```
/* ************************************************************************
                    DATABASE ADMINISTRATION FUNDAMENTALS:
                  INTRODUCTION TO STRUCTURED QUERY LANGUAGE
                    SF21SQL1001, 2021/11/02 - 2021/12/09
                 https://folvera.commons.gc.cuny.edu/?cat=29
************************************************************************** */

 SESSION #3 (2021/11/09): MANIPULATING DATA

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

1. A function, in any programming environment, lets you encapsulate reusable
   logic and build software that is ``composable``, i.e. built of pieces that
   can be reused and put together in a number of different ways to meet the
   needs of the users.  Functions hide the steps and the complexity from other
   code.
   https://www.simple-talk.com/sql/t-sql-programming/sql-server-functions-the-
    basics/

   1.1. Go to https://techonthenet.com/sql_server/functions/index_alpha.php
        for a detailed list of functions.

        1.1.1. As we mentioned before, so functions affect strings.

               CONCAT()    allows you to concatenate strings together
                           https://techonthenet.com/sql_server/functions/
                   concat.php
                           https://techonthenet.com/sql_server/functions/
                   concat2.php
               LEFT()      allows you to extract a substring from a string,
                           starting from the left-most character
                           https://techonthenet.com/sql_server/functions/
                   left.php
               LTRIM()     removes all space characters from the left-hand side
                           of a string
                           https://techonthenet.com/sql_server/functions/
                   ltrim.php
               LOWER()     converts all letters in the specified string to
                           lowercase
                           https://techonthenet.com/sql_server/functions/
                   lower.php
               REPLACE()   replaces a sequence of characters in a string with
                           another set of characters, not case-sensitive
                           https://techonthenet.com/sql_server/functions/
                   replace.php
               RIGHT()     allows you to extract a substring from a string,
                           starting from the right-most character
                           https://techonthenet.com/sql_server/functions/
                   right.php
```

```
45                  RTRIM()     removes all space characters from the right-hand
46                              side of a string
47                              https://techonthenet.com/sql_server/functions/       ↵
                        rtrim.php
48                  SUBSTRING() allows you to extract a substring from a string
49                              https://techonthenet.com/sql_server/functions/       ↵
                        substring.php
50                  UPPER()     converts all letters in the specified string to
51                              uppercase
52                              https://techonthenet.com/sql_server/functions/       ↵
                        upper.php
53
54          1.1.2. We also have functions that affect numeric values.
55
56                  AVG()       returns the average value of an expression
57                              https://techonthenet.com/sql_server/functions/avg.php
58                  CEILING()   returns the smallest integer value that is greater
59                              than or equal to a number
60                              https://techonthenet.com/sql_server/functions/       ↵
                        ceiling.php
61                  COUNT()     returns the count of an expression
62                              https://techonthenet.com/sql_server/functions/       ↵
                        count.php
63                  FLOOR()     returns the largest integer value that is equal to
64                              or less than a number
65                              https://techonthenet.com/sql_server/functions/       ↵
                        floor.php
66                  LEN()       returns the length of the specified string... does
67                              not include trailing space characters at the end the
68                              string when calculating the length
69                              https://techonthenet.com/sql_server/functions/len.php
70                  MAX()       returns the maximum value of an expression
71                              https://techonthenet.com/sql_server/functions/max.php
72                  MIN()       returns the minimum value of an expression
73                              https://techonthenet.com/sql_server/functions/min.php
74                  RAND()      returns a random number or a random number within a
75                              range
76                              https://techonthenet.com/sql_server/functions/       ↵
                        rand.php
77                  ROUND()     returns a number rounded to a certain number of
78                              decimal places
79                              https://techonthenet.com/sql_server/functions/       ↵
                        round.php
80                  SUM()       returns the summed value of an expression
81                              https://techonthenet.com/sql_server/functions/sum.php
82
83      1.2. Note that every time you have a function, you need parenthesis.  Go to
84           https://techonthenet.com/sql_server/functions/index_alpha.php for a
85           complete list of built-in functions.
86
87      1.3. As you might have noticed, some built-in functions manipulate strings.
88           When working with numerical values, first we would have to convert
```

```
89          them into strings as we will see later in the course.
