

```

1  /* *****
2      INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
3          SF25SQL6172025, 2025/11/17 - 2025/12/17
4          https://folvera.commons.gc.cuny.edu/?cat=35
5  *****
6
7  SESSION #3 (2025/11/24): 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,
31
32
33
34
35
36
37         AP1.Terms.TermsDescription,
38         AP1.Terms.TermsDueDays,
39         -- AP1.Vendors.VendorID,
40
41
42
43
44
45
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
58 INNER JOIN AP1.Terms
59
60
61
62
63
64         ON AP1.Invoices.TermsID = AP1.Terms.TermsID
65
66
67
68
69

```

```

-- 1. duplicate column name
--   (`TermsID`), which can
--   be removed (commented
--   out, in this case)
--   without affecting the
--   query output; could also
--   be renamed

-- 2. duplicate column name
--   (`VendorID`), which can
--   be removed (commented
--   out, in this case)
--   without affecting the
--   query output; could also
--   be renamed

-- 3. from table `AP1.Invoices`
-- 4. `INNER JOIN` to retrieve
--   data in the first (left)
--   table (`AP1.Invoices`)
--   that is also in the
--   second (right) table
--   (`AP1.Terms`)
-- 5. `ON` two fields with the
--   same values/data and the
--   same name (`TermsID`);
--   specifying the relation
--   between tables
--   `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 /* *****
101    * As an alternative, the code above can also be written using an alias
102      (`AS`) for each table in order to simplify the code.
103
104          `i` for `AP1.Invoices`
105          `t` for `AP1.Terms`
106          `v` for `AP1.Vendors`
107
108      Note that, if you use an alias (`AS`) for a table (for example, `v` for
109      `AP1.Vendors`), you must use the alias (`v`) when calling the table
110      (`AP1.Vendors`) in the query (for example, calling `AP1.Vendors.VendorID`
111      as `v.VendorID`).
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	CONCAT()	allows you to concatenate strings together
157		<a href="https://techonthenet.com/sql_server/functions/concat.php">https://techonthenet.com/sql_server/functions/concat.php</a>
158		
159	+	(plus) also allows you to concatenate strings together although
160		adding NULL returns a NULL
161		<a href="https://techonthenet.com/sql_server/functions/concat2.php">https://techonthenet.com/sql_server/functions/concat2.php</a>
162		
163	LEFT()	allows you to extract a substring from a string, starting
164		from the left-most character
165		<a href="https://techonthenet.com/sql_server/functions/left.php">https://techonthenet.com/sql_server/functions/left.php</a>
166		
167	LEN()	returns the length of the specified string... does not
168		include trailing space characters at the end the string
169		when calculating the length
170		<a href="https://techonthenet.com/sql_server/functions/len.php">https://techonthenet.com/sql_server/functions/len.php</a>
171		
172	LTRIM()	removes all space characters from the left-hand side of a
173		string
174		<a href="https://techonthenet.com/sql_server/functions/ltrim.php">https://techonthenet.com/sql_server/functions/ltrim.php</a>
175		
176	LOWER()	converts all letters in the specified string to lowercase
177		<a href="https://techonthenet.com/sql_server/functions/lower.php">https://techonthenet.com/sql_server/functions/lower.php</a>
178		
179	REPLACE()	replaces a sequence of characters in a string with another
180		set of characters, not case-sensitive
181		<a href="https://techonthenet.com/sql_server/functions/replace.php">https://techonthenet.com/sql_server/functions/replace.php</a>
