONES    7566 RESEARCH
MARTIN   7654 SALES
BLAKE    7698 SALES
CLARK    7782 ACCOUNTING
SCOTT    7788 RESEARCH
KING     7839 ACCOUNTING
TURNER   7844 SALES
ADAMS    7876 RESEARCH
JAMES    7900 SALES
FORD     7902 RESEARCH
MILLER   7934 ACCOUNTING
```

SOLUTION

Each solution uses functions and syntax specific to its DBMS, but the method (making use of the built-in functions REPLACE and TRANSLATE) is the same for each. The idea is to use REPLACE and TRANSLATE to remove non-digits from the strings, leaving only the numeric values upon which to sort.

DB2

Use the built-in functions REPLACE and TRANSLATE to order by numeric characters in a string:

```
select data
  from V
 order by
        cast(
      replace(
    translate(data,repeat('#',length(data)),
      replace(
    translate(data,'##########','0123456789'),
             '#','')),'#','') as integer)
```

Oracle

Use the built-in functions REPLACE and TRANSLATE to order by numeric characters in a string:

```
select data
  from V
 order by
        to_number(
          replace(
        translate(data,
          replace(
        translate(data,'0123456789','##########'),
                 '#'),rpad('#',20,'#')),'#'))
```

PostgreSQL

Use the built-in functions REPLACE and TRANSLATE to order by numeric characters in a string:

```
  select data
    from V
   order by
          cast(
        replace(
      translate(data,
        replace(
      translate(data,'0123456789','##########'),
               '#',''),rpad('#',20,'#')),'#','') as integer)
```

MySQL and SQL Server

As of the time of this writing, neither vendor supplies the TRANSLATE function.

DISCUSSION

The purpose of view V is only to supply rows on which to demonstrate this recipe's solution. The view simply concatenates several columns from the EMP table. The solution shows how to take such concatenated text as input and sort it by the employee number embedded within.

The ORDER BY clause in each solution may look a bit intimidating but performs quite well and is pretty straightforward once you examine it piece by piece. To order by the numbers in the string, it's easiest to remove any characters that are not numbers. Once the non-numeric characters are removed all that is left to do is cast the string of numerals into a number, then sort as you see fit. Before examining each function call it is important to understand the order in which each function is called. Starting with the innermost call, TRANSLATE (line 8 from each of the original solutions), you see that:

TRANSLATE (line 8) is called and the results are returned to

REPLACE (line 7) and those results are returned to

TRANSLATE (line 6) and those results are returned to

REPLACE (line 5) and those results are returned and finally

cast into a number

The first step is to convert the numbers into characters that do not exist in the rest of the string. For this example, I chose "#" and used TRANSLATE to convert all non-numeric characters into occurrences of "#". For example, the following query shows the original data on the left and the results from the first translation:

```
select data,
       translate(data,'0123456789','##########') as tmp
  from V

DATA                           TMP
 ----------------------------- ----------------------
 CLARK   7782 ACCOUNTING        CLARK   #### ACCOUNTING
 KING    7839 ACCOUNTING        KING    #### ACCOUNTING
 MILLER  7934 ACCOUNTING        MILLER  #### ACCOUNTING
 SMITH   7369 RESEARCH          SMITH   #### RESEARCH
 JONES   7566 RESEARCH          JONES   #### RESEARCH
 SCOTT   7788 RESEARCH          SCOTT   #### RESEARCH
 ADAMS   7876 RESEARCH          ADAMS   #### RESEARCH
 FORD    7902 RESEARCH          FORD    #### RESEARCH
 ALLEN   7499 SALES             ALLEN   #### SALES
 WARD    7521 SALES             WARD    #### SALES
 MARTIN  7654 SALES             MARTIN  #### SALES
 BLAKE   7698 SALES             BLAKE   #### SALES
 TURNER 7844 SALES          TURNER  #### SALES
 JAMES  7900 SALES          JAMES   #### SALES
```

TRANSLATE finds the numerals in each string and converts each one to to the "#" character. The modified strings are then returned to REPLACE (line 11), which removes all occurrences of "#":

```
select data,
replace(
translate(data,'0123456789','##########'),'#') as tmp
  from V

DATA                           TMP
 ------------------------------ -------------------
 CLARK   7782 ACCOUNTING        CLARK   ACCOUNTING
 KING    7839 ACCOUNTING        KING    ACCOUNTING
 MILLER  7934 ACCOUNTING        MILLER  ACCOUNTING
 SMITH   7369 RESEARCH          SMITH   RESEARCH
 JONES   7566 RESEARCH          JONES   RESEARCH
 SCOTT   7788 RESEARCH          SCOTT   RESEARCH
 ADAMS   7876 RESEARCH          ADAMS   RESEARCH
 FORD    7902 RESEARCH          FORD    RESEARCH
 ALLEN   7499 SALES             ALLEN   SALES
 WARD    7521 SALES             WARD    SALES
 MARTIN  7654 SALES             MARTIN  SALES
 BLAKE   7698 SALES             BLAKE   SALES
 TURNER  7844 SALES             TURNER  SALES
 JAMES   7900 SALES             JAMES   SALES
```

The strings are then returned to TRANSLATE once again, but this time it's the second (outermost) TRANSLATE in the solution. TRANSLATE searches the original string for any characters that match the characters in TMP. If any are found, they too are converted to "#"s. This conversion allows all non-numeric characters to be treated as a single character (because they are all transformed to the same character):

```
select data, translate(data,
             replace(
             translate(data,'0123456789','##########'),
             '#'),
             rpad('#',length(data),'#')) as tmp
  from V

DATA                           TMP
------------------------------ -------------------------
CLARK   7782 ACCOUNTING        ########7782###########
KING    7839 ACCOUNTING        ########7839###########
MILLER  7934 ACCOUNTING        ########7934###########
SMITH   7369 RESEARCH          ########7369#########
JONES   7566 RESEARCH          ########7566#########
SCOTT   7788 RESEARCH          ########7788#########
ADAMS   7876 RESEARCH          ########7876#########
FORD    7902 RESEARCH          ########7902#########
ALLEN   7499 SALES             ########7499######
WARD    7521 SALES             ########7521######
MARTIN  7654 SALES             ########7654######
BLAKE   7698 SALES             ########7698######
TURNER  7844 SALES             ########7844######
JAMES   7900 SALES             ########7900######
```

The next step is to remove all “#” characters through a call to REPLACE (line 8), leaving you with only numbers:

```
select data, replace(
             translate(data,
             replace(
           translate(data,'0123456789','##########'),
                     '#'),
                     rpad('#',length(data),'#')),'#') as tmp
  from V

DATA                           TMP
------------------------------ ----------
CLARK   7782 ACCOUNTING        7782
KING    7839 ACCOUNTING        7839
MILLER  7934 ACCOUNTING        7934
SMITH   7369 RESEARCH          7369
JONES   7566 RESEARCH          7566
SCOTT   7788 RESEARCH          7788
ADAMS   7876 RESEARCH          7876
FORD    7902 RESEARCH          7902
ALLEN   7499 SALES             7499
WARD    7521 SALES             7521
MARTIN  7654 SALES             7654
BLAKE   7698 SALES             7698
TURNER  7844 SALES             7844
JAMES   7900 SALES             7900
```

Finally, cast TMP to a number (line 4) using the appropriate DBMS function (often CAST) to accomplish this:

```
select data, to_number(
              replace(
             translate(data,
             replace(
       translate(data,'0123456789','##########'),
                 '#'),
                 rpad('#',length(data),'#')),'#')) as tmp
  from V