90
91      1.4. Some other built-in functions ``return a single value, calculated from
92           values in a column``.  These are referred to as aggregate functions
93           (https://msdn.microsoft.com/en-us/library/ms173454.aspx).
94
95   2. Understanding the concepts above, we can now use them.
96
97      2.1. In the example below, we concatenate (put strings together) columns
98           `FirstName` and `LastName` from table `AP1.ContactUpdates`.
99   ********************************************************************** */
100
101 SELECT CONCAT (
102     FirstName,
103     ' ',
104     LastName
105     ) AS NAME
106 FROM AP1.ContactUpdates;
107
108
109 /* ************************************************************************
110     2.2. In the example below, we concatenate (put strings together) columns
111          `WE `, `ARE `, `LEARNING `, `SQL!`.
112  ********************************************************************** */
113
114 SELECT
115   CONCAT('WE ', 'ARE ', 'LEARNING ', 'SQL!');   -- returns `WE ARE LEARNING
116                                                 --          SQL!`
117
118
119 /* ************************************************************************
120     2.3. In the example below, we concatenate (put strings together) columns
121          `FirstName` and `LastName` from table `AP1.ContactUpdates`, just like
122          the previous example.
123
124          2.3.1. We also use `LTRIM()` and `RTRIM()` to remove leading and
125                 trailing spaces from `FirstName` with `LTRIM(RTRIM(FirstName))`
126                 and `LastName` with `LTRIM(RTRIM(LastName))`.
127  ********************************************************************** */
128
129 SELECT CONCAT (
130     LTRIM(RTRIM(LastName)),
131     ', ',
132     LTRIM(RTRIM(FirstName))
133     ) AS NAME
134 FROM AP1.ContactUpdates;
135
136
137 /* ************************************************************************
138     2.4. In the examples below, we use `UPPER()` to change a string to upper
139          case.
140  ********************************************************************** */
```

```
141
142 SELECT UPPER('this string is in upper case');   -- returns `THIS STRING SHOULD
143                                                 --          IN UPPER CASE`
144
145
146 /* ************************************************************************
147     2.5. In the examples below, we use `LOWER()` to change a string to lower
148          case.
149  ************************************************************************ */
150
151 SELECT LOWER('BUT THIS STRING IS IN LOWER CASE.');
152                                                 -- returns `but this string is
153                                                 --      in lower case.`
154
155
156 /* ************************************************************************
157     2.6. In the examples below, we use `RIGHT()` to extract characters from the
158          right.
159  ************************************************************************ */
160
161 SELECT RIGHT('apple', 2);                       -- returns `le`
162
163
164 /* ************************************************************************
165     2.7. In the examples below, we use `LEFT()` to extract characters from the
166          left.
167  ************************************************************************ */
168
169 SELECT LEFT('apple', 2);                        -- returns `ap`
170
171
172 /* ************************************************************************
173     2.8. In the examples below, we use `SUBSTRING()` to extract characters from
174          the middle -- same as built-in function `MID()` in other database
175          management systems like Oracle.
176  ************************************************************************ */
177
178 SELECT SUBSTRING('apple tree #5', 6, 10);       -- returns ` tree #5`
179
180
181 /* ************************************************************************
182     2.9. In the example below, we use `LEN()` to retrieve the length of a
183          string.
184  ************************************************************************ */
185
186 SELECT LEN('tree      #5');                     -- returns 12
187
188
189 /* ************************************************************************
190     2.10. In the examples below, we use `LTRIM()` and `RTRIM()` to remove any
191           leading and/or trailing spaces from the strings in single quotes.
192
```

```
193           2.10.1. Function `TRIM()` has not been implemented although
194                   advertised by Microsoft
195                   (https://docs.microsoft.com/en-us/sql/t-sql/functions/trim-
                       transact-sql).
196    ********************************************************************** */
197
198   SELECT LTRIM('       tree'),                    -- 1. trimming leading spaces
199     RTRIM('tree       '),                         -- 2. trimming trailing spaces
200     LTRIM(RTRIM('       tree       '));           -- 3. trimming leading and
201                                                    --    trailing spaces
202
203
204   /* ********************************************************************
205       2.11. In the example below, we use `REPLACE()` to replace pattern `mstake`
206             with `mistake`.  Since `mstake` exists in string `This is a mstake`,
207             `REPLACE()` returns `This is a mistake`.
