```
/* ************************************************************************
          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 #5 (2023/04/17): MANIPULATING DATA

  1. Using clauses `BETWEEN`, `NOT`, `UNION`, `EXCEPT` and `INTERSECT`
  2. Understanding function `FORMAT()` for dates and currencies including
     culture codes
 ************************************************************************

 1. LAB #3

    Write a query
    1.01. to call all columns and values from `AP1.Vendors` and any related
          values from `AP1.ContactUpdates` (`LEFT JOIN`),
    1.02. to put together `FirstName` and `LastName` in one field with alias
          `ContactName`,
    1.03. to put together the first letter of `FirstName`, the complete
         `LastName`, `@`, `VendorName` (removing empty spaces between words and
         special characters like `&` and `,`) and `.com` as `ContactEmail`
         presenting the output in lower case,
    1.04. and to put together `VendorContactFName` and `VendorContactLName`
          with aliases `VendorContactName` and `VendorContactEmail` (like
          #1.02).
 ************************************************************************ */

SELECT AP1.Vendors.VendorID,
  AP1.Vendors.VendorName,
  AP1.Vendors.VendorAddress1,
  AP1.Vendors.VendorAddress2,
  AP1.Vendors.VendorCity,
  AP1.Vendors.VendorState,
  AP1.Vendors.VendorZipCode,
  AP1.Vendors.VendorPhone,
  AP1.Vendors.VendorContactFName +              -- 1. concatenation of
  ' ' +                                         --    `FirstName`, a single
  AP1.Vendors.VendorContactLName                --    space and `LastName`
   AS ContactName,                              --    using `+` with alias
                                                --    `ContactName`
   LEFT(AP1.Vendors.VendorContactFName, 1) +    -- 2. concatenation of
    AP1.Vendors.VendorContactLName +            --    `VendorContactFName`,
    '@' +                                       --    a single space and
                                                --    `VendorContactLName`,
                                                --    the `@` sign, followed by
                                                --    `VendorName` (value #6)
    REPLACE(                                    --    2.01. getting value #5
      REPLACE(                                  --    2.02. getting value #4
        REPLACE(                                --    2.03. getting value #3
          REPLACE(                              --    2.04. getting value #2
```

```
            REPLACE(                                --    2.05. getting value #1
              REPLACE(AP1.Vendors.VendorName, ' ', '')
                                                    --    2.06. generating value #1
              , '.', '')                            --    2.07. generating value #2
            , ',', '')                              --    2.08. generating value #3
          , '/', '')                                --    2.09. generating value #4
        , '&', '')                                  --    2.10. generating value #5
      , '''', '') +                                 --    2.11. generating value #6
                                                    --          where `''`
                                                    --          represents a single
                                                    --          quote (`''`) used an
                                                    --          escape character
    '.foo'                                          --    and `.foo` to complete
    AS VendorContactEmail,                          --    email with alias
                                                    --    `VendorContactEmail`
  AP1.Vendors.DefaultTermsID,
  AP1.Vendors.DefaultAccountNo,
  -- AP1.ContactUpdates.VendorID AS Expr1,
  AP1.ContactUpdates.FirstName +                    -- 3. concatenation of
  ' ' +                                             --    `VendorContactFName`, a
  AP1.ContactUpdates.LastName                       --    single space and
    AS ContactName,                                 --    `LastName` using `+` with
                                                    --    alias `VendorContactLName`
  LEFT(AP1.ContactUpdates.FirstName, 1) +           -- 4. concatenation of
  AP1.ContactUpdates.LastName +                     -- `VendorContactFName`,
  '@' +                                             --    a single space and
                                                    --    `VendorContactLName`,
                                                    --    the `@` sign, followed by
                                                    --    `VendorName` (value #6)
  REPLACE(                                          --    4.01. getting value #5
    REPLACE(                                        --    4.02. getting value #4
      REPLACE(                                      --    4.03. getting value #3
        REPLACE(                                    --    4.04. getting value #2
          REPLACE(                                  --    4.05. getting value #1
            REPLACE(AP1.Vendors.VendorName, ' ', '')
                                                    --    4.06. generating value #1
            , '.', '')                              --    4.07. generating value #2
          , ',', '')                                --    4.08. generating value #3
        , '/', '')                                  --    4.09. generating value #4
      , '&', '')                                    --    4.10. generating value #5
    , '''', '') +                                   --    4.11. generating value #6
                                                    --          where `''`
                                                    --          represents a single
                                                    --          quote (`''`) used an
                                                    --          escape character
  '.foo'                                            --    and `.foo` to complete
    AS VendorContactEmail                           --    email with alias
                                                    --    `VendorContactEmail`
FROM AP1.Vendors
LEFT JOIN AP1.ContactUpdates
  ON AP1.Vendors.VendorID = AP1.ContactUpdates.VendorID;
```

```
105
106 /* ************************************************************************
107           As with previous example, we can use an alias for each table, which
108           in this case, allows us to present neater code.
109
110                     `v` for `AP1.Vendors`
111                     `c` for `AP1.ContactUpdates`
112  ************************************************************************ */
113
114 SELECT v.VendorID,
115   v.VendorName,
116   v.VendorAddress1,
117   v.VendorAddress2,
118   v.VendorCity,
119   v.VendorState,
120   v.VendorZipCode,
121   v.VendorPhone,
122   v.VendorContactFName +
123     ' ' +
124     v.VendorContactLName AS ContactName,
125   LEFT(v.VendorContactFName, 1) +
126     v.VendorContactLName + '@' +
127     REPLACE(
128       REPLACE(
129         REPLACE(
130           REPLACE(
131             REPLACE(
132               REPLACE(v.VendorName, ' ', '')
133             , '.', '')
134           , ',', '')
135         , '/', '')
136       , '&', '')
137     , '''', '')
138     + '.foo' AS VendorContactEmail,
139   v.DefaultTermsID,
140   v.DefaultAccountNo,
141   -- c.VendorID AS Expr1,
142   c.FirstName +
143   ' '+
144   c.LastName AS ContactName,
145   LEFT(c.FirstName, 1) +
146     c.LastName +
147     '@' +
148     REPLACE(
149       REPLACE(
150         REPLACE(
151           REPLACE(
152             REPLACE(
153               REPLACE(v.VendorName, ' ', '')
154             , '.', '')
155           , ',', '')
156         , '/', '')
```

```
157       , '&', '')
158     , '''', '')
159     + '.foo' AS VendorContactEmail
160 FROM AP1.Vendors AS v
161 LEFT JOIN AP1.ContactUpdates AS c
162   ON v.VendorID = c.VendorID;
163
164
165 /* ************************************************************************
166           Instead of using a plus sign (`+`), we can use `CONCAT()` since
167           adding a value and NULL returns NULL.  In other words, we lose data,
168           which would be logical error (not syntax error).