DATA                                TMP
------------------------------ ----------
CLARK   7782 ACCOUNTING              7782
KING    7839 ACCOUNTING              7839
MILLER  7934 ACCOUNTING              7934
SMITH   7369 RESEARCH                7369
JONES   7566 RESEARCH                7566
SCOTT   7788 RESEARCH                7788
ADAMS   7876 RESEARCH                7876
FORD    7902 RESEARCH                7902
ALLEN   7499 SALES                   7499
WARD    7521 SALES                   7521
MARTIN  7654 SALES                   7654
BLAKE   7698 SALES                   7698
TURNER  7844 SALES                   7844
JAMES   7900 SALES                   7900
```

When developing queries like this, it's helpful to work with your expressions in the SELECT list. That way, you can easily view the intermediate results as you work toward a final solution. However, because the point of this recipe is to order the results, ultimately you should place all the function calls into the ORDER BY clause:

```
select data
  from V
 order by
        to_number(
          replace(
        translate( data,
          replace(
        translate( data,'0123456789','##########'),
                  '#'),rpad('#',length(data),'#')),'#'))

DATA
--------------------------
SMITH   7369 RESEARCH
ALLEN   7499 SALES
WARD    7521 SALES
JONES   7566 RESEARCH
MARTIN  7654 SALES
BLAKE   7698 SALES
CLARK   7782 ACCOUNTING
SCOTT   7788 RESEARCH
KING    7839 ACCOUNTING
TURNER  7844 SALES
ADAMS   7876 RESEARCH
JAMES   7900 SALES
FORD    7902 RESEARCH
MILLER  7934 ACCOUNTING
```

As a final note, the data in the view is comprised of three fields, only one being numeric. Keep in mind that if there had been multiple numeric fields, they would have all been concatenated into one number before the rows were sorted.

6.10. Creating a Delimited List from Table Rows

PROBLEM

You want to return table rows as values in a delimited list, perhaps delimited by commas, rather than in vertical columns as they normally appear. You want to convert a result set from this:

```
DEPTNO EMPS
------ ----------
    10 CLARK
    10 KING
    10 MILLER
    20 SMITH
    20 ADAMS
    20 FORD
    20 SCOTT
    20 JONES
    30 ALLEN
    30 BLAKE
    30 MARTIN
    30 JAMES
    30 TURNER
    30 WARD
```

to this:

```
 DEPTNO EMPS
------- -----------------------------------
     10 CLARK,KING,MILLER
     20 SMITH,JONES,SCOTT,ADAMS,FORD
     30 ALLEN,WARD,MARTIN,BLAKE,TURNER,JAMES
```

SOLUTION

Each DBMS requires a different approach to this problem. The key is to take advantage of the built-in functions provided by your DBMS. Understanding what is available to you will allow you to exploit your DBMS's functionality and come up with creative solutions for a problem that is typically not solved in SQL.

DB2
Use recursive WITH to build the delimited list:

```
  with x (deptno, cnt, list, empno, len)
    as (
select deptno, count(*) over (partition by deptno),
       cast(ename as varchar(100)), empno, 1
  from emp
 union all
 select x.deptno, x.cnt, x.list ||','|| e.ename, e.empno, x.len+1
  from emp e, x
 where e.deptno = x.deptno
   and e.empno > x. empno
       )
select deptno,list
  from x
 where len = cnt
```

MySQL

Use the built-in function GROUP_CONCAT to build the delimited list:

```
select deptno,
       group_concat(ename order by empno separator, ',') as emps
  from emp
 group by deptno
```

Oracle

Use the built-in function SYS_CONNECT_BY_PATH to build the delimited list:

```
select deptno,
       ltrim(sys_connect_by_path(ename,','),',') emps
  from (
select deptno,
       ename,
       row_number() over
                (partition by deptno order by empno) rn,
       count(*) over
                (partition by deptno) cnt
  from emp
       )
 where level = cnt
 start with rn = 1
connect by prior deptno = deptno and prior rn = rn-1
```

PostgreSQL

PostgreSQL does not offer a standard built-in function for creating a delimited list, so it is necessary to know how many values will be in the list in advance. Once you know the size of the largest list, you can determine the number of values to append to create your list by using standard transposition and concatenation:

```
select deptno,
       rtrim(
           max(case when pos=1 then emps else '' end)||
           max(case when pos=2 then emps else '' end)||
           max(case when pos=3 then emps else '' end)||
           max(case when pos=4 then emps else '' end)||
           max(case when pos=5 then emps else '' end)||
           max(case when pos=6 then emps else '' end),','
       ) as emps
  from (
select a.deptno,
       a.ename||',' as emps,
       d.cnt,
       (select count(*) from emp b
          where a.deptno=b.deptno and b.empno <= a.empno) as pos
  from emp a,
       (select deptno, count(ename) as cnt
          from emp
          group by deptno) d
 where d.deptno=a.deptno
       ) x
 group by deptno
 order by 1
```

SQL Server

Use recursive WITH to build the delimited list:

```
  with x (deptno, cnt, list, empno, len)
     as (
 select deptno, count(*) over (partition by deptno),
        cast(ename as varchar(100)),
        empno,
        1
   from emp
  union all
 select x.deptno, x.cnt,
        cast(x.list + ',' + e.ename as varchar(100)),
        e.empno, x.len+1
   from emp e, x
 where e.deptno = x.deptno
   and e.empno > x. empno
           )
 select deptno,list
   from x
  where len = cnt
  order by 1
```

DISCUSSION

Being able to create delimited lists in SQL is useful because it is a common requirement. Yet each DBMS offers a unique method for building such a list in SQL. There's very little commonality between the vendor-specific solutions; the techniques vary from using recursion, to hierarchal functions, to classic transposition, to aggregation.

DB2 and SQL Server

The solution for these two databases differ slightly in syntax (the concatenation operators are "||" for DB2 and "+" for SQL Server), but the technique is the same. The first query in the WITH clause (upper portion of the UNION ALL) returns the following information about each employee: the department, the number of employees in that department, the name, the ID, and a constant 1 (which at this point doesn't do anything). Recursion takes place in the second query (lower half of the UNION ALL) to build the list. To understand how the list is built, examine the following excerpts from the solution: first, the third SELECT-list item from the second query in the union:

```
x.list ||','|| e.ename
```

and then the WHERE clause from that same query:

```
where e.deptno = x.deptno
  and e.empno > x.empno
```

The solution works by first ensuring the employees are in the same department. Then, for every employee returned by the upper portion of the UNION ALL, append the name of the employees who have a greater EMPNO. By doing this, you ensure that no employee will have his own name appended. The expression

```
x.len+1
```

increments LEN (which starts at 1) every time an employee has been evaluated. When the incremented value equals the number of employees in the department:

```
where len = cnt
```

you know you have evaluated all the employees and have completed building the list. That is crucial to the query as it not only signals when the list is complete, but also stops the recursion from running longer than necessary.

MySQL

The function GROUP_CONCAT does all the work. It concatenates the values found in the column

passed to it, in this case ENAME. It's an aggregate function, thus the need for GROUP BY in the query.

Oracle

The first step to understanding the Oracle query is to break it down. Running the inline view by itself (lines 4–10), you generate a result set that includes the following for each employee: her department, her name, a rank within her respective department that is derived by an ascending sort on EMPNO, and a count of all employees in her department. For example:

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

DEPTNO ENAME      RN CNT
------ ---------- -- ---
    10 CLARK       1   3
    10 KING        2   3
    10 MILLER      3   3
    20 SMITH       1   5
    20 JONES       2   5
    20 SCOTT       3   5
    20 ADAMS       4   5
    20 FORD        5   5
    30 ALLEN       1   6
    30 WARD        2   6
    30 MARTIN      3   6
    30 BLAKE       4   6
    30 TURNER      5   6
    30 JAMES       6   6
```

The purpose of the rank (aliased RN in the query) is to allow you to walk the tree. Since the function ROW_NUMBER generates an enumeration starting from one with no duplicates or gaps, just subtract one (from the current value) to reference a prior (or parent) row. For example, the number prior to 3 is 3 minus 1, which equals 2. In this context, 2 is the parent of 3; you can observe this on line 12. Additionally, the lines

```
start with rn = 1
connect by prior deptno = deptno
```

identify the root for each DEPTNO as having RN equal to 1 and create a new list whenever a new department is encountered (whenever a new occurrence of 1 is found for RN).