208    ********************************************************************** */
209
210   SELECT REPLACE('This is a mstake', 'mstake', 'mistake');
211
212
213   /* ********************************************************************
214             2.11.1. In the example below, we use `REPLACE()` to replace pattern
215                     `gg` with `mistake`.  Since `gg` does not exist in `This is a
216                     mstake`, `REPLACE()` returns the original value.
217    ********************************************************************** */
218
219   SELECT REPLACE('This is a mstake', 'gg', 'mistake');
220
221
222   /* ********************************************************************
223       2.12. In the example below, since there is no function to make the first
224             letter of a string upper case and the rest lower case, we can use
225             a combination of functions `UPPER()`, `LOWER()`, `RIGHT()`, `LEFT()`
226             and `CONCAT()` working from the inside out.
227    ********************************************************************** */
228
229   SELECT CONCAT (
230       UPPER(LEFT('hELLO', 1)                       -- 1. retrieving first
231                                                    --    character from `hELLO`;
232                                                    --    returns `h`
233       )                                            -- 2. making `h` upper case;
234                                                    --    returns `H`
235       ,
236       LOWER(SUBSTRING('hELLO', 2, LEN('hELLO')     -- 3. retrieving variable
237         )                                          --    number of characters
238                                                    --    from character two (2)
239                                                    --    to the length of the
240                                                    --    string (integer value
241                                                    --    of 5); returns `ELLO`
242       )                                            -- 4. making `ELLO` lower
243                                                    --    case; returns `ello`
```

```
244     );                                          -- 5. concatenating all
245                                                 --    previous sections;
246                                                 --    returns `Hello`
247
248
249 /* ****************************************************************************
250     2.13. In the example below, we use `REPLACE()` to change pattern `  ` (two
251           spaces, `CHAR(32)+CHAR(32))` with ` ` (a single space, `CHAR(32)`).
252
253                     SELECT REPLACE('tree      #5', '  ', ' ');
254
255           2.12.1. Since string `tree      #5` has more than two spaces, we need
256                   run several passes of `REPLACE()`.
257
258           2.12.2. The statement runs from the inside out (3, 2, 1, 2, 3).
259
260                     function 3                -- 3. beginning of function #3:
261                                               --    * receiving value of
262                                               --      function #2
263                       function 2              -- 2. beginning of function #2:
264                                               --    * receiving value of
265                                               --      function #1
266                         function 1            -- 1. function #1:
267                                               --    * receiving original
268                                               --      value #0
269                                               --    * returning new value #1
270                       function 2              -- 2. end function of #2:
271                                               --    * returning new value #2
272                     function 3                -- 3. end function of #3:
273                                               --    * returning new value #3
274                                               --     (final value)
275  **************************************************************************** */
276
277
278 SELECT
279   REPLACE(                                    -- 3. pass #3 to replace
280                                               --    `CHAR(32)+CHAR(32)` for
281                                               --    `CHAR(32)`
282                                               --    * returns `tree #5` with
283                                               --      1 space
284     REPLACE(                                  -- 2. pass #2 to replace
285                                               --    `CHAR(32)+CHAR(32)` for
286                                               --    `CHAR(32)`
287                                               --    * returns `tree  #5` with
288                                               --      2 space, which feeds
289                                               --      pass #3
290       REPLACE('tree      #5',                 -- 1. pass #1 to replace
291                                               --    `CHAR(32)+CHAR(32)` for
292                                               --    `CHAR(32)`
293                                               --    * returns `tree   #5`
294                                               --      with 3 spaces, which
295       '  ', ' '),                             --      feeds pass #2
```

```
296      '  ', ' '),                                    -- 2. end of pass #2
297    '  ', ' ');                                      -- 3. end of pass #3
298
299
300  /* ************************************************************************
301      2.13. In the example below, we use `REPLACE()` to replace pattern `tree`
302            for `fruit`.
303
304            2.13.1. Since pattern `tree` exists in `      tree      `  with
305                    leading and trailing spaces around `tree`, `REPLACE()`
306                    returns `      fruit      ` with leading and trailing
307                    spaces around word `fruit`.
308
309            2.13.2. We also use `RTRIM()` and `LTRIM()` to remove trailing and
310                    leading spaces respectively to get `fruit` without leading
311                    and trailing spaces.