169  ************************************************************************ */
170
171 SELECT AP1.Vendors.VendorID,
172   AP1.Vendors.VendorName,
173   AP1.Vendors.VendorAddress1,
174   AP1.Vendors.VendorAddress2,
175   AP1.Vendors.VendorCity,
176   AP1.Vendors.VendorState,
177   AP1.Vendors.VendorZipCode,
178   AP1.Vendors.VendorPhone,
179   CONCAT (                                        -- 1. concatenation of
180     AP1.Vendors.VendorContactFName,               --    `FirstName`, a single
181     ' ',                                          --    space and `LastName` with
182     AP1.Vendors.VendorContactLName                --    alias `ContactName`
183     ) AS ContactName,
184   CONCAT (                                        -- 2. concatenation of
185     LEFT(AP1.Vendors.VendorContactFName, 1),      --    `VendorContactFName`,
186     AP1.Vendors.VendorContactLName,               --    a single space and
187     '@',                                          --    `VendorContactLName`,
188                                                   --    the `@` sign, followed by
189                                                   --    `VendorName` (value #6)
190     REPLACE(                                      --    2.01. getting value #5
191       REPLACE(                                    --    2.02. getting value #4
192         REPLACE(                                  --    2.03. getting value #3
193           REPLACE(                                --    2.04. getting value #2
194             REPLACE(                              --    2.05. getting value #1
195               REPLACE(AP1.Vendors.VendorName, ' ', '')
196                                                   --    2.06. generating value #1
197               , '.', '')                          --    2.07. generating value #2
198             , ',', '')                            --    2.08. generating value #3
199           , '/', '')                              --    2.09. generating value #4
200         , '&', '')                                --    2.10. generating value #5
201       , '''', '')                                 --    2.11. generating value #6
202                                                   --          where `'''`
203                                                   --          represents a single
204                                                   --          quote (`''`) used an
205                                                   --          escape character
206     , '.foo'                                      --    and `.foo` to complete
207     ) AS VendorContactEmail,                      --    email with alias
208                                                   --    `VendorContactEmail`
```

```
209    AP1.Vendors.DefaultTermsID,
210    AP1.Vendors.DefaultAccountNo,
211    -- AP1.ContactUpdates.VendorID AS Expr1,
212    CONCAT (                                        -- 3. concatenation of
213      AP1.ContactUpdates.FirstName,                 --    `VendorContactFName`, a
214      ' ',                                          --    single space and
215      AP1.ContactUpdates.LastName                   --    `LastName` with alias
216      ) AS ContactName,                             --    `VendorContactLName`
217    CONCAT (                                        -- 4. concatenation of
218      LEFT(AP1.ContactUpdates.FirstName, 1),        --    `VendorContactFName`,
219      AP1.ContactUpdates.LastName,                  --    a single space and
220      '@',                                          --    `VendorContactLName`,
221                                                    --    the `@` sign, followed by
222                                                    --    `VendorName` (value #6)
223      REPLACE(                                      --    4.01. getting value #5
224        REPLACE(                                    --    4.02. getting value #4
225          REPLACE(                                  --    4.03. getting value #3
226            REPLACE(                                --    4.04. getting value #2
227              REPLACE(                              --    4.05. getting value #1
228                REPLACE(AP1.Vendors.VendorName, ' ', '')
229                                                    --    4.06. generating value #1
230                , '.', '')                          --    4.07. generating value #2
231              , ',', '')                            --    4.08. generating value #3
232            , '/', '')                              --    4.09. generating value #4
233          , '&', '')                                --    4.10. generating value #5
234        , '''', ''),                                --    4.11. generating value #6
235                                                    --          where `''`
236                                                    --          represents a single
237                                                    --          quote (`''`) used an
238                                                    --          escape character
239      '.foo'                                        --    and `.foo` to complete
240      ) AS VendorContactEmail                       --    email with alias
241                                                    --    `VendorContactEmail`
242  FROM AP1.Vendors
243  LEFT JOIN AP1.ContactUpdates
244    ON AP1.Vendors.VendorID = AP1.ContactUpdates.VendorID;
245
246
247  /* ***************************************************************************
248           Using `CONCAT()` also returns a logical error (not syntax error)
249           since the concatenation to make the second email returns values like
250           `@USPostalService.foo` since there is no corresponding `FirstName`
251           and `LastName`.  We could use a CASE clause.
252   *************************************************************************** */
253
254  SELECT AP1.Vendors.VendorID,
255    AP1.Vendors.VendorName,
256    AP1.Vendors.VendorAddress1,
257    AP1.Vendors.VendorAddress2,
258    AP1.Vendors.VendorCity,
259    AP1.Vendors.VendorState,
260    AP1.Vendors.VendorZipCode,
```

```
261    AP1.Vendors.VendorPhone,
262    CONCAT (                                        -- 1. concatenation of
263      AP1.Vendors.VendorContactFName,               --    `FirstName`, a single
264      ' ',                                          --    space and `LastName` with
265      AP1.Vendors.VendorContactLName                --    alias `ContactName`
266      ) AS ContactName,
267    CONCAT (                                        -- 2. concatenation of
268      LEFT(AP1.Vendors.VendorContactFName, 1),      --    `VendorContactFName`,
269      AP1.Vendors.VendorContactLName,               --    a single space and
270      '@',                                          --    `VendorContactLName`,
271                                                    --    the `@` sign, followed by
272                                                    --    `VendorName` (value #6)
273      REPLACE(                                      --    2.01. getting value #5
274        REPLACE(                                    --    2.02. getting value #4
275          REPLACE(                                  --    2.03. getting value #3
276            REPLACE(                                --    2.04. getting value #2
277              REPLACE(                              --    2.05. getting value #1
278                REPLACE(AP1.Vendors.VendorName, ' ', '')
279                                                    --    2.06. generating value #1
280                , '.', '')                          --    2.07. generating value #2
281              , ',', '')                            --    2.08. generating value #3
282            , '/', '')                              --    2.09. generating value #4
283          , '&', '')                                --    2.10. generating value #5
284        , '''', '')                                 --    2.11. generating value #6
285                                                    --          where `'''`
286                                                    --          represents a single
287                                                    --          quote (`''`) used an
288                                                    --          escape character
289      , '.foo'                                      --    and `.foo` to complete
290      ) AS VendorContactEmail,                      --    email with alias
291                                                    --    `VendorContactEmail`
292    AP1.Vendors.DefaultTermsID,
293    AP1.Vendors.DefaultAccountNo,
294    -- AP1.ContactUpdates.VendorID AS Expr1,
295    CONCAT (                                        -- 3. concatenation of
296      AP1.ContactUpdates.FirstName,                 --    `VendorContactFName`, a
297      ' ',                                          --    single space and
298      AP1.ContactUpdates.LastName                   --    `LastName` with alias
299      ) AS ContactName,                             --    `VendorContactLName`
300
301    CASE
302      WHEN (AP1.ContactUpdates.FirstName <> ''      -- 4. checking if
303        OR AP1.ContactUpdates.FirstName <> ' '      --    `ContactUpdates` is not
304        OR AP1.ContactUpdates.FirstName IS NOT NULL)--  an empty string or not
305                                                    --    equal to a space or
306                                                    --    `IS NOT NULL` (in other
307                                                    --    words, not no-value; must
308                                                    --    have a value) using
309                                                    --    parenthesis to make a
310                                                    --    block of three conditions
311        AND (AP1.ContactUpdates.LastName <> ''      --    and that `ContactUpdates`
312          OR AP1.ContactUpdates.LastName <> ' '     --    is not an empty string or
```

```
313         OR AP1.ContactUpdates.LastName IS NOT NULL)-- not equal to a space or
314                                                    --    `IS NOT NULL` (in other
315                                                    --    words, not no-value; must
316                                                    --    have a value) using
317                                                    --    parenthesis to make a
318                                                    --    block of three conditions
319     THEN CONCAT (                                  -- 4. action to be taken if
320         LEFT(AP1.ContactUpdates.FirstName, 1),     --    previous condition is
321         AP1.ContactUpdates.LastName,               --    true is the concatenation
322         '@',                                       --    of `VendorContactFName`,
323                                                    --    a single space and
324                                                    --    `VendorContactLName`, the
325                                                    --    `@` sign, followed by
326                                                    --    `VendorName` (value #6)
327         REPLACE(                                   --    4.01. getting value #5
328           REPLACE(                                 --    4.02. getting value #4
329             REPLACE(                               --    4.03. getting value #3
330               REPLACE(                             --    4.04. getting value #2
331                 REPLACE(                           --    4.05. getting value #1
332                   REPLACE(AP1.Vendors.VendorName, ' ', '')
333                                                    --    4.06. generating value #1
334                   , '.', '')                       --    4.07. generating value #2
335                 , ',', '')                         --    4.08. generating value #3
336               , '/', '')                           --    4.09. generating value #4
337             , '&', '')                             --    4.10. generating value #5
338           , '''', ''),                             --    4.11. generating value #6
339                                                    --          where `''`
340                                                    --          represents a single
341                                                    --          quote (`''`) used an
342                                                    --          escape character
343         '.foo'                                     --    and `.foo` to complete
344         )
345     ELSE ''
346     END                                            --    end of CASE clause to
347     AS VendorContactEmail                          --    make email with alias
348                                                    --    `VendorContactEmail`
349 FROM AP1.Vendors
350 LEFT JOIN AP1.ContactUpdates
351   ON AP1.Vendors.VendorID = AP1.ContactUpdates.VendorID;
352
353
354 /* ***************************************************************************
355           As with previous examples, we can use an alias for each table, which
356           in this case, allows us to present neater code.