At this point, it's important to stop and look at the ORDER BY portion of the ROW_NUMBER function. Keep in mind the names are ranked by EMPNO and the list will be created in that order. The number of employees per department is calculated (aliased CNT) and is used to ensure that the query returns only the list that has all the employee names for a department. This is done because SYS_CONNECT_ BY_PATH builds the list iteratively, and you do not want to end up with partial lists.

For hierarchical queries, the pseudocolumn LEVEL starts with 1 (for queries not using CONNECT BY, LEVEL is 0, unless you are on 10g and later when LEVEL is only available when using CONNECT BY) and increments by one after each employee in a department has been evaluated (for each level of depth in the hierarchy). Because of this, you know that once LEVEL reaches CNT, you have reached the last EMPNO and will have a complete list.

TIP

_The SYS_CONNECT_BY_PATH function prefixes the list with your chosen delimiter (in this case, a comma). You may or may not want that behavior. In this recipe's solution, the call to the function LTRIM removes the leading comma from the list.

_

PostgreSQL

PostgreSQL's solution requires you to know in advance the maximum number of employees in any one department. Running the inline view by itself (lines 11–18) generates a result set that includes (for each employee) his department, his name with a comma appended, the number of employees in his department, and the number of employees who have an EMPNO that is less than his:

```
deptno  |  emps    | cnt | pos
--------+----------+-----+-----
    20  |  SMITH,  |   5 |   1
    30  |  ALLEN,  |   6 |   1
    30  |  WARD,   |   6 |   2
    20  |  JONES,  |   5 |   2
    30  |  MARTIN, |   6 |   3
    30  |  BLAKE,  |   6 |   4
    10  |  CLARK,  |   3 |   1
    20  |  SCOTT,  |   5 |   3
    10  |  KING,   |   3 |   2
    30  |  TURNER, |   6 |   5
    20  |  ADAMS,  |   5 |   4
    30  |  JAMES,  |   6 |   6
    20  |  FORD,   |   5 |   5
    10  |  MILLER, |   3 |   3
```

The scalar subquery, POS (lines 14-15), is used to rank each employee by EMPNO. For example, the line

```
max(case when pos = 1 then ename else '' end)||
```

evaluates whether or not POS equals 1. The CASE expression returns the employee name when POS is 1, and otherwise returns NULL.

You must query your table first to find the largest number of values that could be in any one list. Based on the EMP table, the largest number of employees in any one department is six, so the largest number of items in a list is six.

The next step is to begin creating the list. Do this by performing some conditional logic (in the form of CASE expressions) on the rows returned from the inline view.

You must write as many CASE expressions as there are possible values to be concatenated together.

If POS equals one, the current name is added to the list. The second CASE expression evaluates whether or not POS equals two; if it does, then the second name is appended to the first. If there is no second name, then an additional comma is appended to the first name (this process is repeated for each distinct value of POS until the last one is reached).

The use of the MAX function is necessary because you want to build only one list per department (you can also use MIN; it makes no difference in this case, since POS returns only one value for each case evaluation). Whenever an aggregate function is used, any items in the SELECT list not acted upon by the aggregate must be specified in the GROUP BY clause. This guarantees you will have only one row per item in the SELECT list not acted upon by the aggregate function.

Notice that you also need the function RTRIM to remove trailing commas; the number of commas will always be equal to the maximum number of values that could potentially be in a list (in this case, six).

6.11. Converting Delimited Data into a Multi-Valued IN-List

PROBLEM

You have delimited data that you want to pass to the IN-list iterator of a WHERE clause. Consider the following string:

```
7654,7698,7782,7788
```

You would like to use the string in a WHERE clause but the following SQL fails because EMPNO is a numeric column:

```
select ename,sal,deptno
  from emp
 where empno in ( '7654,7698,7782,7788' )
```

This SQL fails because, while EMPNO is a numeric column, the IN list is composed of a single string value. You want that string to be treated as a comma-delimited list of numeric values.

SOLUTION

On the surface it may seem that SQL should do the work of treating a delimited string as a list of delimited values for you, but that is not the case. When a comma embedded within quotes is encountered, SQL can't possibly know that signals a multi-valued list. SQL must treat everything between the quotes as a single entity, as one string value. You must break the string up into individual EMPNOs. The key to this solution is to walk the string, but not into individual characters. You want to walk the string into valid EMPNO values.

DB2

By walking the string passed to the IN-list, you can easily convert it to rows. The functions ROW_NUMBER, LOCATE, and SUBSTR are particularly useful here:

```
select empno,ename,sal,deptno
  from emp
 where empno in (
select cast(substr(c,2,locate(',',c,2)-2) as integer) empno
  from (
select substr(csv.emps,cast(iter.pos as integer)) as c
  from (select ','||'7654,7698,7782,7788'||',' emps
          from t1) csv,
       (select id as pos
          from t100 ) iter
 where iter.pos <= length(csv.emps)
       ) x
 where length(c) > 1
   and substr(c,1,1) = ','
       )
```

MySQL

By walking the string passed to the IN-list, you can easily convert it to rows:

```
select empno, ename, sal, deptno
  from emp
 where empno in
       (
select substring_index(
       substring_index(list.vals,',',iter.pos),',',-1) empno
  from (select id pos from t10) as iter,
       (select '7654,7698,7782,7788' as vals
          from t1) list
  where iter.pos <=
       (length(list.vals)-length(replace(list.vals,',','')))+1
       )
```

Oracle

By walking the string passed to the IN-list, you can easily convert it to rows. The functions ROWNUM, SUBSTR, and INSTR are particularly useful here:

```
select empno,ename,sal,deptno
  from emp
 where empno in (
       select to_number(
                  rtrim(
                substr(emps,
                 instr(emps,',',1,iter.pos)+1,
                 instr(emps,',',1,iter.pos+1)
                 instr(emps,',',1,iter.pos)),',')) emps
         from (select ','||'7654,7698,7782,7788'||',' emps from t1) csv,
              (select rownum pos from emp) iter
        where iter.pos <= ((length(csv.emps)-
                  length(replace(csv.emps,',')))/length(','))-1
)
```

Postgres

By walking the string passed to the IN-list, you can easily convert it to rows. The function SPLIT_PART makes it easy to parse the string into individual numbers:

```
select ename,sal,deptno
  from emp
 where empno in (
select cast(empno as integer) as empno
  from (
select split_part(list.vals,',',iter.pos) as empno
  from (select id as pos from t10) iter,
       (select ','||'7654,7698,7782,7788'||',' as vals
          from t1) list
 where iter.pos <=
       length(list.vals)-length(replace(list.vals,',',''))
       ) z
 where length(empno) > 0
       )
```

SQL Server

By walking the string passed to the IN-list, you can easily convert it to rows. The functions ROW_NUMBER, CHARINDEX, and SUBSTRING are particularly useful here:

```
select empno,ename,sal,deptno
  from emp
 where empno in (select substring(c,2,charindex(',',c,2)-2) as empno
  from (
select substring(csv.emps,iter.pos,len(csv.emps)) as c
  from (select ','+'7654,7698,7782,7788'+',' as emps
          from t1) csv,
       (select id as pos
          from t100) iter
 where iter.pos <= len(csv.emps)
      ) x
 where len(c) > 1
   and substring(c,1,1) = ','
      )
```

DISCUSSION

The first and most important step in this solution is to walk the string. Once you've accomplished that, all that's left is to parse the string into individual, numeric values using your DBMS's provided functions.

