```
/* ****************************************************************************
          INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
                    WS23SQL1001, 2023/04/03 to 2023/05/03
                  https://folvera.commons.gc.cuny.edu/?cat=33
 ****************************************************************************

  SESSION #3 (2023/04/10): MANIPULATING DATA

  1. Using built-in functions for strings
  2. Querying two or more datasets (tables or views) using `INNER JOIN`,
     `LEFT [OUTER] JOIN` and `RIGHT [OUTER] JOIN`
 ****************************************************************************

 1. LAB 1
    Write a query calling all shared rows/records (`INNER JOIN`) from
    `AP1.Invoices`, `AP1.Terms` and `AP1.Vendors`.
    * Delete or rename the duplicate name of the columns.
 ************************************************************************** */

SELECT AP1.Invoices.InvoiceID,
  AP1.Invoices.VendorID,
  AP1.Invoices.InvoiceNumber,
  AP1.Invoices.InvoiceDate,
  AP1.Invoices.InvoiceTotal,
  AP1.Invoices.PaymentTotal,
  AP1.Invoices.CreditTotal,
  AP1.Invoices.TermsID,
  AP1.Invoices.InvoiceDueDate,
  AP1.Invoices.PaymentDate,
  -- AP1.Terms.TermsID,                       -- 1. duplicate column name
                                              --    (`TermsID`), which can
                                              --    be removed (commented
                                              --    out, in this case)
                                              --    without affecting the
                                              --    query output; could also
                                              --    be renamed
  AP1.Terms.TermsDescription,
  AP1.Terms.TermsDueDays,
  -- AP1.Vendors.VendorID,                    -- 2. duplicate column name
                                              --    (`VendorID`), which can
                                              --    be removed (commented
                                              --    out, in this case)
                                              --    without affecting the
                                              --    query output; could also
                                              --    be renamed
  AP1.Vendors.VendorName,
  AP1.Vendors.VendorAddress1,
  AP1.Vendors.VendorAddress2,
  AP1.Vendors.VendorCity,
  AP1.Vendors.VendorState,
  AP1.Vendors.VendorZipCode,
  AP1.Vendors.VendorPhone,
```

```
53    AP1.Vendors.VendorContactLName,
54    AP1.Vendors.VendorContactFName,
55    AP1.Vendors.DefaultTermsID,
56    AP1.Vendors.DefaultAccountNo
57  FROM AP1.Invoices                                 -- 3. from table `AP1.Invoices`
58  INNER JOIN AP1.Terms                              -- 4. `INNER JOIN` to retrieve
59                                                    --    data in the first (left)
60                                                    --    table (`AP1.Invoices`)
61                                                    --    that is also in the
62                                                    --    second (right) table
63                                                    --    (`AP1.Terms`)
64    ON AP1.Invoices.TermsID = AP1.Terms.TermsID     -- 5. `ON` two fields with the
65                                                    --    same values/data and the
66                                                    --    same name (`TermsID`);
67                                                    --    specifying the relation
68                                                    --    between tables
69                                                    --    `AP1.Invoices` and
70                                                    --    `AP1.Terms`
71  INNER JOIN AP1.Vendors                            -- 6. `INNER JOIN` to retrieve
72                                                    --    data in the second (left)
73                                                    --    table (`AP1.Terms`) that
74                                                    --    is also in the third
75                                                    --    (right) table
76                                                    --    (`AP1.Vendors`)
77    ON AP1.Vendors.VendorID = AP1.Invoices.VendorID
78                                                    -- 7. `ON` two fields with the
79                                                    --    same values/data and the
80                                                    --    same name (`VendorID`);
81                                                    --    specifying the relation
82                                                    --    between tables
83                                                    --    `AP1.Vendors` and
84                                                    --    `AP1.Invoices`
85      AND AP1.Vendors.DefaultTermsID = AP1.Terms.TermsID;
86                                                    -- 7. `AND` two other fields
87                                                    --    with the same values/data
88                                                    --    and different names
89                                                    --    (`DefaultTermsID` in
90                                                    --    `AP1.Vendors`, which has
91                                                    --    the same data as
92                                                    --    `TermsID` in
93                                                    --    `AP1.Terms`); specifying
94                                                    --    the relation between
95                                                    --    tables `AP1.Vendors` and
96                                                    --    `AP1.Terms`
97
98
99  /* ***************************************************************************
100     * As an alternative, the code above can also be written using an alias
101       (`AS`) for each table in order to simplify the code.