357
358                     `v` for `AP1.Vendors`
359                     `c` for `AP1.ContactUpdates`
360  *************************************************************************** */
361
362 SELECT v.VendorID,
363   v.VendorName,
364   v.VendorAddress1,
```

```
365     v.VendorAddress2,
366     v.VendorCity,
367     v.VendorState,
368     v.VendorZipCode,
369     v.VendorPhone,
370     CONCAT (
371       v.VendorContactFName,
372       ' ',
373       v.VendorContactLName
374       ) AS ContactName,
375     CONCAT (
376       LEFT(v.VendorContactFName, 1),
377       v.VendorContactLName,
378       '@',
379       REPLACE(
380         REPLACE(
381           REPLACE(
382             REPLACE(
383               REPLACE(
384                 REPLACE(v.VendorName, ' ', '')
385               , '.', '')
386             , ',', '')
387           , '/', '')
388         , '&', '')
389       , '''', ''),
390       '.foo'
391       ) AS VendorContactEmail,
392     v.DefaultTermsID,
393     v.DefaultAccountNo,
394     -- c.VendorID AS Expr1,
395     CONCAT (
396       c.FirstName,
397       ' ',
398       c.LastName
399       ) AS ContactName,
400     CASE
401       WHEN (
402           c.FirstName <> ''
403           OR c.FirstName <> ' '
404           OR c.FirstName IS NOT NULL
405           )
406         AND (
407           c.LastName <> ''
408           OR c.LastName <> ' '
409           OR c.LastName IS NOT NULL
410           )
411         THEN CONCAT (
412             LEFT(c.FirstName, 1),
413             c.LastName,
414             REPLACE(
415               REPLACE(
416                 REPLACE(
```

```
                REPLACE(
                  REPLACE(
                    REPLACE(v.VendorName, ' ', '')
                  , '.', '')
                , ',', '')
              , '/', '')
            , '&', '')
          , '''', ''),
          '.foo'
          )
    ELSE ''
    END AS VendorContactEmail
FROM AP1.Vendors AS v
LEFT JOIN AP1.ContactUpdates AS c
  ON v.VendorID = c.VendorID;


/* ******************************************************************************
 2. Before you continue learning about SQL
    (https://searchsqlserver.techtarget.com/definition/SQL) syntax
    (https://whatis.techtarget.com/definition/syntax), we should cover some
    important theory, which you will need whether you need to learn SQL to run
    queries at work and/or you decide to become a database administrator (DBA).

    2.01. SQL (Structured Query Language) is a standardized programming
          language used for managing relational databases and performing
          various operations on the data in them.  Initially created in the
          1970s, SQL is regularly used by database administrators, as well as
          by developers writing data integration scripts and data analysts
          looking to set up and run analytical queries.
          https://searchsqlserver.techtarget.com/definition/SQL

    2.02. ISO/IEC 9075-1:2016 [SQL:2016] describes the conceptual framework
          used in other parts of ISO/IEC 9075 to specify the grammar of SQL and
          the result of processing statements in that language by an
          SQL-implementation.
          ISO/IEC 9075-1:2016 also defines terms and notation used in the other
          parts of ISO/IEC 9075.
          https://www.iso.org/standard/63555.html

    2.03. T-SQL (Transact-SQL) is a set of programming extensions from Sybase
          and Microsoft that add several features to the Structured Query
          Language (SQL), including transaction control, exception and error
          handling, row processing and declared variables.
          https://searchsqlserver.techtarget.com/definition/T-SQL

    2.04. A relational database is a set of tables containing data fitted into
          predefined categories.  Each table (which is sometimes called a
          relation) contains one or more data categories in columns.  Each row
          contains a unique instance of data for the categories defined by the
          columns.
          http://searchsqlserver.techtarget.com/definition/relational-database
```

```
469
470     2.05. Microsoft SQL Server is a relational database management system, or
471           RDBMS, that supports a wide variety of transaction processing,
472           business intelligence and analytics applications in corporate IT
473           environments.  It's one of the three market-leading database
474           technologies, along with Oracle Database and IBM's DB2.
475           Like other RDBMS software, Microsoft SQL Server is built on top of
476           SQL, a standardized programming language that database administrators
477           (DBAs) and other IT professionals use to manage databases and query
478           the data they contain.  SQL Server is tied to Transact-SQL (T-SQL),
479           an implementation of SQL from Microsoft that adds a set of
480           proprietary programming extensions to the standard language.  The
481           original SQL Server code was developed in the 1980s by the former
482           Sybase Inc., which is now owned by SAP.  Sybase initially built the
483           software to run on Unix systems and minicomputer platforms.  It,
484           Microsoft and Ashton-Tate Corp., then the leading vendor of PC
485           databases, teamed up to produce the first version of what became
486           Microsoft SQL Server, designed for the OS/2 operating system and
487           released in 1989.
488           https://searchsqlserver.techtarget.com/definition/SQL-Server
489
490     2.06. Another form of flat file is one in which table data is gathered in
491           lines of ASCII text with the value from each table cell separated by
492           a comma and each row represented with a new line.  This type of flat
493           file is also known as a comma-separated values file (CSV) file.
494           http://searchsqlserver.techtarget.com/definition/flat-file
495
496     2.07. A hierarchical database is a design that uses a one-to-many
497           relationship for data elements.  Hierarchical database models use a
498           tree structure that links a number of disparate elements to one
499           `owner,` or `parent,` primary record.
500           https://www.techopedia.com/definition/19782/hierarchical-database
501
502     2.08. Data Manipulation Language (DML) is the ``vocabulary used to retrieve
503           and work with data... to add, modify, query, or remove data``
504           (https://msdn.microsoft.com/en-us/library/ff848766.aspx).
505
506           SELECT    to retrieve records from one or more tables
507                     https://techonthenet.com/sql/select.php
508
509           INSERT    to insert a one or more records into a table
510                     https://techonthenet.com/sql/insert.php
511
512           UPDATE    to update existing records in the tables
513                     https://techonthenet.com/sql/update.php
514
515           DELETE    to delete a one or more records from a table
516                     https://techonthenet.com/sql/delete.php
517
518           MERGE     to insert, update, or delete operations on a target table
519                     based on the results of a join with a source table
520                     https://msdn.microsoft.com/en-us/library/bb510625.aspx
```

```
521
522     2.09. Data Definition Language (DDL) is the ``vocabulary used to define
523           data structures... to create, alter, or drop data structures``
524           (https://msdn.microsoft.com/en-us/library/ff848799.aspx).
525
526           USE       to select any existing database in SQL schema [or output
527                     from another query]
528                     http://tutorialspoint.com/sql/sql-select-database.htm
529
530           CREATE    to create and define a table [or other database object]
531                     https://techonthenet.com/sql/tables/create_table.php
532
533           ALTER     to add a column, modify a column, drop a column, rename a
534                     column or rename a table [or other database object]
535                     https://techonthenet.com/sql/tables/alter_table.php
536
537           DROP      to remove or delete a table [or other database object]
538                     https://techonthenet.com/sql/tables/drop_table.php
539
540           TRUNCATE  to remove all records from a table
541                     https://techonthenet.com/sql/truncate.php
542
543           DELETE    to delete a one or more records from a table
544                     https://techonthenet.com/sql/delete.php
545
546     2.10. Note that some of these statements can do more than what is covered
547           in these notes for our first sessions.