DB2 and SQL Server

The inline view X (lines 6–11) walks the string. The idea in this solution is to "walk through" the string, so that each row has one less character than the one before it:

```
,7654,7698,7782,7788,
7654,7698,7782,7788,
654,7698,7782,7788,
54,7698,7782,7788,
4,7698,7782,7788,
,7698,7782,7788,
7698,7782,7788,
698,7782,7788,
98,7782,7788,
8,7782,7788,
,7782,7788,
7782,7788,
782,7788,
82,7788,
2,7788,
,7788,
7788,
788,
88,
8,
,
```

Notice that by enclosing the string in commas (the delimiter), there's no need to make special checks as to where the beginning or end of the string is.

The next step is to keep only the values you want to use in the IN-list. The values to keep are the ones with leading commas, with the exception of the last row with its lone comma. Use SUBSTR or SUBSTRING to identify which rows have a leading comma, then keep all characters found before the next comma in that row. Once that's done, cast the string to a number so it can be properly evaluated against the numeric column EMPNO (lines 4–14):

```
 EMPNO
------
  7654
  7698
  7782
  7788
```

The final step is to use the results in a subquery to return the desired rows.

MySQL

The inline view (lines 5–9) walks the string. The expression on line 10 determines how many values are in the string by finding the number of commas (the delimiter) and adding one. The function SUBSTRING_INDEX (line 6) returns all characters in the string before (to the left of) the nth occurrence of a comma (the delimiter):

```
+--------------------+
| empno              |
+--------------------+
| 7654               |
| 7654,7698          |
| 7654,7698,7782     |
| 7654,7698,7782,7788 |
+--------------------+
```

Those rows are then passed to another call to SUBSTRING_INDEX (line 5); this time the nth occurrence of the delimited is –1, which causes all values to the right of the nth occurrence of the delimiter to be kept:

```
+-------+
| empno |
+-------+
| 7654  |
| 7698  |
| 7782  |
| 7788  |
+-------+
```

The final step is to plug the results into a subquery.

Oracle
The first step is to walk the string:

```
select emps,pos
  from (select ','||'7654,7698,7782,7788'||',' emps
          from t1) csv,
       (select rownum pos from emp) iter
 where iter.pos <=
 ((length(csv.emps)-length(replace(csv.emps,',')))/length(','))-1

EMPS                         POS
-------------------- ----------
,7654,7698,7782,7788,          1
,7654,7698,7782,7788,          2
,7654,7698,7782,7788,          3
,7654,7698,7782,7788,          4
```

The number of rows returned represents the number of values in your list. The values for POS are crucial to the query as they are needed to parse the string into individual values. The strings are parsed using SUBSTR and INSTR. POS is used to locate the nth occurrence of the delimiter in each string. By enclosing the strings in commas, no special checks are necessary to determine the

beginning or end of a string. The values passed to SUBSTR, INSTR (lines 7–9) locate the nth and nth+1 occurrence of the delimiter. By subtracting the value returned for the current comma (the location in the string where the current comma is) from the value returned bythe next comma (the location in the string where the next comma is) you can extract each value from the string:

```
select substr(emps,
       instr(emps,',',1,iter.pos)+1,
       instr(emps,',',1,iter.pos+1)
       instr(emps,',',1,iter.pos)) emps
  from (select ','||'7654,7698,7782,7788'||',' emps
          from t1) csv,
       (select rownum pos from emp) iter
 where iter.pos <=
  ((length(csv.emps)-length(replace(csv.emps,',')))/length(','))-1
 EMPS
 -----------
 7654,
 7698,
 7782,
 7788,
```

The final step is to remove the trailing comma from each value, cast it to a number, and plug it into a subquery.

PostgreSQL

The inline view Z (lines 6–9) walks the string. The number of rows returned is determined by how many values are in the string. To find the number of values in the string, subtract the size of the string without the delimiter from the size of the string with the delimiter (line 9). The function SPLIT_PART does the work of parsing the string. It looks for the value that comes before the nth occurrence of the delimiter:

```
select list.vals,
       split_part(list.vals,',',iter.pos) as empno,
       iter.pos
  from (select id as pos from t10) iter,
       (select ','||'7654,7698,7782,7788'||',' as vals
          from t1) list
 where iter.pos <=
       length(list.vals)-length(replace(list.vals,',',''))

     vals             | empno | pos
----------------------+-------+-----
,7654,7698,7782,7788, |       |   1
,7654,7698,7782,7788, | 7654  |   2
,7654,7698,7782,7788, | 7698  |   3
,7654,7698,7782,7788, | 7782  |   4
,7654,7698,7782,7788, | 7788  |   5
```

The final step is to cast the values (EMPNO) to a number and plug it into a subquery.

6.12. Alphabetizing a String

PROBLEM

You want alphabetize the individual characters within strings in your tables. Consider the following result set:

```
ENAME
----------
ADAMS
ALLEN
BLAKE
CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD
```

You would like the result to be:

```
OLD_NAME   NEW_NAME
---------- --------
ADAMS      AADMS
ALLEN      AELLN
BLAKE      ABEKL
CLARK      ACKLR
FORD       DFOR
JAMES      AEJMS
JONES      EJNOS
KING       GIKN
MARTIN     AIMNRT
MILLER     EILLMR
SCOTT      COSTT
SMITH      HIMST
TURNER     ENRRTU
WARD       ADRW
```

SOLUTION

This problem is a perfect example of why it is crucial to understand your DBMS and what functionality is available to you. In situations where your DBMS does not provide built-in functions to facilitate this solution, you need to come up with something creative. Compare the MySQL solution with the rest.

DB2

To alphabetize rows of strings it is necessary to walk each string then order its characters:

```
select ename,
       max(case when pos=1 then c else '' end)||
       max(case when pos=2 then c else '' end)||
       max(case when pos=3 then c else '' end)||
       max(case when pos=4 then c else '' end)||
       max(case when pos=5 then c else '' end)||
       max(case when pos=6 then c else '' end)
  from (
select e.ename,
       cast(substr(e.ename,iter.pos,1) as varchar(100)) c,
       cast(row_number()over(partition by e.ename
                             order by substr(e.ename,iter.pos,1))
           as integer) pos
  from emp e,
       (select cast(row_number()over() as integer) pos
          from emp) iter
 where iter.pos <= length(e.ename)
       ) x
 group by ename
```

MySQL

The key here is the GROUP_CONCAT function, which allows you to not only concatenate the characters that make up each name but also order them:

```
select ename, group_concat(c order by c separator '')
  from (
select ename, substr(a.ename,iter.pos,1) c
  from emp a,
       ( select id pos from t10 ) iter
 where iter.pos <= length(a.ename)
       ) x
 group by ename
```

Oracle

The function SYS_CONNECT_BY_PATH allows you to iteratively build a list:

```
select old_name, new_name
  from (
select old_name, replace(sys_connect_by_path(c,' '),' ') new_name
  from (
select e.ename old_name,
       row_number() over(partition by e.ename
                         order by substr(e.ename,iter.pos,1)) rn,
       substr(e.ename,iter.pos,1) c
  from emp e,
       ( select rownum pos from emp ) iter
 where iter.pos <= length(e.ename)
 order by 1
       ) x
 start with rn = 1
connect by prior rn = rn-1 and prior old_name = old_name
       )
 where length(old_name) = length(new_name)
```