312   ************************************************************************ */
313
314  SELECT RTRIM(LTRIM(REPLACE('      tree      ', 'tree', 'fruit')));
315
316
317  /* ************************************************************************
318      2.14. In the example below, we use `REPLACE()` to replace pattern `Box` for
319            `PO Box`.
320
321            2.14.1. The first pass (inner) of `REPLACE()` changes some fields to
322                    `PO PO Box`.
323
324            2.14.2. The second pass (outer) of `REPLACE()` changes the previous
325                    error (`PO PO Box`) to `PO Box`.
326   ************************************************************************ */
327
328  SELECT AP1.Vendors.VendorID,                      -- 1. fields using format
329    AP1.Vendors.VendorName,                         --    `schema.table.field`
330    REPLACE(                                        -- 2. second pass of
331                                                    --    `REPLACE()` working from
332                                                    --    inside out
333      REPLACE(AP1.Vendors.VendorAddress1,           -- 3. first pass of `REPLACE()`
334      'Box', 'PO Box'),                             --    working from inside out
335    'PO PO Box', 'PO Box') AS VendorAddress1,
336    AP1.Vendors.VendorAddress2,
337    AP1.Vendors.VendorCity,
338    AP1.Vendors.VendorState,
339    AP1.Vendors.VendorZipCode,
340    AP1.Vendors.VendorPhone,
341    AP1.Vendors.VendorContactLName,
342    AP1.Vendors.VendorContactFName,
343    AP1.Vendors.DefaultTermsID,
344    AP1.Vendors.DefaultAccountNo,
345    AP1.Terms.TermsID,
346    AP1.Terms.TermsDescription,
347    AP1.Terms.TermsDueDays
```

```
348  FROM AP1.Vendors
349  INNER JOIN AP1.Terms                              -- 4. `INNER JOIN` to retrieve
350                                                    --    data in the first (left)
351                                                    --    table (`AP1.Vendors`)
352                                                    --    that is also in the
353                                                    --    second (right) table
354                                                    --    (`AP1.Terms`)
355    ON AP1.Vendors.DefaultTermsID = AP1.Terms.TermsID;
356                                                    -- 5. `ON` two fields with the
357                                                    --    same values/data, but in
358                                                    --    this case NOT the same
359                                                    --    name (`DefaultTermsID`
360                                                    --    and `TermsID`)
361
362
363  /* ***************************************************************************
364      2.14. In the example below, we use the functions that we have covered to
365            manipulate strings (any array of characters, such as letters and
366            numbers).
367   *************************************************************************** */
368
369  SELECT VendorID,
370    LEFT(VendorName, 8) AS VendorNameL,             -- 1. retrieving eight (8)
371                                                    --    characters from the left
372                                                    --    of each string value in
373                                                    --    column `VendorName`;
374                                                    --    returns `US Posta`
375                                                    --    (row 1)
376    RIGHT(VendorName, 8) AS VendorNameR,            -- 2. retrieving eight (8)
377                                                    --    characters from the right
378                                                    --    of each string value in
379                                                    --    column `VendorName`;
380                                                    --    returns ` Service`
381                                                    --    including the leading
382                                                    --    space (row 1)
383    CONCAT (                                        -- 3. concatenating the string
384      VendorAddress1,                               --    value in column
385      ' ',                                          --    `VendorAddress1`, a space
386      VendorAddress2                                --    and the value in
387      ) AS VendorAddess,                            --    column `VendorAddress2`;
388                                                    --    returns `PO Box 96621 `
389                                                    --    including the space since
390                                                    --    there was no value in
391                                                    --    `VendorAddress2` (row 2)
392    UPPER(VendorCity) AS VendorCity,                -- 4. changing the the string
393                                                    --    value in column
394                                                    --    `VendorCity` to upper
395                                                    --    case; returns `MADISON`
396                                                    --    (row 1)
397    LOWER(VendorState) AS VendorState,              -- 5. changing the the string
398                                                    --    value in column
399                                                    --    `VendorState` to lower
