```
/* ************************************************************************
          INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
                    SF25SQL6172025, 2025/11/17 - 2025/12/17
                  https://folvera.commons.gc.cuny.edu/?cat=35
 ************************************************************************

  SESSION #5 (2025/12/01): 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;


  /* ***************************************************************************
            As with previous example, we can use an alias for each table, which
            in this case, allows us to present neater code.

                      `v` for `AP1.Vendors`
                      `c` for `AP1.ContactUpdates`
   *************************************************************************** */

  SELECT v.VendorID,
    v.VendorName,
    v.VendorAddress1,
    v.VendorAddress2,
    v.VendorCity,
    v.VendorState,
    v.VendorZipCode,
    v.VendorPhone,
    v.VendorContactFName +
      ' ' +
      v.VendorContactLName AS ContactName,
    LEFT(v.VendorContactFName, 1) +
      v.VendorContactLName + '@' +
      REPLACE(
        REPLACE(
          REPLACE(
            REPLACE(
              REPLACE(
                REPLACE(v.VendorName, ' ', '')
              , '.', '')
            , ',', '')
          , '/', '')
        , '&', '')
      , '''', '')
      + '.foo' AS VendorContactEmail,
    v.DefaultTermsID,
```

```
    v.DefaultAccountNo,
    -- c.VendorID AS Expr1,
    c.FirstName +
    ' '+
    c.LastName AS ContactName,
    LEFT(c.FirstName, 1) +
      c.LastName +
      '@' +
      REPLACE(
        REPLACE(
          REPLACE(
            REPLACE(
              REPLACE(
                REPLACE(v.VendorName, ' ', '')
              , '.', '')
            , ',', '')
          , '/', '')
        , '&', '')
      , '''', '')
      + '.foo' AS VendorContactEmail
  FROM AP1.Vendors AS v
  LEFT JOIN AP1.ContactUpdates AS c
    ON v.VendorID = c.VendorID;


  /* ***************************************************************************
            Instead of using a plus sign (`+`), we can use `CONCAT()` since
            adding a value and NULL returns NULL.  In other words, we lose data,
            which would be logical error (not syntax error).
   *************************************************************************** */

  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,
    CONCAT (                                          -- 1. concatenation of
      AP1.Vendors.VendorContactFName,                 --    `FirstName`, a single
      ' ',                                            --    space and `LastName` with
      AP1.Vendors.VendorContactLName                  --    alias `ContactName`
      ) AS ContactName,
    CONCAT (                                          -- 2. concatenation of
      LEFT(AP1.Vendors.VendorContactFName, 1),        --    `VendorContactFName`,
      AP1.Vendors.VendorContactLName,                 --    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,
    CONCAT (                                        -- 3. concatenation of
      AP1.ContactUpdates.FirstName,                 --    `VendorContactFName`, a
      ' ',                                          --    single space and
      AP1.ContactUpdates.LastName                   --    `LastName` with alias
      ) AS ContactName,                             --    `VendorContactLName`
    CONCAT (                                        -- 4. concatenation of
      LEFT(AP1.ContactUpdates.FirstName, 1),        --    `VendorContactFName`,
      AP1.ContactUpdates.LastName,                  --    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;


  /* ****************************************************************************
            Using `CONCAT()` also returns a logical error (not syntax error)
            since the concatenation to make the second email returns values like
            `@USPostalService.foo` since there is no corresponding `FirstName`
            and `LastName`.  We could use a CASE clause.