548
549           For example, the `CREATE` statement is also used to create other
550           database objects as well as access management, but we will not cover
551           these other statements yet.  Refer to
552           https://msdn.microsoft.com/en-us/library/cc879262.aspx for more
553           information on the `CREATE` statement.
554
555           On a personal note, when looking for information and/or explanation
556           on how to use Microsoft technologies, in this case SQL Server, go to
557           https://techonthenet.com/ or http://tutorialspoint.com/ as
558           https://msdn.microsoft.com/ and other Microsoft websites often seem
559           to be written for advanced users.
560
561           We will use DML and DDL in detail later in the course.
562
563   3. There are several data types
564      (https://msdn.microsoft.com/en-us/library/ms187752.aspx) that you need to
565      know if you are interested in taking the certification exam for Database
566      Fundamentals.  In everyday use, these are the most often used data types in
567      T-SQL (http://searchsqlserver.techtarget.com/definition/T-SQL) -- the
568      version of SQL (http://searchsqlserver.techtarget.com/definition/SQL) used
569      in SQL Server (http://searchsqlserver.techtarget.com/definition/SQL-Server)
570      -- are the following.
571
572           INT       -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)
```

```
                    https://technet.microsoft.com/en-us/library/ms187745.aspx

          DECIMAL   fixed precision and scale numbers, 10^38+1 through 10^38-1
                    https://msdn.microsoft.com/en-us/library/ms187746.aspx
                    * instead of DOUBLE or FLOAT, indicating the whole value
                      followed by the number of decimals where pi(1,10) can
                      hold 3.1415926536, but not 3.14159265359 for eleven (11)
                      decimal spaces

          VARCHAR(n) 2^31-1 bytes (2 GB); variable-length, ASCII
                    (http://whatis.techtarget.com/definition/ASCII-American-
                    Standard-Code-for-Information-Interchange)
                    string data
                    https://technet.microsoft.com/en-us/library/ms176089.aspx
                    not to be confused with NVARCHAR(n) -- variable-length,
                    2^31-1 bytes (2 GB), Unicode
                    (http://whatis.techtarget.com/definition/Unicode) string
                    data, not part of most relational database management
                    systems (RDBMS)
                    https://technet.microsoft.com/en-us/library/ms186939.aspx

          DATE      date
                    https://technet.microsoft.com/en-us/library/bb630352.aspx

          TIME      time
                    https://technet.microsoft.com/en-us/library/bb677243.aspx

          DATETIME  defines a date that is combined with a time of day with
                    fractional seconds that is based on a 24-hour clock
                    https://technet.microsoft.com/en-us/library/ms187819.aspx

          MONEY     money, not part of most relational database management
                    systems (RDBMS)
                    https://technet.microsoft.com/en-us/library/ms179882.aspx

     3.01. Conversion may only take place between data similar types.

                    +----------------------+--------------------------------+
                    | CONVERSION INPUT     | CONVERSION OUTPUT              |
                    +----------------------+--------------------------------+
                    | INT      to  DECIMAL | no loss;  decimal spaces added |
                    |                      | (.00)                          |
                    +----------------------+--------------------------------+
                    | DECIMAL  to  INT     | possible loss of decimal spaces;|
                    |                      | truncated, value not rounded up |
                    |                      | or down                        |
                    +----------------------+--------------------------------+
                    | DECIMAL  to  MONEY   | truncated and rounded to four  |
                    |                      | decimal spaces for mathematical |
                    |                      | calculations (.0000 to .9999);  |
                    |                      | two decimal spaces shown for   |
                    |                      | cents (.00 to .99)             |
```

DATETIME to DATE	date only; time dropped
DATETIME to TIME	time only; date dropped
DATE to DATETIME	date with default value of `00:00.00.000`
TIME to DATETIME	time with default value of `1900/01/01`
INT DECIMAL DATETIME to VARCHAR DATE NVARCHAR TIME	converted to text; no longer numeric data and cannot be used in mathematical calculations
VARCHAR NVARCHAR to INT DECIMAL DATETIME DATE TIME	straight conversion to proper data type as long as the string field only has numbers and structure is correct (for example, text with value of `2019/03/11` to DATE); no conversion if the string has letters or special characters
VARCHAR to NVARCHAR	straight conversion; no data loss
NVARCHAR to VARCHAR	straight conversion if string is encoded as ACIII or UTF-8; possible data loss if string is encoded as Unicode or no conversion at all

3.02. Refer to https://technet.microsoft.com/en-us/library/ms187912.aspx for information on approximate numeric data types -- FLOAT and REAL. If you are considering taking the certification, you should know the concept below and why Microsoft recommends not using approximate numeric data types.

``The float and real data types are known as approximate data types. The behavior of float and real follows the IEEE 754 specification on approximate numeric data types. Approximate numeric data types do not store the exact values specified for many numbers; they store an extremely close approximation of the value. For many applications, the tiny difference between the specified value and the stored approximation is not noticeable. At times, though, the difference becomes noticeable. Because of the approximate nature of the float and real data types, do not

```
                    use these data types when exact numeric behavior is
                    required, such as in financial applications, in operations
                    involving rounding, or in equality checks.  Instead, use
                    the integer, decimal, money, or smallmoney data types.
                    Avoid using float or real columns in WHERE clause search
                    conditions, especially the = and <> operators.  It is best
                    to limit float and real columns to > or < comparisons.  The
                    IEEE 754 specification provides four rounding modes: round
                    to nearest, round up, round down, and round to zero.
                    Microsoft SQL Server uses round up.  All are accurate to
                    the guaranteed precision but can result in slightly
                    different floating-point values.  Because the binary
                    representation of a floating-point number may use one of
                    many legal rounding schemes, it is impossible to reliably
                    quantify a floating-point value.``
                    https://technet.microsoft.com/en-us/library/ms187912.aspx

          Note that FLOAT is commonly used in other relational database
          management systems (RDBMS) like Oracle (http://oracle.com/) and in
          most programming languages including those distributed by Microsoft.

4. As we start, we keep in mind that the most basic structure of a `SELECT`
   statement (https://techonthenet.com/sql/select.php) is the following.

                    SELECT field1, field2...
                    FROM   table1

   From the previous structure, you can add clauses in the following order.
   If you organize the clauses any other order, the query will not work.

                    SELECT table1.field1,       -- 1. calling columns/fields
                      table1.field2,            --    (data)
                      ...
                      table2.field1,
                      table2.field2,
                      ...
                      table3.field1,
                      table3.field2,
                      ...

                    FROM table1                 -- 2. where to find data
                                                --    (tables/views)
                    INNER|LEFT|RIGHT JOIN table2
                      ON table1.shared_field1 = table2.shared_field1
                      AND table1.shared_field2 = table2.shared_field2
                      ...
                    INNER|LEFT|RIGHT JOIN table3
                      ON table1.shared_field1 = table3.shared_field1
                      AND table1.shared_field2 = table3.shared_field2
                      ...

                    WHERE condition1            -- 3. filtering output, what
```

```
728                       AND|OR condition2         --    rows/records you want to
729                       AND|OR condition3         --    retrieve
730                       ...
731
732                     GROUP BY table1.field1,     -- 4. grouping fields not in an
733                       table1.field2,            --    aggregate function
734                       ...
735                       table2.field1,
736                       table2.field2,
737                       ...
738                       table3.field1,
739                       table3.field2,
740                       ...
