

```

1 /* *****
2 ..... INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
3 ..... WS24SQL10001, 2024/03/11 - 2024/04/10
4 ..... https://folvera.commons.gc.cuny.edu/?cat=34
5 *****
6
7 ..... SESSION #3 (2024/03/18): MANIPULATING DATA
8
9 ..... 1. Using built-in functions for strings
10 ..... 2. Querying two or more datasets (tables or views) using `INNER JOIN`,
11 ..... `LEFT [OUTER] JOIN` and `RIGHT [OUTER] JOIN`
12 *****
13
14 ..... 1. LAB 1
15 ..... Write a query calling all shared rows/records (`INNER JOIN`) from
16 ..... `AP1.Invoices`, `AP1.Terms` and `AP1.Vendors`.
17 ..... * Delete or rename the duplicate name of the columns.
18 ..... ***** */
19
20 SELECT AP1.Invoices.InvoiceID,
21 ..... AP1.Invoices.VendorID,
22 ..... AP1.Invoices.InvoiceNumber,
23 ..... AP1.Invoices.InvoiceDate,
24 ..... AP1.Invoices.InvoiceTotal,
25 ..... AP1.Invoices.PaymentTotal,
26 ..... AP1.Invoices.CreditTotal,
27 ..... AP1.Invoices.TermsID,
28 ..... AP1.Invoices.InvoiceDueDate,
29 ..... AP1.Invoices.PaymentDate,
30 ..... -- AP1.Terms.TermsID, ..... -- 1. duplicate column name
31 ..... ..... (`TermsID`), which can
32 ..... ..... be removed (commented
33 ..... ..... out, in this case)
34 ..... ..... without affecting the
35 ..... ..... query output; could also
36 ..... ..... be renamed
37 ..... AP1.Terms.TermsDescription,
38 ..... AP1.Terms.TermsDueDays,
39 ..... -- AP1.Vendors.VendorID, ..... -- 2. duplicate column name
40 ..... ..... (`VendorID`), which can
41 ..... ..... be removed (commented
42 ..... ..... out, in this case)
43 ..... ..... without affecting the
44 ..... ..... query output; could also
45 ..... ..... be renamed
46 ..... AP1.Vendors.VendorName,
47 ..... AP1.Vendors.VendorAddress1,
48 ..... AP1.Vendors.VendorAddress2,
49 ..... AP1.Vendors.VendorCity,
50 ..... AP1.Vendors.VendorState,
51 ..... AP1.Vendors.VendorZipCode,
52 ..... 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 + (plus) 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
208 ..... https://techonthenet.com/sql\_server/functions/count.php
209
210 ..... FLOOR() ..... returns the largest integer value that is equal to or less
211 ..... than a number
212 ..... https://techonthenet.com/sql\_server/functions/floor.php
213
214 ..... LEN() ..... returns the length of the specified string... does not
215 ..... include trailing space characters at the end the string
216 ..... when calculating the length
217 ..... https://techonthenet.com/sql\_server/functions/len.php
218
219 ..... MAX() ..... returns the maximum value of an expression
220 ..... https://techonthenet.com/sql\_server/functions/max.php
221
222 ..... MIN() ..... returns the minimum value of an expression
223 ..... https://techonthenet.com/sql\_server/functions/min.php
224
225 ..... RAND() ..... returns a random number or a random number within a range
226 ..... https://techonthenet.com/sql\_server/functions/rand.php
227
228 ..... ROUND() ..... returns a number rounded to a certain number of decimal
229 ..... places
230 ..... https://techonthenet.com/sql\_server/functions/round.php
231
232 ..... SUM() ..... returns the summed value of an expression
233 ..... https://techonthenet.com/sql\_server/functions/sum.php
234
235 --- Note that every time you have a function, you need parenthesis. Go to
236 --- https://techonthenet.com/sql\_server/functions/index\_alpha.php for a
237 --- complete list of built-in functions.
238
239 --- As you might have noticed, some built-in functions manipulate strings.
240 --- When working with numerical values, first we would have to convert them
241 --- into strings as we will see later in the course.