102
103                     `i` for `AP1.Invoices`
104                     `t` for `AP1.Terms`
```

```
105                     `v` for `AP1.Vendors`
106
107       Note that, if you use an alias (`AS`) for a table (for example, `v` for
108       `AP1.Vendors`), you must use the alias (`v`) when calling the table
109       (`AP1.Vendors`) in the query (for example, calling `AP1.Vendors.VendorID`
110       as `v.VendorID`).
111   ********************************************************************** */
112
113  SELECT i.InvoiceID,
114    i.VendorID,
115    i.InvoiceNumber,
116    i.InvoiceDate,
117    i.InvoiceTotal,
118    i.PaymentTotal,
119    i.CreditTotal,
120    i.TermsID,
121    i.InvoiceDueDate,
122    i.PaymentDate,
123    t.TermsDescription,
124    t.TermsDueDays,
125    v.VendorName,
126    v.VendorAddress1,
127    v.VendorAddress2,
128    v.VendorCity,
129    v.VendorState,
130    v.VendorZipCode,
131    v.VendorPhone,
132    v.VendorContactLName,
133    v.VendorContactFName,
134    v.DefaultTermsID,
135    v.DefaultAccountNo
136  FROM AP1.Invoices AS i
137  INNER JOIN AP1.Terms AS t
138    ON i.TermsID = t.TermsID
139  INNER JOIN AP1.Vendors AS v
140    ON v.VendorID = i.VendorID
141      AND v.DefaultTermsID = t.TermsID;
142
143
144  /* **********************************************************************
145   2. A function, in any programming environment, lets you encapsulate reusable
146      logic and build software that is ``composable``, i.e. built of pieces that
147      can be reused and put together in a number of different ways to meet the
148      needs of the users.  Functions hide the steps and the complexity from other
149      code.
150      https://www.simple-talk.com/sql/t-sql-programming/sql-server-functions-the-
          basics/
151
152      Go to https://techonthenet.com/sql_server/functions/index_alpha.php for a
153      detailed list of functions.
154
155      As we mentioned before, so functions affect strings.
```

```
156
157          CONCAT()   allows you to concatenate strings together
158                     https://techonthenet.com/sql_server/functions/concat.php
159
160          `+`        also allows you to concatenate strings together although
161                     adding NULL returns a NULL
162                     https://techonthenet.com/sql_server/functions/concat2.php
163
164          LEFT()     allows you to extract a substring from a string, starting
165                     from the left-most character
166                     https://techonthenet.com/sql_server/functions/left.php
167
168          LEN()      returns the length of the specified string... does not
169                     include trailing space characters at the end the string
170                     when calculating the length
171                     https://techonthenet.com/sql_server/functions/len.php
172
173          LTRIM()    removes all space characters from the left-hand side of a
174                     string
175                     https://techonthenet.com/sql_server/functions/ltrim.php
176
177          LOWER()    converts all letters in the specified string to lowercase
178                     https://techonthenet.com/sql_server/functions/lower.php
179
180          REPLACE()  replaces a sequence of characters in a string with another
181                     set of characters, not case-sensitive
182                     https://techonthenet.com/sql_server/functions/replace.php
183
184          RIGHT()    allows you to extract a substring from a string, starting
185                     from the right-most character
186                     https://techonthenet.com/sql_server/functions/right.php
187
188          RTRIM()    removes all space characters from the right-hand side of a
189                     string
190                     https://techonthenet.com/sql_server/functions/rtrim.php
191
192          SUBSTRING() allows you to extract a substring from a string
193                     https://techonthenet.com/sql_server/functions/substring.php