182		
183	RIGHT()	allows you to extract a substring from a string, starting
184		from the right-most character
185		<a href="https://techonthenet.com/sql_server/functions/right.php">https://techonthenet.com/sql_server/functions/right.php</a>
186		
187	RTRIM()	removes all space characters from the right-hand side of a
188		string
189		<a href="https://techonthenet.com/sql_server/functions/rtrim.php">https://techonthenet.com/sql_server/functions/rtrim.php</a>
190		
191	SUBSTRING()	allows you to extract a substring from a string
192		<a href="https://techonthenet.com/sql_server/functions/substring.php">https://techonthenet.com/sql_server/functions/substring.php</a>
193		
194	UPPER()	converts all letters in the specified string to uppercase
195		<a href="https://techonthenet.com/sql_server/functions/upper.php">https://techonthenet.com/sql_server/functions/upper.php</a>
196		
197		

198 We also have functions that affect numeric values.

199	AVG()	returns the average value of an expression
200		<a href="https://techonthenet.com/sql_server/functions/avg.php">https://techonthenet.com/sql_server/functions/avg.php</a>
201		
202	CEILING()	returns the smallest integer value that is greater than or
203		equal to a number
204		<a href="https://techonthenet.com/sql_server/functions/ceiling.php">https://techonthenet.com/sql_server/functions/ceiling.php</a>
205		
206	COUNT()	returns the count of an expression
207		

208 [https://techonthenet.com/sql\\_server/functions/count.php](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](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](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](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](https://techonthenet.com/sql_server/functions/min.php)  
224  
225 PRODUCT() returns the product of an expression  
226  
<https://www.sqlservercentral.com/articles/t-sql-in-sql-server-2025-the-operator>  
227  
<https://learn.microsoft.com/en-us/sql/t-sql/functions/product-aggregate-transact-sql?view=sql-server-ver17>  
\* brand new T-SQL 2025  
228  
229  
230 RAND() returns a random number or a random number within a range  
231 [https://techonthenet.com/sql\\_server/functions/rand.php](https://techonthenet.com/sql_server/functions/rand.php)  
232  
233 ROUND() returns a number rounded to a certain number of decimal  
234 places  
235 [https://techonthenet.com/sql\\_server/functions/round.php](https://techonthenet.com/sql_server/functions/round.php)  
236  
237 SUM() returns the summed value of an expression  
238 [https://techonthenet.com/sql\\_server/functions/sum.php](https://techonthenet.com/sql_server/functions/sum.php)  
239

240 Note that every time you have a function, you need parenthesis. Go to  
241 [https://techonthenet.com/sql\\_server/functions/index\\_alpha.php](https://techonthenet.com/sql_server/functions/index_alpha.php) for a  
242 complete list of built-in functions.  
243

244 As you might have noticed, some built-in functions manipulate strings.  
245 When working with numerical values, first we would have to convert them  
246 into strings as we will see later in the course.  
247

248 Some other built-in functions ``return a single value, calculated from  
249 values in a column``. These are referred to as aggregate functions  
250 (<https://msdn.microsoft.com/en-us/library/ms173454.aspx>).  
251

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

253  
254 2.01. In the example below, we concatenate (put strings together) columns  
255 `FirstName` and `LastName` from table `AP1.ContactUpdates`.

256 `***** */`