```

```
400                                                 --    case; returns `dc`
401                                                 --    (row 2)
402    VendorZipCode,
403    SUBSTRING(VendorPhone, 4, 3) AS VendorPhone,  -- 6. retrieving three (3)
404                                                 --    characters starting from
405                                                 --    the character four (4)
406                                                 --    of each string value in
407                                                 --    column `VendorName`;
408                                                 --    returns `555` (row 1) and
409                                                 --    `255` (row 73)
410    REPLACE(VendorContactLName, 'en', 'XX')      -- 7. replacing pattern `en` in
411      AS VendorContactLName,                     --    each string value in
412                                                 --    column
413                                                 --    `VendorContactLName` with
414                                                 --    pattern `XX` when found;
415                                                 --    returns `MaegXX` (row 7)
416                                                 --    and `AileXX` (row 16)
417    VendorContactFName,
418    LEN(VendorContactFName)                      -- 8. retrieving the length as
419      AS VendorContactFNameLEN,                  --    an integer of each string
420                                                 --    value in column
421                                                 --    `VendorContactFName`;
422                                                 --    returns 9 (row 1)
423    DefaultTermsID,
424    DefaultAccountNo
425  FROM AP1.Vendors;
426
427
428  /* ***************************************************************************
429      2.14. In the example below, we write a query (`SELECT` statement) calling
430            all shared data (`INNER JOIN`) from tables `AP1.Vendors`,
431            `AP1.Invoices` and `AP1.InvoiceLineItems` using the following syntax.
432
433                      SELECT table1.field1,
434                        table1.field2 ...
435                        table2.field1,
436                        table2.field2 ...
437                        table3.field1,
438                        table3.field2 ...
439                      FROM table1
440                      INNER JOIN table2
441                        ON table1.common_field1(id1) = table2.common_field1(id1)
442                      INNER JOIN table3
443                        ON table1.common_field2(id2) = table3.common_field2(id2);
444
445            Then we can delete or rename using an alias the duplicate name of the
446            columns.
447   *************************************************************************** */
448
449  SELECT AP1.Vendors.VendorID,
450    AP1.Vendors.VendorName,
451    AP1.Vendors.VendorAddress1,
```

```
452    AP1.Vendors.VendorAddress2,
453    AP1.Vendors.VendorCity,
454    AP1.Vendors.VendorState,
455    AP1.Vendors.VendorZipCode,
456    AP1.Vendors.VendorPhone,
457    AP1.Vendors.VendorContactLName,
458    AP1.Vendors.VendorContactFName,
459    AP1.Vendors.DefaultTermsID,
460    AP1.Vendors.DefaultAccountNo,
461    AP1.Invoices.InvoiceID,
462    -- AP1.Invoices.VendorID,                    -- 1. duplicate column name
463                                                --    (`VendorID`), which can
464                                                --    be removed (commented
465                                                --    out, in this case)
466                                                --    without affecting the
467                                                --    query output; could also
468                                                --    be renamed
469    AP1.Invoices.InvoiceNumber,
470    AP1.Invoices.InvoiceDate,
471    AP1.Invoices.InvoiceTotal,
472    AP1.Invoices.PaymentTotal,
473    AP1.Invoices.CreditTotal,
474    AP1.Invoices.TermsID,
475    AP1.Invoices.InvoiceDueDate,
476    AP1.Invoices.PaymentDate,
477    -- AP1.InvoiceLineItems.InvoiceID,           -- 2. duplicate column name
478                                                --    (`InvoiceID`), which can
479                                                --    be removed (commented
480                                                --    out, in this case)
481                                                --    without affecting the
482                                                --    query output; could also
483                                                --    be renamed
484    AP1.InvoiceLineItems.InvoiceSequence,
485    AP1.InvoiceLineItems.AccountNo,
486    AP1.InvoiceLineItems.InvoiceLineItemAmount,
487    AP1.InvoiceLineItems.InvoiceLineItemDescription
488  FROM AP1.Vendors                              -- 3. from table `AP1.Vendors`
489  INNER JOIN AP1.Invoices                       -- 4. `INNER JOIN` to retrieve
490                                                --    data in the first (left)
491                                                --    table (`AP1.Vendors`)
492                                                --    that is also in the
493                                                --    second (right) table
494                                                --    (`AP1.Invoices`)
495    ON AP1.Vendors.VendorID = AP1.Invoices.VendorID
496                                                -- 5. `ON` two fields with the