194
195          UPPER()    converts all letters in the specified string to uppercase
196                     https://techonthenet.com/sql_server/functions/upper.php
197
198      We also have functions that affect numeric values.
199
200          AVG()      returns the average value of an expression
201                     https://techonthenet.com/sql_server/functions/avg.php
202
203          CEILING()  returns the smallest integer value that is greater than or
204                     equal to a number
205                     https://techonthenet.com/sql_server/functions/ceiling.php
206
207          COUNT()    returns the count of an expression
```

```
                    https://techonthenet.com/sql_server/functions/count.php

          FLOOR()   returns the largest integer value that is equal to or less
                    than a number
                    https://techonthenet.com/sql_server/functions/floor.php

          LEN()     returns the length of the specified string... does not
                    include trailing space characters at the end the string
                    when calculating the length
                    https://techonthenet.com/sql_server/functions/len.php

          MAX()     returns the maximum value of an expression
                    https://techonthenet.com/sql_server/functions/max.php

          MIN()     returns the minimum value of an expression
                    https://techonthenet.com/sql_server/functions/min.php

          RAND()    returns a random number or a random number within a range
                    https://techonthenet.com/sql_server/functions/rand.php

          ROUND()   returns a number rounded to a certain number of decimal
                    places
                    https://techonthenet.com/sql_server/functions/round.php

          SUM()     returns the summed value of an expression
                    https://techonthenet.com/sql_server/functions/sum.php

    Note that every time you have a function, you need parenthesis.  Go to
    https://techonthenet.com/sql_server/functions/index_alpha.php for a
    complete list of built-in functions.

    As you might have noticed, some built-in functions manipulate strings.
    When working with numerical values, first we would have to convert them
    into strings as we will see later in the course.

    Some other built-in functions ``return a single value, calculated from
    values in a column``.  These are referred to as aggregate functions
    (https://msdn.microsoft.com/en-us/library/ms173454.aspx).

 2. Understanding the concepts above, we can now use them.

    2.01. In the example below, we concatenate (put strings together) columns
          `FirstName` and `LastName` from table `AP1.ContactUpdates`.
 ********************************************************************** */

SELECT CONCAT (
    FirstName,
    ' ',
    LastName
    ) AS NAME
FROM AP1.ContactUpdates;

```

```
260
261 /* ************************************************************************
262     2.02. In the example below, we concatenate (put strings together) columns
263           `WE `, `ARE `, `LEARNING `, `SQL!` and print the result to the
264           console.
265  ************************************************************************ */
266
267 PRINT CONCAT('WE ', 'ARE ', 'LEARNING ', 'SQL!');
268                                                 -- returns `WE ARE LEARNING
269                                                 --   SQL!`
270
271
272 /* ************************************************************************
273     2.03. In the example below, we concatenate (put strings together) columns
274           `FirstName` and `LastName` from table `AP1.ContactUpdates`, just like
275           the previous example.
276
277           We also use `LTRIM()` and `RTRIM()` to remove leading and trailing
278           spaces from `FirstName` with `LTRIM(RTRIM(FirstName))` and `LastName`
279           with `LTRIM(RTRIM(LastName))`.
280  ************************************************************************ */
281
282 SELECT CONCAT (
283     LTRIM(RTRIM(LastName)),
284     ', ',
285     LTRIM(RTRIM(FirstName))
286     ) AS NAME
287 FROM AP1.ContactUpdates;
288
289
290 /* ************************************************************************
291     2.04. In the examples below, we use `UPPER()` to change a string to upper
292           case and print the result to console.
293  ************************************************************************ */
294
295 PRINT UPPER('this string is in upper case');    -- returns `THIS STRING SHOULD
296                                                 --   IN UPPER CASE`
297
298
299 /* ************************************************************************
300     2.05. In the examples below, we use `LOWER()` to change a string to lower
301           case.
302  ************************************************************************ */
303
304 PRINT LOWER('BUT THIS STRING IS IN LOWER CASE.');
305                                                 -- returns `but this string is
306                                                 --   in lower case.`
307
308
309 /* ************************************************************************
310     2.06. In the examples below, we use `RIGHT()` to extract characters from
311           the right.
```

```
312  ********************************************************************** */
313
314 PRINT RIGHT('apple', 2);                          -- returns `le`
315
316
317 /* ***************************************************************************
318     2.07. In the examples below, we use `LEFT()` to extract characters from the
319           left.
320  ********************************************************************** */
321
322 PRINT LEFT('apple', 2);                           -- returns `ap`
323
324
325 /* ***************************************************************************
326     2.08. In the examples below, we use `SUBSTRING()` to extract characters
327           from the middle -- same as the built-in function `MID()` in other
328           relational database management systems (RDBMS) like Oracle -- and
329           print the result to the console
330  ********************************************************************** */
331
332 PRINT SUBSTRING('apple tree #5', 6, 10);          -- returns ` tree #5`
333
334
335 /* ***************************************************************************
336     2.09. In the example below, we use `LEN()` to retrieve the length of a
337           string.