242
243 --- Some other built-in functions ``return a single value, calculated from
244 --- values in a column``. These are referred to as aggregate functions
245 --- (https://msdn.microsoft.com/en-us/library/ms173454.aspx).
246
247 2. Understanding the concepts above, we can now use them.
248
249 --- 2.01. In the example below, we concatenate (put strings together) columns
250 --- ``FirstName`` and ``LastName`` from table ``AP1.ContactUpdates``.
251 --- ***** */
252
253 SELECT CONCAT (
254 ..... FirstName,
255 ..... ' ',
256 ..... LastName
257 ..... ) AS NAME
258 FROM AP1.ContactUpdates;
259
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!'); -- returns `WE ARE LEARNING
268                                     -- SQL!`
269
270
271 /* *****
272 2.03. In the example below, we concatenate (put strings together) columns
273 `FirstName` and `LastName` from table `AP1.ContactUpdates`, just like
274 the previous example.
275
276 We also use `LTRIM()` and `RTRIM()` to remove leading and trailing
277 spaces from `FirstName` with `LTRIM(RTRIM(FirstName))` and `LastName`
278 with `LTRIM(RTRIM(LastName))`.
279 ***** */
280
281 SELECT CONCAT (
282     LTRIM(RTRIM(LastName)),
283     ', ',
284     LTRIM(RTRIM(FirstName))
285 ) AS NAME
286 FROM AP1.ContactUpdates;
287
288
289 /* *****
290 2.04. In the examples below, we use `UPPER()` to change a string to upper
291 case and print the result to console.
292 ***** */
293
294 PRINT UPPER('this string is in upper case'); -- returns `THIS STRING SHOULD
295                                     -- IN UPPER CASE`
296
297
298 /* *****
299 2.05. In the examples below, we use `LOWER()` to change a string to lower
300 case.
301 ***** */
302
303 PRINT LOWER('BUT THIS STRING IS IN LOWER CASE. ');
304                                     -- returns `but this string is
305                                     -- in lower case.`
306
307
308 /* *****
309 2.06. In the examples below, we use `RIGHT()` to extract characters from
310 the right.
311 ***** */
312
313 PRINT RIGHT('apple', 2); -- returns `le`
314
315
316 /* *****
317 2.07. In the examples below, we use `LEFT()` to extract characters from the
318 left.
319 ***** */
320
321 PRINT LEFT('apple', 2); -- returns `ap`
322
323
324 /* *****
325 2.08. In the examples below, we use `SUBSTRING()` to extract characters

```

```

326 ..... from the middle -- same as the built-in function `MID()` in other
327 ..... relational database management systems (RDBMS) like Oracle -- and
328 ..... print the result to the console
329 ..... ***** */
330
331 PRINT SUBSTRING('apple tree #5', 6, 10); ..... -- returns `tree #5`
332
333
334 /* *****
335 ..... 2.09. In the example below, we use `LEN()` to retrieve the length of a
336 ..... string.
337 ..... ***** */
338
339 PRINT LEN('tree #5'); ..... -- returns 12 (numeric value)
340
341
342 /* *****
343 ..... 2.10. In the examples below, we use `LTRIM()` and `RTRIM()` to remove any
344 ..... leading and/or trailing spaces from the strings in single quotes and
345 ..... print the result to the console.
346
347 ..... We could also use function `TRIM()` only in SQL Server
348 ..... (https://docs.microsoft.com/en-us/sql/t-sql/functions/trim-transact-sql).
349 ..... ***** */
350
351 PRINT LTRIM('apple tree'), ..... -- 1. trimming leading spaces
352 RTRIM('tree #5'), ..... -- 2. trimming trailing spaces
353 LTRIM(RTRIM('apple tree #5')); ..... -- 3. trimming leading and
354 ..... trailing spaces
355
356
357 /* *****
358 ..... 2.11. In the example below, we use `REPLACE()` to replace pattern `mstake`
359 ..... with `mistake`. Since `mstake` exists in string `This is a mstake`,
360 ..... `REPLACE()` returns `This is a mistake`.