```
257
258 SELECT CONCAT (
259     FirstName,
260     ' ',
261     LastName
262 ) AS NAME
263 FROM AP1.ContactUpdates;
```

264  
265  
266 `/* *****`

267 2.02. In the example below, we concatenate (put strings together) columns  
268 `WE `, `ARE `, `LEARNING `, `SQL!` and print the result to the  
269 console.  
270 `***** */`

```
271 PRINT CONCAT('WE ', 'ARE ', 'LEARNING ', 'SQL!'); -- returns `WE ARE LEARNING`
272
```

```

273                                     --      SQL!`
274
275
276 /* *****
277 2.03. In the example below, we concatenate (put strings together) columns
278 `FirstName` and `LastName` from table `AP1.ContactUpdates`, just like
279 the previous example.
280
281 We also use `LTRIM()` and `RTRIM()` to remove leading and trailing
282 spaces from `FirstName` with `LTRIM(RTRIM(FirstName))` and `LastName`
283 with `LTRIM(RTRIM(LastName))`.
284 ***** */
285
286 SELECT CONCAT (
287     LTRIM(RTRIM(LastName)),
288     ' ',
289     LTRIM(RTRIM(FirstName))
290 ) AS NAME
291 FROM AP1.ContactUpdates;
292
293
294 /* *****
295 2.04. In the examples below, we use `UPPER()` to change a string to upper
296 case and print the result to console.
297 ***** */
298
299 PRINT UPPER('this string is in upper case');    -- returns `THIS STRING SHOULD
300                                               --      IN UPPER CASE`
301
302
303 /* *****
304 2.05. In the examples below, we use `LOWER()` to change a string to lower
305 case.
306 ***** */
307
308 PRINT LOWER('BUT THIS STRING IS IN LOWER CASE. ');
309                                               -- returns `but this string is
310                                               --      in lower case.`
311
312
313 /* *****
314 2.06. In the examples below, we use `RIGHT()` to extract characters from
315 the right.
316 ***** */
317
318 PRINT RIGHT('apple', 2);                    -- returns `le`
319
320
321 /* *****
322 2.07. In the examples below, we use `LEFT()` to extract characters from the
323 left.
324 ***** */
325
326 PRINT LEFT('apple', 2);                      -- returns `ap`
327
328
329 /* *****
330 2.08. In the examples below, we use `SUBSTRING()` to extract characters
331 from the middle -- same as the built-in function `MID()` in other
332 relational database management systems (RDBMS) like Oracle -- and
333 print the result to the console
334 ***** */
335
336 PRINT SUBSTRING('apple tree #5', 6, 10);    -- returns ` tree #5`
337
338
339 /* *****
340 2.09. In the example below, we use `LEN()` to retrieve the length of a
341 string.

```

```

342      ***** */
343
344 PRINT LEN('tree      #5');          -- returns 12 (numeric value)
345
346
347 /* *****
348     2.10. In the examples below, we use `LTRIM()` and `RTRIM()` to remove any
349     leading and/or trailing spaces from the strings in single quotes and
350     print the result to the console.
351
352     We could also use function `TRIM()` only in SQL Server
353     (https://docs.microsoft.com/en-us/sql/t-sql/functions/trim-transact-sql).
354     ***** */
355
356 PRINT LTRIM('      tree'),          -- 1. trimming leading spaces
357 RTRIM('tree      '),              -- 2. trimming trailing spaces
358 LTRIM(RTRIM('      tree      '));  -- 3. trimming leading and
359                                     -- trailing spaces
360
361
362 /* *****
363     2.11. In the example below, we use `REPLACE()` to replace pattern `mstake`
364     with `mistake`. Since `mstake` exists in string `This is a mstake`,
365     `REPLACE()` returns `This is a mistake`.
366     ***** */
367
368 PRINT REPLACE('This is a mstake', 'mstake', 'mistake');
369                                     -- returns `This is a mistake`
370
371
372 /* *****
373     In the example below, we use `REPLACE()` to replace pattern `gg` with
374     `mistake`. Since `gg` does not exist in `This is a mstake`,
375     `REPLACE()` returns the original value.
376     ***** */
377
378 PRINT REPLACE('This is a mstake', 'gg', 'mistake');
379                                     -- returns `This is a mstake`
380
381
382 /* *****
383     2.12. In the example below, since there is no function to make the first
384     letter of a string upper case and the rest lower case, we can use
385     a combination of functions `UPPER()`, `LOWER()`, `RIGHT()`, `LEFT()`
386     and `CONCAT()` working from the inside out and print the result to
387     the console.
388     ***** */
389
390 PRINT CONCAT (
391     UPPER(LEFT('hELLO', 1))          -- 1. retrieving first
392                                     -- character from `hELLO`;
393                                     -- returns `h`
394 )                                     -- 2. making `h` upper case;
395                                     -- returns `H`
396 ,
397 LOWER(SUBSTRING('hELLO', 2, LEN('hELLO')) -- 3. retrieving variable
398 )                                     -- number of characters
399                                     -- from character two (2)
400                                     -- to the length of the
401                                     -- string (integer value
402                                     -- of 5); returns `ELLO`
403 )                                     -- 4. making `ELLO` lower
404                                     -- case; returns `ello`
405 );                                     -- 5. concatenating all
406                                     -- previous sections;
407                                     -- returns `Hello`
408
409
410 /* *****

```

411 2.13. In the example below, we use `REPLACE()` to change pattern ` ` (two  
412 spaces, `CHAR(32)+CHAR(32)`) with ` ` (a single space, `CHAR(32)`).