338  ********************************************************************** */
339
340 PRINT LEN('tree      #5');                        -- returns 12
341
342
343 /* ***************************************************************************
344     2.10. In the examples below, we use `LTRIM()` and `RTRIM()` to remove any
345           leading and/or trailing spaces from the strings in single quotes and
346           print the result to the console.
347
348           We could also use function `TRIM()` only in SQL Server
349           (https://docs.microsoft.com/en-us/sql/t-sql/functions/trim-transact-
               sql).
350  ********************************************************************** */
351
352 PRINT LTRIM('       tree'),                       -- 1. trimming leading spaces
353   RTRIM('tree       '),                         -- 2. trimming trailing spaces
354   LTRIM(RTRIM('       tree       '));            -- 3. trimming leading and
355                                                  --    trailing spaces
356
357
358 /* ***************************************************************************
359     2.11. In the example below, we use `REPLACE()` to replace pattern `mstake`
360           with `mistake`.  Since `mstake`
361           exists in string `This is a mstake`, `REPLACE()` returns `This is a
362           mistake`.
```

```
363  ********************************************************************** */
364
365  PRINT REPLACE('This is a mstake', 'mstake', 'mistake');
366                                                  -- returns `This is a mistake`
367
368
369  /* ********************************************************************
370           In the example below, we use `REPLACE()` to replace pattern `gg` with
371           `mistake`.  Since `gg` does not
372           exist in `This is a mstake`, `REPLACE()` returns the original value.
373   ********************************************************************** */
374
375  PRINT REPLACE('This is a mstake', 'gg', 'mistake');
376                                                  -- returns `This is a mstake`
377
378
379  /* ********************************************************************
380      2.12. In the example below, since there is no function to make the first
381           letter of a string upper case and the rest lower case, we can use
382           a combination of functions `UPPER()`, `LOWER()`, `RIGHT()`, `LEFT()`
383           and `CONCAT()` working from the inside out and print the result to
384           the console
385   ********************************************************************** */
386
387  PRINT CONCAT (
388      UPPER(LEFT('hELLO', 1)                      -- 1. retrieving first
389                                                  --    character from `hELLO`;
390                                                  --    returns `h`
391      )                                           -- 2. making `h` upper case;
392                                                  --    returns `H`
393      ,
394      LOWER(SUBSTRING('hELLO', 2, LEN('hELLO')    -- 3. retrieving variable
395        )                                         --    number of characters
396                                                  --    from character two (2)
397                                                  --    to the length of the
398                                                  --    string (integer value
399                                                  --    of 5); returns `ELLO`
400      )                                           -- 4. making `ELLO` lower
401                                                  --    case; returns `ello`
402      );                                          -- 5. concatenating all
403                                                  --    previous sections;
404                                                  --    returns `Hello`
405
406
407  /* ********************************************************************
408      2.13. In the example below, we use `REPLACE()` to change pattern `  ` (two
409           spaces, `CHAR(32)+CHAR(32))` with ` ` (a single space, `CHAR(32)`).
410
411                    PRINT REPLACE('tree      #5', '  ', ' ');
412
413           Since string `tree      #5` has more than two spaces, we need run
414           several passes of `REPLACE()`.
```

```
415
416           The statement runs from the inside out (3, 2, 1, 2, 3).
417
418                     function 3                  -- 3. beginning of function #3:
419                                                 --    * receiving value of
420                                                 --      function #2
421                       function 2                -- 2. beginning of function #2:
422                                                 --    * receiving value of
423                                                 --      function #1
424                         function 1              -- 1. function #1:
425                                                 --    * receiving original
426                                                 --      value #0
427                                                 --    * returning new value #1
428                       function 2                -- 2. end function of #2:
429                                                 --    * returning new value #2
430                     function 3                  -- 3. end function of #3:
431                                                 --    * returning new value #3
432                                                 --     (final value)
433   ************************************************************************ */
434
435 PRINT
436   REPLACE(                                      -- 3. pass #3 to replace
437                                                 --    `CHAR(32)+CHAR(32)` for
438                                                 --    `CHAR(32)`
439                                                 --    * returns `tree #5` with
440                                                 --      1 space
441     REPLACE(                                    -- 2. pass #2 to replace
442                                                 --    `CHAR(32)+CHAR(32)` for
443                                                 --    `CHAR(32)`
444                                                 --    * returns `tree  #5` with
445                                                 --      2 space, which feeds
446                                                 --      pass #3
447       REPLACE('tree      #5',                   -- 1. pass #1 to replace
448                                                 --    `CHAR(32)+CHAR(32)` for
449                                                 --    `CHAR(32)`
450                                                 --    * returns `tree   #5`
451                                                 --      with 3 spaces, which
452       '  ', ' '),                               --      feeds pass #2
453     '  ', ' '),                                 -- 2. end of pass #2
454   '  ', ' ');                                   -- 3. end of pass #3
455
456
457 /* ************************************************************************
458     2.13. In the example below, we use `REPLACE()` to replace pattern `tree`
459           for `fruit`.