741
742                     ORDER BY                    -- 5. organizing rows/records
743                       table1.field1 ASC|DESC,   --    (output) in ascending
744                       table1.field2 ASC|DESC,   --    (`ASC`) or descending
745                       ...                       --    (`DESC`) order
746                       table2.field1 ASC|DESC,
747                       table2.field2 ASC|DESC,
748                       ...
749                       table3.field1 ASC|DESC,
750                       table3.field2 ASC|DESC,
751                       ...
752
753     4.01. In the example below, we retrieve all (`*`) columns from table
754           `AP1.Vendors`.
755  ************************************************************************ */
756
757  SELECT *
758  FROM AP1.Vendors;                              -- retrieves all values from
759                                                 -- table `AP1.Vendors`
760
761
762  /* ************************************************************************
763     4.02. The only time you can use `SELECT` without `FROM` is when you want
764           the machine to return a value, similar to `PRINT`.
765  ************************************************************************ */
766
767  SELECT 9 * 8;                                  -- returns integer 72 (a
768                                                 -- mathematical equation)
769
770  SELECT 'Hello there';                          -- returns string `Hello there`
771                                                 -- (a simple string)
772
773
774  /* ************************************************************************
775     4.02. As you can see in the examples above, we are not retrieving data from
776           any table.  You can get the same results using `PRINT`.
777  ************************************************************************ */
778
779  PRINT 9 * 8;                                   -- prints integer 72 (a
```

```
780                                                 -- mathematical equation)
781
782 PRINT 'Hello there';                             -- prints string `Hello there`
783                                                 -- (a simple string)
784
785
786 /* ************************************************************************
787  5. We have covered built-in functions that affect strings.
788
789     CONCAT()        allows you to concatenate strings together
790                     https://techonthenet.com/sql_server/functions/concat.php
791
792     `+`             allows you to concatenate 2 or more strings together
793                     https://techonthenet.com/sql_server/functions/concat2.php
794
795     LEFT()          allows you to extract a substring from a string, starting
796                     from the left-most character
797                     https://techonthenet.com/sql_server/functions/left.php
798
799     LEN()           returns the length of the specified string... does not
800                     include trailing space characters at the end the string
801                     when calculating the length
802                     https://techonthenet.com/sql_server/functions/len.php
803
804     LTRIM()         removes all space characters from the left-hand side of a
805                     string
806                     https://techonthenet.com/sql_server/functions/ltrim.php
807
808     LOWER()         converts all letters in the specified string to lowercase
809                     https://techonthenet.com/sql_server/functions/lower.php
810
811     REPLACE()       replaces a sequence of characters in a string with another
812                     set of characters, not case-sensitive
813                     https://techonthenet.com/sql_server/functions/replace.php
814
815     RIGHT()         allows you to extract a substring from a string, starting
816                     from the right-most character
817                     https://techonthenet.com/sql_server/functions/right.php
818
819     RTRIM()         removes all space characters from the right-hand side of a
820                     string
821                     https://techonthenet.com/sql_server/functions/rtrim.php
822
823     SUBSTRING()     allows you to extract a substring from a string
824                     https://techonthenet.com/sql_server/functions/substring.php
825
826     UPPER()         converts all letters in the specified string to uppercase
827                     https://techonthenet.com/sql_server/functions/upper.php
828
829     Now we will see functions used with numeric values.
830
831     AVG()           returns the average value of an expression
```

```
832                     https://techonthenet.com/sql_server/functions/avg.php
833
834     CEILING()       returns the smallest integer value that is greater than or
835                     equal to a number
836                     https://techonthenet.com/sql_server/functions/ceiling.php
837
838     COUNT()         returns the count of an expression
839                     https://techonthenet.com/sql_server/functions/count.php
840
841     FLOOR()         returns the largest integer value that is equal to or less
842                     than a number
843                     https://techonthenet.com/sql_server/functions/floor.php
844
845     MAX()           returns the maximum value of an expression
846                     https://techonthenet.com/sql_server/functions/max.php
847
848     MIN()           returns the minimum value of an expression
849                     https://techonthenet.com/sql_server/functions/min.php
850
851     RAND()          returns a random number or a random number within a range
852                     https://techonthenet.com/sql_server/functions/rand.php
853
854     ROUND()         returns a number rounded to a certain number of decimal
855                     places
856                     https://techonthenet.com/sql_server/functions/round.php
857
858     SUM()           returns the summed value of an expression
859                     https://techonthenet.com/sql_server/functions/sum.php
860
861   6. In the examples below, we use each one of the numeric functions with the
862      answer for each on the comment on the right.
863   ********************************************************************** */
864
865  SELECT SUM(InvoiceTotal) AS InvoiceTotalSUM,     -- returns 214290.51
866    AVG(InvoiceTotal) AS InvoiceTotalAVG,          -- returns 1879.7413
867    COUNT(InvoiceTotal) AS InvoiceTotalCOUNT,      -- returns 114
868    ROUND(InvoiceTotal, 1) AS InvoiceTotalROUND,   -- returns 3813.30
869                                                   --         40.20 ...
870    FLOOR(InvoiceTotal) AS InvoiceTotalFLOOR,      -- returns 3813.00
871                                                   --         40.00 ...
872    CEILING(InvoiceTotal) AS InvoiceTotalCEILING, -- returns 3814.00
873                                                   --         41.00 ...
874    MAX(InvoiceTotal) AS InvoiceTotalMAX,          -- returns 37966.19
875    MIN(InvoiceTotal) AS InvoiceTotalMIN,          -- returns 6.00
876    RAND(InvoiceTotal) AS InvoiceTotalRAND,        -- returns 0.713591993212924
877                                                   --         0.713610626184182...
878    FORMAT(InvoiceTotal, 'c', 'en-us')             -- `c` for currency with
879      AS InvoiceTotal,                             -- culture `en-us` (English US)
880                                                   -- returns $3,813.33
881                                                   --         $40.20 ...
882    FORMAT(InvoiceDueDate, 'd', 'en-us')           -- `d` (lower case) for short
883      AS InvoiceDueDate,                           -- date returning no leading
```

```
884                                             -- zeros with culture `en-us`
885                                             -- (English US);
886                                             -- returns 1/8/2012
887                                             --         1/10/2012 ...
888   FORMAT(InvoiceDueDate, 'D', 'en-us')      -- `D` (upper case) for long
889     AS InvoiceDueDate,                      -- date returning full day of
890                                             -- the week, full month, no
891                                             -- leading zeros with culture
892                                             -- `en-us` (English US);
893                                             -- returns
894                                             --    Sunday, January 8, 2012
895                                             --    Tuesday, January 10, 2012
896                                             --    ...
897   FORMAT(InvoiceDueDate, 'MM/dd/yyyy', 'en-us') -- custom date using format
898     AS InvoiceDueDate                       -- `MM/dd/yyyy` which overrides
899                                             -- culture `en-us` (English
900                                             -- US); returns 01/08/2012
901                                             --              01/10/2012 ...
902 FROM AP1.Invoices
903 GROUP BY InvoiceTotal,
904   AP1.Invoices.InvoiceDueDate
905
906
907 /* ************************************************************************
908     6.01. When using an aggregate function, we must use `GROUP BY` and list all
909           columns not in affected by any aggregate function.
910
911           In the example below, we retrieve `VendorState` plus the count of
912           column `VendorState` for each `VendorState` (`COUNT(VendorState)`).
913
914           We can use `DISTINCT` to make sure that duplicate values (rows) are
915           not included in the output of a query.
916
917           We can use `ORDER BY` to organize output by a specific column or list
918           of columns.
919
920           The default option for `ORDER BY` is ascending (`ASC`), which can be
921           omitted (1, 2, 3... a, b, c...).
922
923           The opposite option for `ORDER BY` is descending (`DESC`), which must
924           be used if needed (...3, 2, 1 ...c, b, a).