PostgreSQL

PostgreSQL does not offer any built-in functions to easily sort characters in a string, so it is necessary not only to walk through each string but also to know in advance the largest length of any one name. View V is used in this solution for readability:

```
create or replace view V as
select x.*
  from (
select a.ename,
       substr(a.ename,iter.pos,1) as c
  from emp a,
       (select id as pos from t10) iter
 where iter.pos <= length(a.ename)
 order by 1,2
       ) x
```

The following select statement leverages the view:

```
select ename,
       max(case when pos=1 then
                case when cnt=1 then c
                     else rpad(c,cast(cnt as integer),c)
                end
                else ''
           end)||
       max(case when pos=2 then
                case when cnt=1 then c
                     else rpad(c,cast(cnt as integer),c)
                end
                else ''
           end)||
       max(case when pos=3 then
                case when cnt=1 then c
                     else rpad(c,cast(cnt as integer),c)
                end
                else ''
           end)||
       max(case when pos=4 then
                case when cnt=1 then c
                     else rpad(c,cast(cnt as integer),c)
                end
                else ''
           end)||
       max(case when pos=5 then
                case when cnt=1 then c
                     else rpad(c,cast(cnt as integer),c)
                end
                else ''
           end)||
       max(case when pos=6 then
                case when cnt=1 then c
                     else rpad(c,cast(cnt as integer),c)
                end
                else ''
           end)
  from (
select a.ename, a.c,
       (select count(*)
          from v b
         where a.ename=b.ename and a.c=b.c ) as cnt,
       (select count(*)+1
          from v b
         where a.ename=b.ename and b.c<a.c) as pos
  from v a
       ) x
 group by ename
```

SQL Server

To alphabetize rows of strings it is necessary to walk each string, and then order their characters:

```
select ename,
          max(case when pos=1 then c else '' end)+
          max(case when pos=2 then c else '' end)+
          max(case when pos=3 then c else '' end)+
          max(case when pos=4 then c else '' end)+
          max(case when pos=5 then c else '' end)+
          max(case when pos=6 then c else '' end)
     from (
   select e.ename,
        substring(e.ename,iter.pos,1) as c,
        row_number() over (
         partition by e.ename
             order by substring(e.ename,iter.pos,1)) as pos
   from emp e,
        (select row_number()over(order by ename) as pos
           from emp) iter
  where iter.pos <= len(e.ename)
         ) x
  group by ename
```

DISCUSSION

DB2 and SQL Server

The inline view X returns each character in each name as a row. The function SUBSTR or SUBSTRING extracts each character from each name, and the function ROW_NUMBER ranks each character alphabetically:

```
ENAME  C  POS
-----  -  ---
ADAMS  A  1
ADAMS  A  2
ADAMS  D  3
ADAMS  M  4
ADAMS  S  5
…
```

To return each letter of a string as a row, you must walk the string. This is accomplished with inline view ITER.

Now that the letters in each name have been alphabetized, the last step is to put those letters back together, into a string, in the order they are ranked. Each letter's position is evaluated by the CASE statements (lines 2–7). If a character is found at a particular position it is then concatenated to the result of the next evaluation (the following CASE statement). Because the aggregate function MAX is

used as well, only one character per position POS is returned, so that only one row per name is returned. The CASE evaluation goes up to the number 6, which is the maximum number of characters in any name in table EMP.

MySQL

The inline view X (lines 3–6) returns each character in each name as a row. The function SUBSTR extracts each character from each name:

```
ENAME  C
_____  _
ADAMS  A
ADAMS  A
ADAMS  D
ADAMS  M
ADAMS  S
…
```

Inline view ITER is used to walk the string. From there, the rest of the work is done by the GROUP_CONCAT function. By specifying an order, the function not only concatenates each letter, it does so alphabetically.

Oracle

The real work is done by inline view X (lines 5–11), where the characters in each name are extracted and put into alphabetical order. This is accomplished by walking the string, then imposing order on those characters. The rest of the query merely glues the names back together.

The tearing apart of names can be seen by executing only inline view X:

```
OLD_NAME           RN C
---------- ---------- -
ADAMS               1 A
ADAMS               2 A
ADAMS               3 D
ADAMS               4 M
ADAMS               5 S
…
```

The next step is to take the alphabetized characters and rebuild each name. This is done with the function SYS_CONNECT_BY_PATH by appending each character to the ones before it:

```
OLD_NAME   NEW_NAME
---------- ---------
ADAMS      A
ADAMS      AA
ADAMS      AAD
ADAMS      AADM
ADAMS      AADMS
…
```

The final step is to keep only the strings that have the same length as the names they were built from.

PostgreSQL
For readability, view V is used in this solution to walk the string. The function SUBSTR, in the view definition, extracts each character from each name so that the view returns:

```
ENAME C
----- -
ADAMS A
ADAMS A
ADAMS D
ADAMS M
ADAMS S
…
```

The view also orders the results by ENAME and by each letter in each name. The inline view X (lines 15–18) returns the names and characters from view V, the number of times each character occurs in each name, and its position (alphabetically):

```
ename | c | cnt | pos
------+---+-----+-----
ADAMS | A |   2 |   1
ADAMS | A |   2 |   1
ADAMS | D |   1 |   3
ADAMS | M |   1 |   4
ADAMS | S |   1 |   5
```

The extra columns CNT and POS, returned by the inline view X, are crucial to the solution. POS is used to rank each character and CNT is used to determine the number of times the character exists in each name. The final step is to evaluate the position of each character and rebuild the name. You'll notice that each case statement is actually two case statements. This is to determine whether or not a character occursmore than once in a name; if it does, then rather than return that character, what is returned is that character appended to itself CNT times. The aggregate function, MAX, is used to ensure there is only one row per name.

6.13. Identifying Strings That Can Be Treated as Numbers

PROBLEM

You have a column that is defined to hold character data. Unfortunately, the rows contain mixed numeric and character data. Consider view V:

```
create view V as
select replace(mixed,' ','') as mixed
  from (
select substr(ename,1,2)||
       cast(deptno as char(4))||
       substr(ename,3,2) as mixed
  from emp
 where deptno = 10
 union all
select cast(empno as char(4)) as mixed
  from emp
 where deptno = 20
 union all
select ename as mixed
  from emp
 where deptno = 30
       ) x
select * from v

 MIXED
 --------------
 CL10AR
 KI10NG
 MI10LL
 7369
 7566
 7788
 7876
 7902
 ALLEN
 WARD
 MARTIN
 BLAKE
 TURNER
 JAMES
```

You want to return rows that are numbers only, or that contain at least one number. If the numbers are mixed with character data, you want to remove the characters and return only the numbers. For the sample data above you want the following result set:

```
   MIXED
--------
      10
      10
      10
    7369
    7566
    7788
    7876
    7902
```

SOLUTION

The functions REPLACE and TRANSLATE are extremely useful for manipulating strings and individual characters. The key is to convert all numbers to a single character, which then makes it easy to isolate and identify any number by referring to a single character.