460
461           Since pattern `tree` exists in `      tree     ` with leading and
462           trailing spaces around `tree`, `REPLACE()` returns `      fruit     `
463           with leading and trailing spaces around word `fruit`.
464
465           We also use `RTRIM()` and `LTRIM()` to remove trailing and leading
466           spaces respectively to get `fruit` without leading and trailing
```

```
467           spaces.
468    ********************************************************************** */
469
470  PRINT RTRIM(LTRIM(REPLACE('      tree     ', 'tree', 'fruit')));
471                                                -- returns `      fruit     `
472
473
474  /* *********************************************************************
475      2.14. In the example below, we use `REPLACE()` to replace pattern `Box` for
476            `PO Box`.  The first pass (inner) of `REPLACE()` changes some fields
477            to `PO PO Box`.  The second pass (outer) of `REPLACE()` changes the
478            previous logical error (`PO PO Box`) to `PO Box`.
479    ********************************************************************** */
480
481  SELECT AP1.Vendors.VendorID,                    -- 1. fields using format
482    AP1.Vendors.VendorName,                       --    `schema.table.field`
483    REPLACE(                                      -- 2. second pass of
484                                                  --    `REPLACE()` working from
485                                                  --    inside out
486      REPLACE(AP1.Vendors.VendorAddress1,         -- 3. first pass of `REPLACE()`
487      'Box', 'PO Box'),                           --    working from inside out
488    'PO PO Box', 'PO Box') AS VendorAddress1,
489    AP1.Vendors.VendorAddress2,
490    AP1.Vendors.VendorCity,
491    AP1.Vendors.VendorState,
492    AP1.Vendors.VendorZipCode,
493    REPLACE(                                      -- 4. replacing `() -` from the
494                                                  --    concatenation in #5
495                                                  --    instead of a CASE clause
496                                                  --    (logic block), which we
497                                                  --    will cover later
498      CONCAT (                                    -- 5. concatenating an opening
499        '(',                                      --    parenthesis, first three
500        LEFT(Vendors.VendorPhone, 3),             --    characters of
501        ') ',                                     --    `VendorPhone`, a closing
502        SUBSTRING(Vendors.VendorPhone, 4, 3),     --    parenthesis with a space,
503        '-',                                      --    the substring of
504        RIGHT(Vendors.VendorPhone, 4)             --    `VendorPhone` starting
505        ), '() -', '') AS VendorPhone             --    from the fourth character
506                                                  --    and taking 3, a hyphen
507                                                  --    and the last four
508                                                  --    characters of `VendorPhone`
509    AP1.Vendors.VendorContactLName,
510    AP1.Vendors.VendorContactFName,
511    AP1.Vendors.DefaultTermsID,
512    AP1.Vendors.DefaultAccountNo,
513    AP1.Terms.TermsID,
514    AP1.Terms.TermsDescription,
515    AP1.Terms.TermsDueDays
516  FROM AP1.Vendors
517  INNER JOIN AP1.Terms                            -- 6. `INNER JOIN` to retrieve
518                                                  --    data in the first (left)
```

```
519                                                 --    table (`AP1.Vendors`)
520                                                 --    that is also in the
521                                                 --    second (right) table
522                                                 --    (`AP1.Terms`)
523   ON AP1.Vendors.DefaultTermsID = AP1.Terms.TermsID;
524                                                 -- 7. `ON` two fields with the
525                                                 --    same values/data, but in
526                                                 --    this case NOT the same
527                                                 --    name (`DefaultTermsID`
528                                                 --    and `TermsID`)
529
530
531 /* ***************************************************************************
532     2.15. The query above can also be written using an alias for each table.