925  ************************************************************************ */
926
927 SELECT DISTINCT                            -- 1. to avoid duplicates
928   VendorState,                             -- 2. column not in aggregate
929                                            --    function
930   COUNT(VendorState)                       -- 3. column in aggregate
931                                            --    function (calculation)
932 FROM AP1.Vendors                           -- 4. from table `AP1.Vendors`
933 GROUP BY VendorState                       -- 5. must use `GROUP BY` when
934                                            --    using any aggregate
935                                            --    function, listing all
```

```
936                                                 --    columns not in the
937                                                 --    aggregate function
938 ORDER BY VendorState ASC;                       -- 6. organizing results by
939                                                 --    column `VendorState` in
940                                                 --    ascending order
941
942
943 /* ************************************************************************
944     6.02. In the example below, we retrieve `VendorID` plus the sum of column
945           `PaymentTotal` for each `VendorID` (`SUM(PaymentTotal)`).
946  ************************************************************************ */
947
948 SELECT DISTINCT                                 -- 1. to avoid duplicates
949   VendorID,                                     -- 2. column not in aggregate
950                                                 --    function
951   SUM(PaymentTotal)                             -- 3. column in aggregate
952                                                 --    function (calculation)
953 FROM AP1.Invoices                               -- 4. from table `AP1.Invoices`
954 GROUP BY VendorID                               -- 5. must use `GROUP BY` when
955                                                 --    using any aggregate
956                                                 --    function, listing all
957                                                 --    columns not in the
958                                                 --    aggregate function
959 ORDER BY VendorID DESC;                         -- 6. organizing results by
960                                                 --    column `VendorID` in
961                                                 --    descending order
962
963
964 /* ************************************************************************
965  7. In the example below, the query returns all values from the `AP1.Vendors`
966     table with all related values from table `AP1.Invoices`,
967     `AP1.InvoiceLineItems` and `AP1.Terms`.
968
969     7.01. The relation between related tables `AP1.Invoices`,
970           `AP1.InvoiceLineItems` and `AP1.Terms` is `INNER JOIN` since the
971           value (row ID) of one table in referenced in another.
972
973     7.02. Dollar amounts are formatted as `c` (currency) with culture `en-us`
974           (English-United States).  Dates are formatted as `MM/dd/yyyy` (two
975           digits for month and day, four digits for year) and culture `en-us`
976           (English-United States).  Refer to
977           https://msdn.microsoft.com/en-us/library/hh213506.aspx for more
978           information.  Note that formatting a numeric value changes it to an
979           alpha-numeric value -- change in data type.
980
981     7.03. To include the average value of `InvoiceTotal` of all records from
982           table `AP1.Invoices`, we use a sub-query (also referred to as nested
983           query, http://tutorialspoint.com/sql/sql-sub-queries.htm).  We use
984           alias `AvgInvoiceTotal` to refer to this new column.
985
986                     (
987                       SELECT FORMAT(AVG(AP1.Invoices.InvoiceTotal),'c','en-us')
```

```
                    FROM AP1.Invoices
                )
                  AS AvgInvoiceTotal

        There are various values for culture (one per language and country
        combination).  The following are just a few, probably the most common
        in American businesses.  Refer to
        http://sql-server-helper.com/sql-server-2012/format-string-function-culture.aspx
        for a more detailed list of cultures.

                +-----------+--------------+--------------+-------------+
                | CULTURE   | LANGUAGE     | COUNTRY      | RESULT      |
                +-----------+--------------+--------------+-------------+
                | en-us     | English      | USA          | dollar      |
                +-----------+--------------+--------------+-------------+
                | en-gb     | English      | Great Britain | pound      |
                +-----------+--------------+--------------+-------------+
                | de-de     | German       | Germany      | euro        |
                +-----------+--------------+--------------+-------------+
                | zh-cn     | Simplified   | China        | yuan        |
                |           | Chinese      |              |             |
                +-----------+--------------+--------------+-------------+
                | jp-jp     | Japanese     | Japan        | yen         |
                +-----------+--------------+--------------+-------------+

        Refer to https://www.iso.org/iso-4217-currency-codes.html for more
        information on currency codes (ISO 4217).

        When formatting DATETIME fields, you can use any of the formats below
        and the culture (`en-us`).  The default format in data type DATETIME
        is `yyyy-MM-dd hh:mm:ss.nnnnnnn`.  Refer to
        https://docs.microsoft.com/en-us/sql/t-sql/functions/datename-transact-sql
        for more information about dates.

                +-----------+--------------+-----------------------------+
                | OPTION    | OUTPUT       | FORMAT                      |
                +-----------+--------------+-----------------------------+
                | c         | currency     | `c`, `en-us`                |
                |           | depending on |                             |
                |           | culture (`$`) |                            |
                +-----------+--------------+-----------------------------+
                | d         | day without  | `d`, `en-us`                |
                |           | leading zero, |                            |
                |           | day without  |                             |
                |           | leading zero |                             |
                |           | and complete |                             |
                |           | year         |                             |
                |           | (04/17/2023) |                             |
                +-----------+--------------+-----------------------------+
                | D         | whole day of | `D`, `en-us`                |
```

```
1038                    |           | the week,     |                             |
1039                    |           | first letter  |                             |
1040                    |           | capitalized;  |                             |
1041                    |           | whole month,  |                             |
1042                    |           | first letter  |                             |
1043                    |           | capitalized;  |                             |
1044                    |           | day without   |                             |
1045                    |           | leading zero  |                             |
1046                    |           | and complete  |                             |
1047                    |           | year (Monday, |                             |
1048                    |           | April 17,     |                             |
1049                    |           | 2023)         |                             |
1050                    +-----------+---------------+-----------------------------+
1051
1052                    +-----------+---------------+-----------------------------+
1053                    | DATEPART  | OUTPUT        | FORMAT                      |
1054                    +-----------+---------------+-----------------------------+
1055                    | dw        | whole day of  | `dw MMMM dd, yyyy`          |
1056                    |           | the week,     | `dw MMMM d, yyyy`           |
1057                    |           | first letter  | `dw MMMM dd, yy`            |
1058                    |           | capitalized   | `dw MMMM d, yy`             |
1059                    |           | (Monday)      |                             |
1060                    +-----------+---------------+-----------------------------+
1061                    | MMMM      | whole month,  | `MMMM dd, yyyy`             |
1062                    |           | first letter  | `MMMM d, yyyy`              |
1063                    |           | capitalized   | `MMMM dd, yy`               |
1064                    |           | (April)       | `MMMM d, yy`                |
1065                    +-----------+---------------+-----------------------------+
1066                    | MMM       | month in      | `MMM dd, yyyy`              |
1067                    |           | abbreviation, | `MMM d, yyyy`               |
1068                    |           | first letter  | `MMM dd, yy`                |
1069                    |           | capitalized   | `MMM d, yy`                 |