DB2

Use functions TRANSLATE, REPLACE, and POSSTR to isolate the numeric characters in each row. The calls to CAST are necessary in view V; otherwise, the view will fail to be created due to type conversion errors. You'll need the function REPLACE to remove extraneous white space due to casting to the fixed length CHAR:

```
select mixed old,
       cast(
         case
         when
           replace(
          translate(mixed,'9999999999','0123456789'),'9','') = ''
         then
            mixed
         else replace(
           translate(mixed,
              repeat('#',length(mixed)),
             replace(
              translate(mixed,'9999999999','0123456789'),'9','')),
                     '#','')
         end as integer ) mixed
  from V
 where posstr(translate(mixed,'9999999999','0123456789'),'9') > 0
```

MySQL

The syntax for MySQL is slightly different and will define view V as:

```
create view V as
select concat(
         substr(ename,1,2),
         replace(cast(deptno as char(4)),' ',''),
         substr(ename,3,2)
       ) as mixed
  from emp
 where deptno = 10
 union all
select replace(cast(empno as char(4)), ' ', '')
  from emp where deptno = 20
 union all
select ename from emp where deptno = 30
```

Because MySQL does not support the TRANSLATE function, you must walk each row and evaluate it on a character-by-character basis.

```
select cast(group_concat(c order by pos separator '') as unsigned)
       as MIXED1
  from (
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
       ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
   and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
       ) y
 group by mixed
 order by 1
```

Oracle

Use functions TRANSLATE, REPLACE, and INSTR to isolate the numeric characters in each row. The calls to CAST are not necessary in view V. Use the function REPLACE to remove extraneous white space due to casting to the fixed length CHAR. If you decide you would like to keep the explicit type conversion calls in the view definition, it is suggested you cast to VARCHAR2:

```
select to_number (
        case
        when
           replace(translate(mixed,'0123456789','9999999999'),'9')
          is not null
        then
             replace(
           translate(mixed,
             replace(
           translate(mixed,'0123456789','9999999999'),'9'),
                     rpad('#',length(mixed),'#')),'#')
        else
             mixed
        end
        ) mixed
  from V
 where instr(translate(mixed,'0123456789','9999999999'),'9') > 0
```

PostgreSQL

Use functions TRANSLATE, REPLACE, and STRPOS to isolate the numeric characters in each row. The calls to CAST are not necessary in view V. Use the function REPLACE ito remove extraneous white space due to casting to the fixed length CHAR. If you decide you would like to keep the explicit type conversion calls in the view definition, it is suggested you cast to VARCHAR:

```
select cast(
       case
       when
        replace(translate(mixed,'0123456789','9999999999'),'9','')
        is not null
       then
          replace(
       translate(mixed,
          replace(
       translate(mixed,'0123456789','9999999999'),'9',''),
                rpad('#',length(mixed),'#')),'#','')
       else
         mixed
       end as integer ) as mixed
   from V
 where strpos(translate(mixed,'0123456789','9999999999'),'9') > 0
```

SQL Server

The built-in function ISNUMERIC along with a wildcard search allows you to easily identify strings that contains numbers, but getting numeric characters out of a string is not particularly efficient because the TRANSLATE function is not supported.

DISCUSSION

The TRANSLATE function is very useful here as it allows you to easily isolate and identify numbers and characters. The trick is to convert all numbers to a single character; this way, rather than searching for different numbers you only search for one character.

DB2, Oracle, and PostgreSQL
The syntax differs slightly among these DBMSs, but the technique is the same. I'll use the solution for PostgreSQL for the discussion.

The real work is done by functions TRANSLATE and REPLACE. To get the final result set requires several function calls, each listed below in one query:

```
select mixed as orig,
translate(mixed,'0123456789','9999999999') as mixed1,
replace(translate(mixed,'0123456789','9999999999'),'9','') as mixed2,
 translate(mixed,
 replace(
 translate(mixed,'0123456789','9999999999'),'9',''),
          rpad('#',length(mixed),'#')) as mixed3,
 replace(
 translate(mixed,
 replace(
translate(mixed,'0123456789','9999999999'),'9',''),
          rpad('#',length(mixed),'#')),'#','') as mixed4
  from V
 where strpos(translate(mixed,'0123456789','9999999999'),'9') > 0
```

```
  ORIG  | MIXED1 | MIXED2 | MIXED3 | MIXED4 | MIXED5
--------+--------+--------+--------+--------+--------
 CL10AR | CL99AR | CLAR   | ##10## | 10     |     10
 KI10NG | KI99NG | KING   | ##10## | 10     |     10
 MI10LL | MI99LL | MILL   | ##10## | 10     |     10
 7369   | 9999   |        | 7369   | 7369   |   7369
 7566   | 9999   |        | 7566   | 7566   |   7566
 7788   | 9999   |        | 7788   | 7788   |   7788
 7876   | 9999   |        | 7876   | 7876   |   7876
 7902   | 9999   |        | 7902   | 7902   |   7902
```

First, notice that any rows without at least one number are removed. How this is accomplished will become clear as you examine each of the columns in the above result set. The rows that are kept are the values in the ORIG column and are the rows that will eventually make up the result set. The first step to extracting the numbers is to use the function TRANSLATE to convert any number to a 9 (you can use any digit; 9 is arbitrary), this is represented by the values in MIXED1. Now that all numbers are 9's, they can be treating as a single unit. The next step is to remove all of the numbers by using the function REPLACE. Because all digits are now 9, REPLACE simply looks for any 9's and removes

them. This is represented by the values in MIXED2. The next step, MIXED3, uses values that are returned by MIXED2. These values are then compared to the values in ORIG. If any characters from MIXED2 are found in ORIG, they are converted to the # character by TRANSLATE. The result set from MIXED3 shows that the letters, not the numbers, have now been singled out and converted to a single character. Now that all non-numeric characters are represented by #'s, they can be treated as a single unit. The next step, MIXED4, uses REPLACE to find and remove any # characters in each row; what's left are numbers only. The final step is to cast the numeric characters as numbers. Now that you've gone through the steps, you can see how the WHERE clause works. The results from MIXED1 are passed to STRPOS, and if a 9 is found (the position in the string where the first 9 is located) the result must be greater than 0. For rows that return a value greater than zero, it means there's at least one number in that row and it should be kept.

MySQL

The first step is to walk each string and evaluate each character and determine whether or not it's a number:

```
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
      ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
 order by 1,2
```

```
+--------+------+------+
| mixed  | pos  | c    |
+--------+------+------+
| 7369   |    1 | 7    |
| 7369   |    2 | 3    |
| 7369   |    3 | 6    |
| 7369   |    4 | 9    |
…
| ALLEN  |    1 | A    |
| ALLEN  |    2 | L    |
| ALLEN  |    3 | L    |
| ALLEN  |    4 | E    |
| ALLEN  |    5 | N    |
…
| CL10AR |    1 | C    |
| CL10AR |    2 | L    |
| CL10AR |    3 | 1    |
| CL10AR |    4 | 0    |
| CL10AR |    5 | A    |
| CL10AR |    6 | R    |
+--------+------+------+
```

Now that each character in each string can be evaluated individually, the next step is to keep only the rows that have a number in the C column:

```
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
   from V,
       ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
  and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
 order by 1,2
```

```
+--------+------+------+
| mixed  | pos  | c    |
+--------+------+------+
| 7369   |    1 | 7    |
| 7369   |    2 | 3    |
| 7369   |    3 | 6    |
| 7369   |    4 | 9    |
…
| CL10AR |    3 | 1    |
| CL10AR |    4 | 0    |
…
+--------+------+------+
```

At this point, all the rows in column C are numbers. The next step is to use GROUP_CONCAT to concatenate the numbers to form their respective whole number in MIXED. The final result is then cast as a number:

```
select cast(group_concat(c order by pos separator '') as unsigned)
         as MIXED1
  from (
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
      ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
  and ascii(substr(x.mixed,iter.pos,1)) between 48 and 57
       ) y
  group by mixed
  order by 1

+--------+
| MIXED1 |
+--------+
|    10  |
|    10  |
|    10  |
|  7369  |
|  7566  |
|  7788  |
|  7876  |
|  7902  |
+--------+
```

As a final note, keep in mind that any digits in each string will be concatenated to form one numeric value. For example, an input value of, say, ’99Gennick87’ will result in the value 9987 being returned. This is something to keep in mind, particularly when working with serialized data.