361 ..... ***** */
362
363 PRINT REPLACE('This is a mstake', 'mstake', 'mistake');
364 ..... -- returns `This is a mistake`
365
366
367 /* *****
368 ..... In the example below, we use `REPLACE()` to replace pattern `gg` with
369 ..... `mistake`. Since `gg` does not exist in `This is a mstake`,
370 ..... `REPLACE()` returns the original value.
371 ..... ***** */
372
373 PRINT REPLACE('This is a mstake', 'gg', 'mistake');
374 ..... -- returns `This is a mstake`
375
376
377 /* *****
378 ..... 2.12. In the example below, since there is no function to make the first
379 ..... letter of a string upper case and the rest lower case, we can use
380 ..... a combination of functions `UPPER()`, `LOWER()`, `RIGHT()`, `LEFT()`
381 ..... and `CONCAT()` working from the inside out and print the result to
382 ..... the console.
383 ..... ***** */
384
385 PRINT CONCAT (
386 ..... UPPER(LEFT('HELLO', 1)) ..... -- 1. retrieving first
387 ..... character from `HELLO`;
388 ..... returns `H`
389 ..... ) ..... -- 2. making `H` upper case;
390 ..... returns `H`

```

```

391     ,
392     LOWER(SUBSTRING('HELLO', 2, LEN('HELLO')) -- 3. retrieving variable
393     ) -- 3. number of characters
394     -- 3. from character two (2)
395     -- 3. to the length of the
396     -- 3. string (integer value
397     -- 3. of 5); returns `ELLO`
398     ) -- 4. making `ELLO` lower
399     -- 4. case; returns `ello`
400     ); -- 5. concatenating all
401     -- 5. previous sections;
402     -- 5. returns `Hello`
403
404
405 /* *****
406 2.13. In the example below, we use `REPLACE()` to change pattern ` ` (two
407 spaces, `CHAR(32)+CHAR(32)`) with ` ` (a single space, `CHAR(32)`).
408
409 PRINT REPLACE('tree   #5', ' ', ' ');
410
411 Since string `tree   #5` has more than two spaces, we need run
412 several passes of `REPLACE()`.
413
414 The statement runs from the inside out (3, 2, 1, 2, 3).
415
416     function 3 -- 3. beginning of function #3:
417     -- 3. * receiving value of
418     -- 3. function #2
419     function 2 -- 2. beginning of function #2:
420     -- 2. * receiving value of
421     -- 2. function #1
422     function 1 -- 1. function #1:
423     -- 1. * receiving original
424     -- 1. value #0
425     -- 1. * returning new value #1
426     function 2 -- 2. end function of #2:
427     -- 2. * returning new value #2
428     function 3 -- 3. end function of #3:
429     -- 3. * returning new value #3
430     -- 3. (final value)
431     ***** */
432
433 PRINT
434     REPLACE( -- 3. pass #3 to replace
435     -- 3. `CHAR(32)+CHAR(32)` for
436     -- 3. `CHAR(32)`
437     -- 3. * returns `tree #5` with
438     -- 3. 1 space
439     REPLACE( -- 2. pass #2 to replace
440     -- 2. `CHAR(32)+CHAR(32)` for
441     -- 2. `CHAR(32)`
442     -- 2. * returns `tree #5` with
443     -- 2. 2 space, which feeds
444     -- 2. pass #3
445     REPLACE('tree   #5', -- 1. pass #1 to replace
446     -- 1. `CHAR(32)+CHAR(32)` for
447     -- 1. `CHAR(32)`
448     -- 1. * returns `tree #5`
449     -- 1. with 3 spaces, which
450     -- 1. feeds pass #2
451     -- 1. ), -- 2. end of pass #2
452     -- 1. ), -- 3. end of pass #3
453     -- 1. );
454
455 /* *****

```

```

456 2.14. In the example below, we use `REPLACE()` to replace pattern `tree`
457 for `fruit`.
458
459 Since pattern `tree` exists in `tree` with leading and
460 trailing spaces around `tree`, `REPLACE()` returns `fruit`
461 with leading and trailing spaces around word `fruit`.
462
463 We also use `RTRIM()` and `LTRIM()` to remove trailing and leading
464 spaces respectively to get `fruit` without leading and trailing
465 spaces.