```
413  
414 PRINT REPLACE('tree #5', ' ', ' ');  
415
```

416 Since string `tree #5` has more than two spaces, we need run  
417 several passes of `REPLACE()`.

418  
419 The statement runs from the inside out (3, 2, 1, 2, 3).

```
420  
421 function 3 -- 3. beginning of function #3:  
422 -- * receiving value of  
423 -- function #2  
424 function 2 -- 2. beginning of function #2:  
425 -- * receiving value of  
426 -- function #1  
427 function 1 -- 1. function #1:  
428 -- * receiving original  
429 -- value #0  
430 -- * returning new value #1  
431 function 2 -- 2. end function of #2:  
432 -- * returning new value #2  
433 function 3 -- 3. end function of #3:  
434 -- * returning new value #3  
435 -- (final value)  
436 ***** */
```

```
437  
438 PRINT  
439 REPLACE( -- 3. pass #3 to replace  
440 -- `CHAR(32)+CHAR(32)` for  
441 -- `CHAR(32)`  
442 -- * returns `tree #5` with  
443 -- 1 space  
444 REPLACE( -- 2. pass #2 to replace  
445 -- `CHAR(32)+CHAR(32)` for  
446 -- `CHAR(32)`  
447 -- * returns `tree #5` with  
448 -- 2 space, which feeds  
449 -- pass #3  
450 REPLACE('tree #5', -- 1. pass #1 to replace  
451 -- `CHAR(32)+CHAR(32)` for  
452 -- `CHAR(32)`  
453 -- * returns `tree #5`  
454 -- with 3 spaces, which  
455 -- feeds pass #2  
456 -- 2. end of pass #2  
457 -- 3. end of pass #3  
458  
459
```

460 /\* \*\*\*\*\* \*/  
461 2.14. In the example below, we use `REPLACE()` to replace pattern `tree`  
462 for `fruit`.

463  
464 Since pattern `tree` exists in ` tree ` with leading and  
465 trailing spaces around `tree`, `REPLACE()` returns ` fruit ` with  
466 leading and trailing spaces around word `fruit`.

467  
468 We also use `RTRIM()` and `LTRIM()` to remove trailing and leading  
469 spaces respectively to get `fruit` without leading and trailing  
470 spaces.

```
471 ***** */  
472  
473 PRINT RTRIM(LTRIM(REPLACE(' tree ', 'tree', 'fruit')));  
474 -- returns ` fruit `;  
475 -- empty spaces untouched  
476  
477
```

478 /\* \*\*\*\*\* \*/  
479 2.15. In the example below, we use `REPLACE()` to replace pattern `Box` for