533  *************************************************************************** */
534
535 SELECT v.VendorID,
536   v.VendorName,
537   REPLACE(
538     REPLACE(v.VendorAddress1,
539     'Box', 'PO Box'),
540   'PO PO Box', 'PO Box') AS VendorAddress1,
541   v.VendorAddress2,
542   v.VendorCity,
543   v.VendorState,
544   v.VendorZipCode,
545   REPLACE(CONCAT (
546       '(',
547       LEFT(v.VendorPhone, 3),
548       ') ',
549       SUBSTRING(v.VendorPhone, 4, 3),
550       '-',
551       RIGHT(v.VendorPhone, 4)
552       ), '() -', '') AS VendorPhone,
553   v.VendorContactLName,
554   v.VendorContactFName,
555   v.DefaultTermsID,
556   v.DefaultAccountNo,
557   t.TermsID,
558   t.TermsDescription,
559   t.TermsDueDays
560 FROM AP1.Vendors AS v                           -- 1. using alias `v` for table
561                                                 --    `AP1.Vendors`
562 INNER JOIN AP1.Terms AS t                       -- 2. using alias `t` for table
563                                                 --    `AP1.Terms`
564   ON v.DefaultTermsID = t.TermsID;
565
566
567 /* ***************************************************************************
568     2.16. In the example below, we use the functions that we have covered to
569           manipulate strings (any array of characters, such as letters and
570           numbers).
```

```
571   ********************************************************************************* */
572
573  SELECT VendorID,
574    LEFT(VendorName, 8) AS VendorNameL,           -- 1. retrieving eight (8)
575                                                  --    characters from the left
576                                                  --    of each string value in
577                                                  --    column `VendorName`;
578                                                  --    returns `US Posta`
579                                                  --    (row 1)
580    RIGHT(VendorName, 8) AS VendorNameR,          -- 2. retrieving eight (8)
581                                                  --    characters from the right
582                                                  --    of each string value in
583                                                  --    column `VendorName`;
584                                                  --    returns ` Service`
585                                                  --    including the leading
586                                                  --    space (row 1)
587    CONCAT (                                      -- 3. concatenating the string
588      VendorAddress1,                             --    value in column
589      ' ',                                        --    `VendorAddress1`, a space
590      VendorAddress2                              --    and the value in
591      ) AS VendorAddess,                          --    column `VendorAddress2`;
592                                                  --    returns `PO Box 96621 `
593                                                  --    including the space since
594                                                  --    there was no value in
595                                                  --    `VendorAddress2` (row 2)
596    UPPER(VendorCity) AS VendorCity,              -- 4. changing the the string
597                                                  --    value in column
598                                                  --    `VendorCity` to upper
599                                                  --    case; returns `MADISON`
600                                                  --    (row 1)
601    LOWER(VendorState) AS VendorState,            -- 5. changing the the string
602                                                  --    value in column
603                                                  --    `VendorState` to lower
604                                                  --    case; returns `dc`
605                                                  --    (row 2)
606    VendorZipCode,
607    SUBSTRING(VendorPhone, 4, 3) AS VendorPhone,  -- 6. retrieving three (3)
608                                                  --    characters starting from
609                                                  --    the character four (4)
610                                                  --    of each string value in
611                                                  --    column `VendorName`;
612                                                  --    returns `555` (row 1) and
613                                                  --    `255` (row 73)
614    REPLACE(VendorContactLName, 'en', 'XX')       -- 7. replacing pattern `en` in
615      AS VendorContactLName,                      --    each string value in
616                                                  --    column
617                                                  --    `VendorContactLName` with
618                                                  --    pattern `XX` when found;
619                                                  --    returns `MaegXX` (row 7)
620                                                  --    and `AileXX` (row 16)
621    VendorContactFName,
622    LEN(VendorContactFName)                       -- 8. retrieving the length as
```

```
623      AS VendorContactFNameLEN,                  --    an integer of each string
624                                                 --    value in column
625                                                 --    `VendorContactFName`;
626                                                 --    returns 9 (row 1)
627    DefaultTermsID,
628    DefaultAccountNo
629  FROM AP1.Vendors;
630
631
632  /* ***************************************************************************
633   https://folvera.commons.gc.cuny.edu/?p=1216
634   ************************************************************************** */
```