466 ***** */
467
468 PRINT RTRIM(LTRIM(REPLACE('tree', 'tree', 'fruit')));
469 -- returns `fruit`;
470 -- empty spaces untouched
471
472
473 /* *****
474 2.15. In the example below, we use `REPLACE()` to replace pattern `Box` for
475 `PO Box`. The first pass (inner) of `REPLACE()` changes some fields
476 to `PO PO Box`. The second pass (outer) of `REPLACE()` changes the
477 previous logical error (`PO PO Box`) to `PO Box`.
478 ***** */
479
480 SELECT AP1.Vendors.VendorID, -- 1. fields using format
481 AP1.Vendors.VendorName, -- `schema.table.field`
482 REPLACE( -- 2. second pass of
483 -- `REPLACE()` working from
484 -- inside out
485 REPLACE(AP1.Vendors.VendorAddress1, -- 3. first pass of `REPLACE()`
486 'Box', 'PO Box'), -- working from inside out
487 'PO PO Box', 'PO Box') AS VendorAddress1,
488 AP1.Vendors.VendorAddress2,
489 AP1.Vendors.VendorCity,
490 AP1.Vendors.VendorState,
491 AP1.Vendors.VendorZipCode,
492 REPLACE( -- 4. replacing `() -` from the
493 -- concatenation in #5
494 -- instead of a CASE clause
495 -- (logic block), which we
496 -- will cover later
497 CONCAT( -- 5. concatenating an opening
498 '(', -- parenthesis, first three
499 LEFT(Vendors.VendorPhone, 3), -- characters of
500 ')', -- `VendorPhone`, a closing
501 SUBSTRING(Vendors.VendorPhone, 4, 3), -- parenthesis with a space,
502 '-', -- the substring of
503 RIGHT(Vendors.VendorPhone, 4) -- `VendorPhone` starting
504 ), '() -', '') AS VendorPhone -- from the fourth character
505 -- and taking 3, a hyphen
506 -- and the last four
507 -- characters of
508 -- `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`) as in many
529 ..... real world databases
530
531
532 /* *****
533 2.16. The query above can also be written using an alias for each table.
534 ***** */
535
536 SELECT v.VendorID,
537        v.VendorName,
538        REPLACE(
539            REPLACE(v.VendorAddress1,
540                'Box', 'PO Box'),
541            'PO PO Box', 'PO Box') AS VendorAddress1,
542        v.VendorAddress2,
543        v.VendorCity,
544        v.VendorState,
545        v.VendorZipCode,
546        REPLACE(CONCAT (
547            '(',
548            LEFT(v.VendorPhone, 3),
549            ')',
550            SUBSTRING(v.VendorPhone, 4, 3),
551            '-',
552            RIGHT(v.VendorPhone, 4)
553            ), '() -', '') AS VendorPhone,
554        v.VendorContactLName,
555        v.VendorContactFName,
556        v.DefaultTermsID,
557        v.DefaultAccountNo,
558        t.TermsID,
559        t.TermsDescription,
560        t.TermsDueDays
561 FROM AP1.Vendors AS v ..... -- 1. using alias `v` for table
562 ..... `AP1.Vendors`
563 INNER JOIN AP1.Terms AS t ..... -- 2. using alias `t` for table
564 ..... `AP1.Terms`
565 ON v.DefaultTermsID = t.TermsID;
566
567
568 /* *****
569 2.17. In the example below, we use the functions that we have covered to
570 ..... manipulate strings (any array of characters, such as letters and
571 ..... numbers).