1070                    |           | (Apr)         | `dd-MMM-yy`(default Oracle) |
1071                    |           |               | `d-MMM-yy` (default Oracle) |
1072                    +-----------+---------------+-----------------------------+
1073                    | MM        | month number  | `MM/dd/yyyy`                |
1074                    |           | with leading  | `MM/d/yyyy`                 |
1075                    |           | zero (04)     | `MM/dd/yy`                  |
1076                    |           |               | `MM/d/yy`                   |
1077                    +-----------+---------------+-----------------------------+
1078                    | M         | month number  | `M/dd/yyyy`                 |
1079                    |           | without       | `M/d/yyyy`                  |
1080                    |           | leading zero  | `M/dd/yy`                   |
1081                    |           | (4)           | `M/d/yy`                    |
1082                    +-----------+---------------+-----------------------------+
1083                    | dddd      | day of week   | `dddd, MMM d, yyyy`         |
1084                    |           | (Monday)      | `dddd, MMMM d, yyyy`        |
1085                    +-----------+---------------+-----------------------------+
1086                    | ddd       | day of week   | `ddd, MMM d, yyyy`          |
1087                    |           | abbreviation  } `ddd, MMMM d, yyyy`         |
1088                    |           | (Mon)         |                             |
1089                    +-----------+---------------+-----------------------------+
```

```
1090                    | dd        | day with      | `MM/dd/yyyy`                |
1091                    |           | leading zero  | `M/dd/yyyy`                 |
1092                    |           | (17)          | `MM/dd/yy`                  |
1093                    |           |               | `M/dd/yy`                   |
1094                    +-----------+---------------+-----------------------------+
1095                    | d         | day without   | `MM/d/yyyy`                 |
1096                    |           | leading zero  | `M/d/yyyy`                  |
1097                    |           | (17)          | `MM/d/yy`                   |
1098                    |           |               | `M/d/yy`                    |
1099                    +-----------+---------------+-----------------------------+
1100                    | yy        | last two      | `M/dd/yy`                   |
1101                    |           | digits of year| `M/d/yy`                    |
1102                    |           | (23)          | `MM/d/yy`                   |
1103                    |           |               | `M/d/yy`                    |
1104                    +-----------+---------------+-----------------------------+
1105                    | yyyy      | complete year | `M/dd/yyyy`                 |
1106                    |           | (2023)        | `M/d/yyyy`                  |
1107                    |           |               | `MM/d/yyyy`                 |
1108                    |           |               | `M/d/yyyy`                  |
1109                    +-----------+---------------+-----------------------------+
1110                    | HH        | 24-hour,      | `HH:mm:ss`                  |
1111                    |           | military time |                             |
1112                    |           | with leading  |                             |
1113                    |           | zero (20)     |                             |
1114                    +-----------+---------------+-----------------------------+
1115                    | H         | 24-hour,      | `H:mm:ss`                   |
1116                    |           | military time |                             |
1117                    |           | without       |                             |
1118                    |           | leading zero  |                             |
1119                    |           | (20)          |                             |
1120                    +-----------+---------------+-----------------------------+
1121                    | hh        | 12-hour       | `hh:mm:ss`                  |
1122                    |           | (AM/PM), with |                             |
1123                    |           | leading zero  |                             |
1124                    |           | (08 PM)       |                             |
1125                    +-----------+---------------+-----------------------------+
1126                    | h         | 12-hour       | `h:mm:ss`                   |
1127                    |           | (AM/PM),      |                             |
1128                    |           | without       |                             |
1129                    |           | leading zero  |                             |
1130                    |           | (8 PM)        |                             |
1131                    +-----------+---------------+-----------------------------+
1132                    | mm        | minutes (13)  | `HH:mm:ss`                  |
1133                    +-----------+---------------+ `H:mm:ss`                   |
1134                    | ss        | seconds (58)  | `hh:mm:ss`                  |
1135                    |           |               | `h:mm:ss`                   |
1136                    +-----------+---------------+-----------------------------+
1137                    | nnnnnnn   | six decimal   | `HH:mm:ss.nnnnnnn`          |
1138                    |           | spaces,       | `H:mm:ss.nnnnnnn`           |
1139                    |           | fractions of  | `hh:mm:ss.nnnnnnn`          |
1140                    |           | a second      | `h:mm:ss.nnnnnnn`           |
1141                    +-----------+---------------+-----------------------------+
```

```
1142
1143          Although we are using aggregate function `AVG()`, we do not need to
1144          use `GROUP BY` since the function is inside the sub-query.
1145
1146          Go to https://docs.microsoft.com/en-us/sql/t-sql/functions/format-
               transact-sql
1147          for more information on `FORMAT()`.
1148  ********************************************************************** */
1149
1150 SELECT DISTINCT AP1.Vendors.VendorID,
1151   AP1.Vendors.VendorName,
1152   CONCAT (                                    -- 1. concatenating
1153     AP1.Vendors.VendorAddress1,               --    `VendorAddress1`, an
1154     ' ',                                      --    empty space and
1155     AP1.Vendors.VendorAddress2                --    `VendorAddress2`
1156     ) AS VendorAddress,                       --    as `VendorAddress`
1157   AP1.Vendors.VendorCity,
1158   AP1.Vendors.VendorState,
1159   CONCAT (                                    -- 2. concatenating
1160     AP1.Vendors.VendorZipCode,                --    `VendorZipCode` and a
1161     '-0000'                                   --    dummy Plus4 as
1162     ) AS VendorZipCode,                       --    VendorZipCode
1163
1164
1165   CASE
1166     WHEN AP1.Vendors.VendorPhone <> ''        -- 3. checking that
1167       OR AP1.Vendors.VendorPhone <> ' '       --    `VendorPhone` is not an
1168       OR AP1.Vendors.VendorPhone IS NOT NULL  --    empty string or not a
1169                                               --    space or not `IS NOT
1170                                               --    NULL` (in other
1171                                               --    words, not no-value; must
1172                                               --    have a value) using
1173     THEN CONCAT (                             -- 4. concatenating an opening
1174         '(',                                  --    parenthesis, the first 3
1175         LEFT(AP1.Vendors.VendorPhone, 3),     --    characters of
1176                                               --    `VendorPhone` (area
1177                                               --    code), corresponding
1178         ') ',                                 --    closing parenthesis with
1179         SUBSTRING(AP1.Vendors.VendorPhone, 4, 3),--   a space, the substring
1180                                               --    from `VendorPhone`
1181                                               --    starting with character 4
1182                                               --    taking 3 characters
1183                                               --    (branch exchange), a
1184         '-',                                  --    hyphen and the 4 four
1185         RIGHT(AP1.Vendors.VendorPhone, 4)     --    characters of
1186                                               --    `VendorPhone`
1187                                               --    (subscriber number) using
1188         )                                     --    alias `VendorPhone`
1189       ELSE ''
1190     END
1191       AS VendorPhone,
1192   LTRIM(RTRIM(                                -- 5. trimming the output of
```

```
1193                                                --    the concatenation of
1194        CONCAT(AP1.Vendors.VendorContactLName,  --    `VendorContactLName`, a
1195        ', ',                                   --    comma with a space and
1196        AP1.Vendors.VendorContactFName))        --    `VendorContactFName`
1197    ) AS VendorContactName,                     --    using alias
1198                                                --    `VendorContactName`