```

480         `PO Box`. The first pass (inner) of `REPLACE()` changes some fields
481         to `PO PO Box`. The second pass (outer) of `REPLACE()` changes the
482         previous logical error (`PO PO Box`) to `PO Box`.
483         ***** */
484
485 SELECT AP1.Vendors.VendorID, -- 1. fields using format
486         AP1.Vendors.VendorName, -- `schema.table.field`
487         REPLACE ( -- 2. second pass of
488 -- `REPLACE()` working from
489 -- inside out
490         REPLACE (AP1.Vendors.VendorAddress1, -- 3. first pass of `REPLACE()`
491         'Box', 'PO Box'), -- working from inside out
492         'PO PO Box', 'PO Box') AS VendorAddress1,
493         AP1.Vendors.VendorAddress2,
494         AP1.Vendors.VendorCity,
495         AP1.Vendors.VendorState,
496         AP1.Vendors.VendorZipCode,
497         REPLACE ( -- 4. replacing `() -` from the
498 -- concatenation in #5
499 -- instead of a CASE clause
500 -- (logic block), which we
501 -- will cover later
502         CONCAT ( -- 5. concatenating an opening
503         '(', -- parenthesis, first three
504         LEFT (Vendors.VendorPhone, 3), -- characters of
505         ') ', -- `VendorPhone`, a closing
506         SUBSTRING (Vendors.VendorPhone, 4, 3), -- parenthesis with a space,
507         '-', -- the substring of
508         RIGHT (Vendors.VendorPhone, 4) -- `VendorPhone` starting
509         ), '() -', '') AS VendorPhone -- from the fourth character
510 -- and taking 3, a hyphen
511 -- and the last four
512 -- characters of
513 -- `VendorPhone`
514         AP1.Vendors.VendorContactLName,
515         AP1.Vendors.VendorContactFName,
516         AP1.Vendors.DefaultTermsID,
517         AP1.Vendors.DefaultAccountNo,
518         AP1.Terms.TermsID,
519         AP1.Terms.TermsDescription,
520         AP1.Terms.TermsDueDays
521 FROM AP1.Vendors
522 INNER JOIN AP1.Terms -- 6. `INNER JOIN` to retrieve
523 -- data in the first (left)
524 -- table (`AP1.Vendors`)
525 -- that is also in the
526 -- second (right) table
527 -- (`AP1.Terms`)
528         ON AP1.Vendors.DefaultTermsID = AP1.Terms.TermsID; -- 7. `ON` two fields with the
529 -- same values/data, but in
530 -- this case NOT the same
531 -- name (`DefaultTermsID`
532 -- and `TermsID`) as in many
533 -- real world databases
534
535
536
537 /* *****
538     2.16. The query above can also be written using an alias for each table.
539     ***** */

```

```

540
541 SELECT v.VendorID,
542         v.VendorName,
543         REPLACE (
544         REPLACE (v.VendorAddress1,
545         'Box', 'PO Box'),
546         'PO PO Box', 'PO Box') AS VendorAddress1,
547         v.VendorAddress2,
548         v.VendorCity,

```

```

549     v.VendorState,
550     v.VendorZipCode,
551     REPLACE(CONCAT (
552         '(',
553         LEFT(v.VendorPhone, 3),
554         ')',
555         SUBSTRING(v.VendorPhone, 4, 3),
556         '-',
557         RIGHT(v.VendorPhone, 4)
558     ), '()' -', '') AS VendorPhone,
559     v.VendorContactLName,
560     v.VendorContactFName,
561     v.DefaultTermsID,
562     v.DefaultAccountNo,
563     t.TermsID,
564     t.TermsDescription,
565     t.TermsDueDays
566 FROM AP1.Vendors AS v -- 1. using alias `v` for table
567 -- `AP1.Vendors`
568 INNER JOIN AP1.Terms AS t -- 2. using alias `t` for table
569 -- `AP1.Terms`
570     ON v.DefaultTermsID = t.TermsID;
571
572
573 /* *****
574     2.17. In the example below, we use the functions that we have covered to
575     manipulate strings (any array of characters, such as letters and
576     numbers).
577     ***** */
578
579 SELECT VendorID,
580     LEFT(VendorName, 8) AS VendorNameL, -- 1. retrieving eight (8)
581 -- characters from the left
582 -- of each string value in
583 -- column `VendorName`;
584 -- returns `US Posta`
585 -- (row 1)
586     RIGHT(VendorName, 8) AS VendorNameR, -- 2. retrieving eight (8)
587 -- characters from the right
588 -- of each string value in
589 -- column `VendorName`;
590 -- returns `Service`
591 -- including the leading
592 -- space (row 1)
593     CONCAT ( -- 3. concatenating the string
594         VendorAddress1, -- value in column
595         ' ', -- `VendorAddress1`, a space
596         VendorAddress2 -- and the value in
597     ) AS VendorAddress, -- column `VendorAddress2`;
598 -- returns `PO Box 96621`
599 -- including the space since
600 -- there was no value in
601 -- `VendorAddress2` (row 2)
602     UPPER(VendorCity) AS VendorCity, -- 4. changing the the string
603 -- value in column
604 -- `VendorCity` to upper
605 -- case; returns `MADISON`
606 -- (row 1)
607     LOWER(VendorState) AS VendorState, -- 5. changing the the string
608 -- value in column
609 -- `VendorState` to lower
610 -- case; returns `dc`
611 -- (row 2)
612     VendorZipCode,
613     SUBSTRING(VendorPhone, 4, 3) AS VendorPhone, -- 6. retrieving three (3)
614 -- characters starting from
615 -- the character four (4)
616 -- of each string value in
617 -- column `VendorName`;

```