497                                                --    same values/data and the
498                                                --    same name (`VendorID`);
499                                                --    specifying the relation
500                                                --    between tables
501                                                --    `AP1.Vendors` and
502                                                --    `AP1.Invoices`
503  INNER JOIN AP1.InvoiceLineItems               -- 6. `INNER JOIN` to retrieve
```

```
504                                                 --    data in the first (left)
505                                                 --    table (`AP1.Invoices`)
506                                                 --    that is also in the
507                                                 --    second (right) table
508                                                 --    (`AP1.InvoiceLineItems`)
509   ON AP1.Invoices.InvoiceID = AP1.InvoiceLineItems.InvoiceID;
510                                                 -- 7. `ON` two fields with the
511                                                 --    same values/data and the
512                                                 --    same name (`InvoiceID`);
513                                                 --    specifying the relation
514                                                 --    between tables
515                                                 --    `AP1.Invoices` and
516                                                 --    `AP1.InvoiceLineItems`
517
518
519 /* ***************************************************************************
520  4. LAB #2
521     Write a query without duplicate rows (`SELECT DISTINCT`)
522     4.1. to call all columns from `AP1.Vendors` and `AP1.Invoices`, shared data
523          only (`INNER JOIN`)
524     4.2. to present `VendorPhone` in `(123) 456-7890` structure, making sure to
525          avoid NULLs and logical errors (`() -`).
526  *************************************************************************** */
527
528 SELECT DISTINCT                                 -- 1. to retrieve unique rows
529   AP1.Vendors.VendorID,
530   AP1.Vendors.VendorName,
531   AP1.Vendors.VendorAddress1,
532   AP1.Vendors.VendorAddress2,
533   AP1.Vendors.VendorCity,
534   AP1.Vendors.VendorState,
535   AP1.Vendors.VendorZipCode,
536   CASE                                          -- 2. start of `CASE` clause
537     WHEN AP1.Vendors.VendorPhone IS NOT NULL    -- 3. condition #1: when field
538                                                 --    has a value
539       THEN CONCAT (                             -- 4. what action to take:
540           '(',                                  --    4.1. concatenation of a
541                                                 --         opening parenthesis
542           LEFT(AP1.Vendors.VendorPhone, 3),     --    4.2. three (3) characters
543                                                 --         from the left of
544                                                 --         `VendorPhone`
545           ') ',                                 --    4.3. closing parenthesis
546           SUBSTRING(AP1.Vendors.VendorPhone,4,3),--   4.4. substring from
547                                                 --         character four (4)
548                                                 --         taking three (3)
549                                                 --         characters of
550                                                 --         `VendorPhone`
551           '-',                                  --    4.5. a hyphen between
552                                                 --         branch and
553                                                 --         subscriber number
554           RIGHT(AP1.Vendors.VendorPhone, 4)     --    4.6. four (4) characters
555                                                 --         from the right of
```

```
556                                                 --         `VendorPhone`
557             )                                   -- 5. end of `CONCAT` function
558       ELSE ''                                   -- 6. escape condition in case
559                                                 --    previous conditions fail;
560                                                 --    returning an empty string
561       END AS VendorPhone,                       -- 7. end of `CASE` clause with
562                                                 --    alias `VendorPhone`
563     AP1.Vendors.VendorContactLName,
564     AP1.Vendors.VendorContactFName,
565     AP1.Vendors.DefaultTermsID,
566     AP1.Vendors.DefaultAccountNo,
567     AP1.Invoices.InvoiceID,
568     -- AP1.Invoices.VendorID AS Expr1,           -- 8. duplicate field
569                                                 --    `VendorID` commented out
570     AP1.Invoices.InvoiceNumber,
571     AP1.Invoices.InvoiceDate,
572     AP1.Invoices.InvoiceTotal,
573     AP1.Invoices.PaymentTotal,
574     AP1.Invoices.CreditTotal,
575     AP1.Invoices.TermsID,
576     AP1.Invoices.InvoiceDueDate,
577     AP1.Invoices.PaymentDate
578   FROM AP1.Vendors
579   INNER JOIN AP1.Invoices
580     ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;
581
582   /* ***************************************************************************
583    https://folvera.commons.gc.cuny.edu/?p=997
584    ************************************************************************** */
```