6.14. Extracting the nth Delimited Substring

PROBLEM

You want to extract a specified, delimited substring from a string. Consider the following view V, which generates source data for this problem:

```
create view V as
select 'mo,larry,curly' as name
  from t1
 union all
select 'tina,gina,jaunita,regina,leena' as name
  from t1
```

Output from the view is as follows:

```
select * from v

NAME
------------------
mo,larry,curly
tina,gina,jaunita,regina,leena
```

You would like to extract the second name in each row, so the final result set would be:

```
  SUB
-----
larry
gina
```

SOLUTION

The key to solving this problem is to return each name as an individual row while preserving the order in which the name exists in the list. Exactly how you do these things depends on which DBMS you are using.

DB2

After walking the NAMEs returned by view V, use the function ROW_NUMBER to keep only the second name from each string:

```
select substr(c,2,locate(',',c,2)-2)
  from (
select pos, name, substr(name, pos) c,
       row_number() over( partition by name
                      order by length(substr(name,pos)) desc) rn
 from (
select ',' ||csv.name|| ',' as name,
        cast(iter.pos as integer) as pos
  from V csv,
       (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
       ) x
 where length(substr(name,pos)) > 1
  and substr(substr(name,pos),1,1) = ','
       ) y
 where rn = 2
```

MySQL

After walking the NAMEs returned by view V, use the position of the commas to return only the second name in each string:

```
select name
  from (
select iter.pos,
       substring_index(
       substring_index(src.name,',',iter.pos),',',-1) name
  from V src,
       (select id pos from t10) iter,
 where iter.pos <=
       length(src.name)-length(replace(src.name,',',''))
       ) x
 where pos = 2
```

Oracle

After walking the NAMEs returned by view V, retrieve the second name in each list by using SUBSTR and INSTR:

```
select sub
  from (
select iter.pos,
       src.name,
       substr( src.name,
        instr( src.name,',',1,iter.pos )+1,
        instr( src.name,',',1,iter.pos+1 ) -
        instr( src.name,',',1,iter.pos )-1) sub
  from (select ','||name||',' as name from V) src,
       (select rownum pos from emp) iter
 where iter.pos < length(src.name)-length(replace(src.name,','))
       )
 where pos = 2
```

PostgreSQL

Use the function SPLIT_PART to help return each individual name as a row:

```
select name
  from (
select iter.pos, split_part(src.name,',',iter.pos) as name
  from (select id as pos from t10) iter,
       (select cast(name as text) as name from v) src
 where iter.pos <=
        length(src.name)-length(replace(src.name,',',''))+1
       ) x
 where pos = 2
```

SQL Server

After walking the NAMEs returned by view V, use the function ROW_NUMBER to keep only the second name from each string:

```
select substring(c,2,charindex(',',c,2)-2)
  from (
select pos, name, substring(name, pos, len(name)) as c,
       row_number() over(
        partition by name
        order by len(substring(name,pos,len(name))) desc) rn
  from (
select ',' + csv.name + ',' as name,
       iter.pos
  from V csv,
       (select id as pos from t100 ) iter
 where iter.pos <= len(csv.name)+2
       ) x
 where len(substring(name,pos,len(name))) > 1
   and substring(substring(name,pos,len(name)),1,1) = ','
       ) y
 where rn = 2
```

DISCUSSION

DB2 and SQL Server

The syntax is slightly different between these two DBMSs, but the technique is the same. I will use the solution for DB2 for the discussion. The strings are walked and the results are represented by inline view X:

```
select ','||csv.name|| ',' as name,
       iter.pos
  from v csv,
       (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2

EMPS                           POS
------------------------------ ----
,tina,gina,jaunita,regina,leena,    1
,tina,gina,jaunita,regina,leena,    2
,tina,gina,jaunita,regina,leena,    3
…
```

The next step is to then step through each character in each string:

```
select pos, name, substr(name, pos) c,
        row_number() over(partition by name
                          order by length(substr(name, pos)) desc) rn
  from (
select ','||csv.name||',' as name,
       cast(iter.pos as integer) as pos
  from v csv,
      (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
       ) x
 where length(substr(name,pos)) > 1

POS EMPS            C                RN
--- --------------- ---------------- --
  1 ,mo,larry,curly, ,mo,larry,curly, 1
  2 ,mo,larry,curly, mo,larry,curly,  2
  3 ,mo,larry,curly, o,larry,curly,   3
  4 ,mo,larry,curly, ,larry,curly,    4
  …
```

Now that different portions of the string are available to you, simply identify which rows to keep. The rows you are interested in are the ones that begin with a comma; the rest can be discarded:

```
select pos, name, substr(name,pos) c,
       row_number() over(partition by name
                         order by length(substr(name, pos)) desc) rn
  from (
select ','||csv.name||',' as name,
       cast(iter.pos as integer) as pos
  from v csv,
       (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
       ) x
 where length(substr(name,pos)) > 1
    and substr(substr(name,pos),1,1) = ','

POS  EMPS                              C                                 RN
 ---  --------------                    ----------------                   --
   1  ,mo,larry,curly,                  ,mo,larry,curly,                    1
   4  ,mo,larry,curly,                  ,larry,curly,                       2
  10  ,mo,larry,curly,                  ,curly,                             3
   1  ,tina,gina,jaunita,regina,leena,  ,tina,gina,jaunita,regina,leena,    1
   6  ,tina,gina,jaunita,regina,leena,  ,gina,jaunita,regina,leena,         2
  11  ,tina,gina,jaunita,regina,leena,  ,jaunita,regina,leena,              3
  19  ,tina,gina,jaunita,regina,leena,  ,regina,leena,                      4
  26  ,tina,gina,jaunita,regina,leena,  ,leena,                             5
```

This is an important step as it sets up how you will get the nth substring. Notice that many rows have been eliminated from this query because of the following condition in the WHERE clause:

```
substr(substr(name,pos),1,1) = ','
```

You'll notice that ,larry,curly, was ranked 4, but now is ranked 2. Remember, the WHERE clause is evaluated before the SELECT, so the rows with leading commas are kept, then ROW_NUMBER performs its ranking. At this point it's easy to see that, to get the nth substring you want rows where RN equals n. The last step is to keep only the rows you are interested in (in this case where RN equals 2) and use SUBSTR to extract the name from that row. The name to keep is the first name in the row: larry from ,larry,curly, and gina from ,gina,jaunita,regina,leena,.

MySQL
The inline view X walks each string. You can determine how many values are in each string by counting the delimiters in the string:

```
select iter.pos, src.name
  from (select id pos from t10) iter,
       V src
 where iter.pos <=
       length(src.name)-length(replace(src.name,',',''))
```

```
+------+------------------------------+
| pos  | name                         |
+------+------------------------------+
|    1 | mo,larry,curly               |
|    2 | mo,larry,curly               |
|    1 | tina,gina,jaunita,regina,leena |
|    2 | tina,gina,jaunita,regina,leena |
|    3 | tina,gina,jaunita,regina,leena |
|    4 | tina,gina,jaunita,regina,leena |
+------+------------------------------+
```

In this case, there is one fewer row than values in each string because that's all that is needed. The function SUBSTRING_INDEX takes care of parsing the needed values:

```
select iter.pos,src.name name1,
        substring_index(src.name,',',iter.pos) name2,
        substring_index(
         substring_index(src.name,',',iter.pos),',',-1) name3
   from (select id pos from t10) iter,
        V src
  where  iter.pos <=
        length(src.name)-length(replace(src.name,',',''))
```

```
+------+-----------------------------+-----------------------+---------+
| pos | name1 | name2 | name3 |
+------+-----------------------------+-----------------------+---------+
| 1 | mo,larry,curly | mo | mo |
| 2 | mo,larry,curly | mo,larry | larry |
| 1 | tina,gina,jaunita,regina,leena | tina | tina |
| 2 | tina,gina,jaunita,regina,leena | tina,gina | gina |
| 3 | tina,gina,jaunita,regina,leena | tina,gina,jaunita | jaunita |
| 4 | tina,gina,jaunita,regina,leena | tina,gina,jaunita,regina | regina |
+------+-----------------------------+-----------------------+---------+
```

I've shown three name fields, so you can see how the nested SUBSTRING_INDEX calls work. The inner call returns all characters to the left of the nth occurrence of a comma. The outer call returns everything to the right of the first comma it finds (starting from the end of the string). The final step is to keep the value for NAME3 where POS equals n, in this case 2.