572 ***** */
573
574 SELECT VendorID,
575        LEFT(VendorName, 8) AS VendorNameL, ..... -- 1. retrieving eight (8)
576 ..... characters from the left
577 ..... of each string value in
578 ..... column `VendorName`;
579 ..... returns `US Posta`
580 ..... (row 1)
581        RIGHT(VendorName, 8) AS VendorNameR, ..... -- 2. retrieving eight (8)
582 ..... characters from the right
583 ..... of each string value in
584 ..... column `VendorName`;
585 ..... returns `Service`

```

```

586 ..... -- including the leading
587 ..... space (row 1)
588 CONCAT ( ..... -- 3. concatenating the string
589 VendorAddress1, ..... value in column
590 ' ', ..... `VendorAddress1`, a space
591 VendorAddress2 ..... and the value in
592 ) AS VendorAddress, ..... column `VendorAddress2`;
593 ..... returns `PO Box 96621`
594 ..... including the space since
595 ..... there was no value in
596 ..... `VendorAddress2` (row 2)
597 UPPER(VendorCity) AS VendorCity, ..... -- 4. changing the the string
598 ..... value in column
599 ..... `VendorCity` to upper
600 ..... case; returns `MADISON`
601 ..... (row 1)
602 LOWER(VendorState) AS VendorState, ..... -- 5. changing the the string
603 ..... value in column
604 ..... `VendorState` to lower
605 ..... case; returns `dc`
606 ..... (row 2)
607 VendorZipCode,
608 SUBSTRING(VendorPhone, 4, 3) AS VendorPhone, ..... -- 6. retrieving three (3)
609 ..... characters starting from
610 ..... the character four (4)
611 ..... of each string value in
612 ..... column `VendorName`;
613 ..... returns `555` (row 1) and
614 ..... `255` (row 73)
615 REPLACE(VendorContactLName, 'en', 'XX') ..... -- 7. replacing pattern `en` in
616 AS VendorContactLName, ..... each string value in
617 ..... column
618 ..... `VendorContactLName` with
619 ..... pattern `XX` when found;
620 ..... returns `MaegXX` (row 7)
621 ..... and `AileXX` (row 16)
622 VendorContactFName,
623 LEN(VendorContactFName) ..... -- 8. retrieving the length as
624 AS VendorContactFNameLEN, ..... an integer of each string
625 ..... value in column
626 ..... `VendorContactFName`;
627 ..... returns 9 (row 1)
628 DefaultTermsID,
629 DefaultAccountNo
630 FROM AP1.Vendors;
631
632
633 /* *****
634 ..... 2.18. In the example below, we take the original value of
635 ..... `AP1.Vendors.VendorPhone` (`8005551205`) and divide it into three (3)
636 ..... sections -- the area code (`800`), the branch exchange (`555`) and
637 ..... the subscriber number (`1205`). Then we concatenate the latter
638 ..... values with hard-coded values (opening and closing parenthesis and a
639 ..... hyphen) to present the phone number in a more legible way
640 ..... (`(800) 555-1205`).
641 ..... ***** */
642
643 SELECT VendorPhone AS original_number, ..... -- 1. dividing original value
644 ..... of VendorPhone
645 ..... (`8005551205`) to get
646 ..... `(800) 555-1205`
647 LEFT(VendorPhone, 3) AS area_code, ..... -- 2. retrieving three (3)
648 ..... characters from the left
649 ..... (`800`, area code)
650 SUBSTRING(VendorPhone, 4, 3) AS branch_exch, ..... -- 3. retrieving the middle

```

```

651 ..... -- three (3) characters
652 ..... -- starting for the fourth
653 ..... -- (4th) character (`555`,
654 ..... -- branch exchange)
655 RIGHT(VendorPhone, 4) AS subscriber_number, -- 4. retrieving four (4)
656 ..... -- characters from the right
657 ..... -- (`1205`, subscriber
658 ..... -- number)
659 CONCAT ( -- 5. concatenating the latter
660 ..... -- values plus hard-coded
661 ..... -- values (opening and
662 ..... -- closing parenthesis and a
663 ..... -- hyphen)
664 ..... '(', -- 5.1. value 1, the opening
665 ..... -- parenthesis
666 LEFT(VendorPhone, 3), -- 5.2. value 2, area code
667 ..... ')', -- 5.3. value 3, the closing
668 ..... -- parenthesis and a
669 ..... -- space
670 SUBSTRING(VendorPhone, 4, 3), -- 5.4. value 4, branch
671 ..... -- exchange
672 ..... '-', -- 5.5. value 5, hyphen
673 RIGHT(VendorPhone, 4) -- 5.6. value 6, subscriber
674 ..... -- number
675 ..... ) AS legible_phone_number
676 FROM AP1.Vendors;
677
678
679 /* *****
680 https://folvera.commons.gc.cuny.edu/?p=1264
681 ***** */

```