1199    AP1.Vendors.DefaultAccountNo,
1200    AP1.Invoices.InvoiceID,
1201    AP1.Invoices.InvoiceNumber,
1202    FORMAT(AP1.Invoices.InvoiceDate,            -- 5. formatting column as
1203      'MM/dd/yyyy', 'en-us')                    --    `MM/dd/yyyy` (date) with
1204                                                --    culture `en-us` as
1205    AS InvoiceDate,                             --    `InvoiceDate`
1206    FORMAT(AP1.Invoices.InvoiceTotal,           -- 7. formatting column as
1207      'c', 'en-us')                             --    `c` (currency) with
1208                                                --    culture `en-us` with
1209    AS InvoiceTotal,                            --    alias `InvoiceTotal`
1210    (
1211      SELECT                                    -- 8. embedded query calling
1212        FORMAT(AVG(AP1.Invoices.InvoiceTotal),  --    `AVG(InvoiceTotal)`
1213          'c', 'en-us')                         --    formatted as `c`
1214                                                --    (currency) with culture
1215                                                --    `en-us`
1216      FROM AP1.Invoices                         --    from all values in table
1217                                                --    `AP1.Invoices` as
1218    ) AS AvgInvoiceTotal,                       --    `AvgInvoiceTotal`
1219    FORMAT(AP1.Invoices.PaymentTotal,           -- 9. formatting column as `c`
1220      'c', 'en-us')                             --    (currency) with culture
1221    AS PaymentTotal,                            --    `en-us` as `PaymentTotal`
1222    FORMAT(AP1.Invoices.CreditTotal,            -- 10. formatting column as `c`
1223      'c', 'en-us')                             --     (currency) with culture
1224    AS CreditTotal,                             --     `en-us` as `CreditTotal`
1225    FORMAT(AP1.Invoices.InvoiceDueDate,         -- 11. formatting column as
1226      'MM/dd/yyyy', 'en-us')                    --     `MM/dd/yyyy` (date) with
1227                                                --     culture `en-us` as
1228    AS InvoiceDueDate,                          --     `InvoiceDueDate`
1229    FORMAT(AP1.Invoices.PaymentDate,            -- 12. formatting column as
1230      'MM/dd/yyyy', 'en-us')                    --     `MM/dd/yyyy` (date) with
1231                                                --     culture `en-us` as
1232    AS PaymentDate,                             --     `PaymentDate`
1233    AP1.InvoiceLineItems.InvoiceSequence,
1234    AP1.InvoiceLineItems.AccountNo,
1235    FORMAT(AP1.InvoiceLineItems.InvoiceLineItemAmount,
1236                                                -- 13. formatting column as
1237      'c', 'en-us')                             --    `c` (currency) with
1238                                                --    culture `en-us` as
1239    AS InvoiceLineItemAmount,                   --    `InvoiceLineItemAmount`
1240    AP1.InvoiceLineItems.InvoiceLineItemDescription,
1241    AP1.Terms.TermsDescription,
1242    AP1.Terms.TermsDueDays
1243  FROM AP1.InvoiceLineItems                     -- 14. from
1244                                                --     `AP1.InvoiceLineItems`
```

```
1245  INNER JOIN AP1.Invoices                          --     using `INNER JOIN` to
1246                                                   --     to connect to
1247                                                   --     `AP1.Invoices` to get
1248                                                   --     all shared values from
1249    ON AP1.InvoiceLineItems.InvoiceID = AP1.Invoices.InvoiceID
1250                                                   --     `AP1.InvoiceLineItems`
1251                                                   --     and `AP1.Invoices`
1252  INNER JOIN AP1.Terms                             --     using `INNER JOIN` to
1253                                                   --     connect to `AP1.Terms`
1254                                                   --     to get all shared values
1255                                                   --     from
1256    ON AP1.Invoices.TermsID = AP1.Terms.TermsID    --     (`AP1.InvoiceLineItems`
1257                                                   --     and `AP1.Invoices`) and
1258                                                   --     `AP1.Terms` using
1259  RIGHT JOIN AP1.Vendors                           --     `RIGHT JOIN` to connect
1260                                                   --     to `AP1.Vendors` to get
1261                                                   --     values from
1262                                                   --     `AP1.Vendors` and
1263                                                   --     related data from
1264    ON AP1.Invoices.VendorID=AP1.Vendors.VendorID  --     (`AP1.InvoiceLineItems`
1265                                                   --     and `AP1.Invoices` and
1266                                                   --     `AP1.Terms`)
1267  ORDER BY                                         -- 15. ordering results by
1268    AP1.Vendors.VendorName,                        --     `VendorName`first and
1269    AP1.Invoices.InvoiceID;                        --     then by `InvoiceID`
1270
1271
1272  /* ***************************************************************************
1273            As with previous example, we can use an alias for each table, which
1274            in this case, allows us to present neater code.
1275
1276                      `il` for `AP1.InvoiceLineItems`
1277                      `i`  for `AP1.Invoices`
1278                      `t`  for `AP1.Terms`
1279                      `v`  for `AP1.Vendors`
1280   ************************************************************************** */
1281
1282  SELECT DISTINCT v.VendorID,
1283    v.VendorName,
1284    CONCAT (
1285      v.VendorAddress1,
1286      ' ',
1287      v.VendorAddress2
1288      ) AS VendorAddress,
1289    v.VendorCity,
1290    v.VendorState,
1291    CONCAT (
1292      v.VendorZipCode,
1293      '-0000'
1294      ) AS VendorZipCode,
1295    CASE
1296      WHEN v.VendorPhone <> ''
```

```
1297        OR v.VendorPhone <> ' '
1298        OR v.VendorPhone IS NOT NULL
1299        THEN CONCAT (
1300            '(',
1301            LEFT(v.VendorPhone, 3),
1302            ') ',
1303            SUBSTRING(v.VendorPhone, 4, 3),
1304            RIGHT(v.VendorPhone, 4)
1305            )
1306      ELSE ''
1307      END AS VendorPhone,
1308    LTRIM(RTRIM(CONCAT (
1309          v.VendorContactLName,
1310          ', ',
1311          v.VendorContactFName
1312          ))) AS VendorContactName,
1313    v.DefaultAccountNo,
1314    i.InvoiceID,
1315    i.InvoiceNumber,
1316    FORMAT(i.InvoiceDate, 'MM/dd/yyyy', 'en-us') AS InvoiceDate,
1317    FORMAT(i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1318    (
1319      SELECT FORMAT(AVG(i.InvoiceTotal), 'c', 'en-us')
1320      FROM AP1.Invoices AS i
1321      ) AS AvgInvoiceTotal,
1322    FORMAT(i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1323    FORMAT(i.CreditTotal, 'c', 'en-us') AS CreditTotal,
1324    FORMAT(i.InvoiceDueDate, 'MM/dd/yyyy', 'en-us') AS InvoiceDueDate,
1325    FORMAT(i.PaymentDate, 'MM/dd/yyyy', 'en-us') AS PaymentDate,
1326    il.InvoiceSequence,
1327    il.AccountNo,
1328    FORMAT(il.InvoiceLineItemAmount, 'c', 'en-us')
1329      AS InvoiceLineItemAmount,
1330    il.InvoiceLineItemDescription,
1331    t.TermsDescription,
1332    t.TermsDueDays
1333  FROM AP1.InvoiceLineItems AS il
1334  INNER JOIN AP1.Invoices AS i
1335    ON il.InvoiceID = i.InvoiceID
1336  INNER JOIN AP1.Terms AS t
1337    ON i.TermsID = t.TermsID
1338  RIGHT JOIN AP1.Vendors AS v
1339    ON i.VendorID = v.VendorID
1340  ORDER BY v.VendorName,
1341    i.InvoiceID;
1342
1343
1344  /* ***************************************************************************
1345   8. To get the difference between two dates, we use `DATEDIFF()`, which
1346      ``returns the difference between two date values, based on the interval
1347      specified`` (https://techonthenet.com/sql_server/functions/datediff.php).
1348
```

```
1349      We also call functions `DAY()`
1350      (https://techonthenet.com/sql_server/functions/day.php), `MONTH()`
1351      (https://techonthenet.com/sql_server/functions/month.php) and `YEAR()`
1352      (https://techonthenet.com/sql_server/functions/year.php).
1353
1354      8.01. In the example below, we use `01/01/2017` as the starting date and
1355            `04/17/2023` as the end date.
1356   ************************************************************************ */
1357
1358  SELECT DATEDIFF(DAY,'01/01/2017','04/17/2023') AS DatediffDays, -- 2,297 days
1359    DATEDIFF(MONTH,'01/01/2017','04/17/2023') AS DatediffMonths,  --    75 months
1360    DATEDIFF(YEAR,'01/01/2017','04/17/2023') AS DatediffYears;    --     6 years
1361
1362
1363  /* ************************************************************************
1364      8.02. Instead of hard-coding today's date, we can use function `GETDATE()`
1365            to retrieve the local system datetime.
1366   ************************************************************************ */
1367
1368  SELECT DATEDIFF(DAY, '01/01/2017', GETDATE()) AS DatediffDays,  -- 2,297 days
1369    DATEDIFF(MONTH, '01/01/2017', GETDATE()) AS DatediffMonths,   --    75 months
1370    DATEDIFF(YEAR, '01/01/2017', GETDATE()) AS DatediffYears;     --     6 years
1371
1372
1373  /* ************************************************************************
1374   https://folvera.commons.gc.cuny.edu/?p=1223
1375   ************************************************************************ */
```