Oracle

The inline view walks each string. The number of times each string is returned is determined by how many values are in each string. The solution finds the number of values in each string by counting the number of delimiters in it. Because each string is enclosed in commas, the number of values in a string is the number of commas minus one. The strings are then UNIONed and joined to a table with a cardinality that is at least the number of values in the largest string. The functions SUBSTR and INSTR use the value of POS to parse each string:

```
select iter.pos, src.name,
       substr( src.name,
        instr( src.name,',',1,iter.pos )+1,
        instr( src.name,',',1,iter.pos+1 )
        instr( src.name,',',1,iter.pos )-1) sub
  from (select ','||name||',' as name from v) src,
       (select rownum pos from emp) iter
 where iter.pos < length(src.name)-length(replace(src.name,','))

POS NAME                            SUB
--- ------------------------------- -------------
  1 ,mo,larry,curly,                mo
  1 , tina,gina,jaunita,regina,leena, tina
  2 ,mo,larry,curly,                larry
  2 , tina,gina,jaunita,regina,leena, gina
  3 ,mo,larry,curly,                curly
  3 , tina,gina,jaunita,regina,leena, jaunita
  4 , tina,gina,jaunita,regina,leena, regina
  5 , tina,gina,jaunita,regina,leena, leena
```

The first call to INSTR within SUBSTR determines the start position of the substring to extract. The next call to INSTR within SUBSTR finds the position of the nth comma (same as the start position) as well the position of the nth + 1 comma. Subtracting the two values returns the length of the substring to extract. Because every value is parsed into its own row, simply specify WHERE POS = n to keep the nth substring (in this case, where POS = 2, so, the second substring in the list).

PostgreSQL
The inline view X walks each string. The number of rows returned is determined by how many values are in each string. To find the number of values in each string, find the number of delimiters in each string and add one. The function SPLIT_PART uses the values in POS to find the nth occurrence of the delimiter and parse the string into values:

```
select iter.pos, src.name as name1,
       split_part(src.name,',',iter.pos) as name2
   from (select id as pos from t10) iter,
        (select cast(name as text) as name from v) src
  where iter.pos <=
        length(src.name)-length(replace(src.name,',',''))+1

 pos |             name1              |  name2
-----+--------------------------------+---------
   1 | mo,larry,curly                 | mo
   2 | mo,larry,curly                 | larry
   3 | mo,larry,curly                 | curly
   1 | tina,gina,jaunita,regina,leena | tina
   2 | tina,gina,jaunita,regina,leena | gina
   3 | tina,gina,jaunita,regina,leena | jaunita
   4 | tina,gina,jaunita,regina,leena | regina
   5 | tina,gina,jaunita,regina,leena | leena
```

I've shown NAME twice so you can see how SPLIT_PART parses each string using POS. Once each string is parsed, the final step is the keep the rows where POS equals the nth substring you are interested in, in this case, 2.

6.15. Parsing an IP Address

PROBLEM

You want to parse an IP address's fields into columns. Consider the following IP address:

```
111.22.3.4
```

You would like the result of your query to be:

```
A     B    C      D
----- ----- ----- ---
111   22    3     4
```

SOLUTION

The solution depends on the built-in functions provided by your DBMS. Regardless of your DBMS, being able to locate periods and the numbers immediately surrounding them are the keys to the solution.

DB2

Use the recursive WITH clause to simulate an iteration through the IP address while using SUBSTR to easily parse it. A leading period is added to the IP address so that every set of numbers has a period in front of it and can be treated the same way.

```
with x (pos,ip) as (
  values (1,'.92.111.0.222')
  union all
 select pos+1,ip from x where pos+1 <= 20
)
select max(case when rn=1 then e end) a,
       max(case when rn=2 then e end) b,
       max(case when rn=3 then e end) c,
       max(case when rn=4 then e end) d
  from (
select pos,c,d,
       case when posstr(d,'.') > 0 then substr(d,1,posstr(d,'.')-1)
            else d
       end as e,
       row_number() over( order by pos desc) rn
  from (
select pos, ip,right(ip,pos) as c, substr(right(ip,pos),2) as d
  from x
 where pos <= length(ip)
  and substr(right(ip,pos),1,1) = '.'
     ) x
     ) y
```

MySQL

The function SUBSTR_INDEX makes parsing an IP address an easy operation:

```
select substring_index(substring_index(y.ip,'.',1),'.',-1) a,
       substring_index(substring_index(y.ip,'.',2),'.',-1) b,
       substring_index(substring_index(y.ip,'.',3),'.',-1) c,
       substring_index(substring_index(y.ip,'.',4),'.',-1) d
  from (select '92.111.0.2' as ip from t1) y
```

Oracle

Use the built-in function SUBSTR and INSTR to parse and navigate through the IP address:

```
select ip,
       substr(ip, 1, instr(ip,'.')-1 ) a,
       substr(ip, instr(ip,'.')+1,
                   instr(ip,'.',1,2)-instr(ip,'.')-1 ) b,
       substr(ip, instr(ip,'.',1,2)+1,
                   instr(ip,'.',1,3)-instr(ip,'.',1,2)-1 ) c,
       substr(ip, instr(ip,'.',1,3)+1 ) d
  from (select '92.111.0.2' as ip from t1)
```

PostgreSQL

Use the built-in function SPLIT_PART to parse an IP address:

```
select split_part(y.ip,'.',1) as a,
       split_part(y.ip,'.',2) as b,
       split_part(y.ip,'.',3) as c,
       split_part(y.ip,'.',4) as d
  from (select cast('92.111.0.2' as text) as ip from t1) as y
```

SQL Server

Use the recursive WITH clause to simulate an iteration through the IP address while using SUBSTR to easily parse it. A leading period is added to the IP address so that every set of numbers has a period in front of it and can be treated the same way:

```
 with x (pos,ip) as (
   select 1 as pos,'.92.111.0.222' as ip from t1
   union all
  select pos+1,ip from x where pos+1 <= 20
 )
 select max(case when rn=1 then e end) a,
        max(case when rn=2 then e end) b,
        max(case when rn=3 then e end) c,
        max(case when rn=4 then e end) d
   from  (
 select pos,c,d,
        case when charindex('.',d) > 0
           then substring(d,1,charindex('.',d)-1)
           else d
       end as e,
       row_number() over(order by pos desc) rn
  from (
select pos, ip,right(ip,pos) as c,
       substring(right(ip,pos),2,len(ip)) as d
  from x
 where pos <= len(ip)
   and substring(right(ip,pos),1,1) = '.'
      ) x
      ) y
```

DISCUSSION

By using the built-in functions for your database, you can easily walk through parts of a string. The key is being able to locate each of the periods in the address. Then you can parse the numbers between each.