```

618                                     -- returns `555` (row 1) and
619                                     -- `255` (row 73)
620 REPLACE (VendorContactLName, 'en', 'XX') -- 7. replacing pattern `en` in
621     AS VendorContactLName,             -- each string value in
622                                     -- column
623                                     -- `VendorContactLName` with
624                                     -- pattern `XX` when found;
625                                     -- returns `MaegXX` (row 7)
626                                     -- and `AileXX` (row 16)
627 VendorContactFName,
628 LEN (VendorContactFName)             -- 8. retrieving the length as
629     AS VendorContactFNameLEN,         -- an integer of each string
630                                     -- value in column
631                                     -- `VendorContactFName`;
632                                     -- returns 9 (row 1)
633 DefaultTermsID,
634 DefaultAccountNo
635 FROM AP1.Vendors;

```

```

636
637
638 /* *****
639 2.18. In the example below, we take the original value of
640 `AP1.Vendors.VendorPhone` (`8005551205`) and divide it into three (3)
641 sections -- the area code (`800`), the branch exchange (`555`) and
642 the subscriber number (`1205`). Then we concatenate the latter
643 values with hard-coded values (opening and closing parenthesis and a
644 hyphen) to present the phone number in a more legible way
645 (`(800) 555-1205`).
646 ***** */

```

```

647
648 SELECT VendorPhone AS original_number, -- 1. dividing original value
649                                     -- of VendorPhone
650                                     -- (`8005551205`) to get
651                                     -- `(800) 555-1205`
652 LEFT (VendorPhone, 3) AS area_code, -- 2. retrieving three (3)
653                                     -- characters from the left
654                                     -- (`800`, area code)
655 SUBSTRING (VendorPhone, 4, 3) AS branch_exch, -- 3. retrieving the middle
656                                     -- three (3) characters
657                                     -- starting for the fourth
658                                     -- (4th) character (`555`,
659                                     -- branch exchange)
660 RIGHT (VendorPhone, 4) AS subscriber_number, -- 4. retrieving four (4)
661                                     -- characters from the right
662                                     -- (`1205`, subscriber
663                                     -- number)
664 CONCAT ( -- 5. concatenating the latter
665                                     -- values plus hard-coded
666                                     -- values (opening and
667                                     -- closing parenthesis and a
668                                     -- hyphen)
669     '(', -- 5.1. value 1, the opening
670                                     -- parenthesis
671     LEFT (VendorPhone, 3), -- 5.2. value 2, area code
672     ') ', -- 5.3. value 3, the closing
673                                     -- parenthesis and a
674                                     -- space
675     SUBSTRING (VendorPhone, 4, 3), -- 5.4. value 4, branch
676                                     -- exchange
677     '-', -- 5.5. value 5, hyphen
678     RIGHT (VendorPhone, 4) -- 5.6. value 6, subscriber
679                                     -- number
680 ) AS legible_phone_number
681 FROM AP1.Vendors;

```

```

682
683
684 /* *****
685 https://folvera.commons.gc.cuny.edu/?p=1335
686 ***** */

```