   **************************************************************************** */

  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,
    CONCAT (                                        -- 1. concatenation of
      AP1.Vendors.VendorContactFName,               --    `FirstName`, a single
      ' ',                                          --    space and `LastName` with
      AP1.Vendors.VendorContactLName                --    alias `ContactName`
      ) AS ContactName,
    CONCAT (                                        -- 2. concatenation of
      LEFT(AP1.Vendors.VendorContactFName, 1),      --    `VendorContactFName`,
      AP1.Vendors.VendorContactLName,               --    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
```

```
277                 REPLACE(AP1.Vendors.VendorName, ' ', '')
278                                                     --    2.06. generating value #1
279                   , '.', '')                        --    2.07. generating value #2
280                 , ',', '')                          --    2.08. generating value #3
281               , '/', '')                            --    2.09. generating value #4
282             , '&', '')                              --    2.10. generating value #5
283           , '''', '')                               --    2.11. generating value #6
284                                                     --          where `''`
285                                                     --          represents a single
286                                                     --          quote (`'`) used an
287                                                     --          escape character
288       , '.foo'                                      --    and `.foo` to complete
289       ) AS VendorContactEmail,                      --    email with alias
290                                                     --    `VendorContactEmail`
291     AP1.Vendors.DefaultTermsID,
292     AP1.Vendors.DefaultAccountNo,
293     -- AP1.ContactUpdates.VendorID AS Expr1,
294     CONCAT (                                        -- 3. concatenation of
295       AP1.ContactUpdates.FirstName,                 --    `VendorContactFName`, a
296       ' ',                                          --    single space and
297       AP1.ContactUpdates.LastName                   --    `LastName` with alias
298       ) AS ContactName,                             --    `VendorContactLName`
299
300     CASE
301       WHEN (AP1.ContactUpdates.FirstName <> ''      -- 4. checking if
302         OR AP1.ContactUpdates.FirstName <> ' '      --    `ContactUpdates` is not
303         OR AP1.ContactUpdates.FirstName IS NOT NULL)--  an empty string or not
304                                                     --    equal to a space or
305                                                     --    `IS NOT NULL` (in other
306                                                     --    words, not no-value; must
307                                                     --    have a value) using
308                                                     --    parenthesis to make a
309                                                     --    block of three conditions
310         AND (AP1.ContactUpdates.LastName <> ''      --    and that `ContactUpdates`
311           OR AP1.ContactUpdates.LastName <> ' '     --    is not an empty string or
312           OR AP1.ContactUpdates.LastName IS NOT NULL)-- not equal to a space or
313                                                     --    `IS NOT NULL` (in other
314                                                     --    words, not no-value; must
315                                                     --    have a value) using
316                                                     --    parenthesis to make a
317                                                     --    block of three conditions
318       THEN CONCAT (                                 -- 4. action to be taken if
319           LEFT(AP1.ContactUpdates.FirstName, 1),    --    previous condition is
320           AP1.ContactUpdates.LastName,              --    true is the concatenation
321           '@',                                      --    of `VendorContactFName`,
322                                                     --    a single space and
323                                                     --    `VendorContactLName`, the
324                                                     --    `@` sign, followed by
325                                                     --    `VendorName` (value #6)
326           REPLACE(                                  --    4.01. getting value #5
327             REPLACE(                                --    4.02. getting value #4
328               REPLACE(                              --    4.03. getting value #3
329                 REPLACE(                            --    4.04. getting value #2
330                   REPLACE(                          --    4.05. getting value #1
331                     REPLACE(AP1.Vendors.VendorName, ' ', '')
332                                                     --    4.06. generating value #1
333                   , '.', '')                        --    4.07. generating value #2
334                 , ',', '')                          --    4.08. generating value #3
335               , '/', '')                            --    4.09. generating value #4
336             , '&', '')                              --    4.10. generating value #5
337           , '''', ''),                              --    4.11. generating value #6
338                                                     --          where `''`
339                                                     --          represents a single
340                                                     --          quote (`'`) used an
341                                                     --          escape character
342           '.foo'                                    --    and `.foo` to complete
343           )
344       ELSE ''
345       END                                           --    end of CASE clause to
```

```
    AS VendorContactEmail                          --    make email with alias
                                                   --    `VendorContactEmail`
FROM AP1.Vendors
LEFT JOIN AP1.ContactUpdates
  ON AP1.Vendors.VendorID = AP1.ContactUpdates.VendorID;


/* ***************************************************************************
          As with previous examples, we can use an alias for each table, which
          in this case, allows us to present neater code.

                    `v` for `AP1.Vendors`
                    `c` for `AP1.ContactUpdates`
 *************************************************************************** */

SELECT v.VendorID,
  v.VendorName,
  v.VendorAddress1,
  v.VendorAddress2,
  v.VendorCity,
  v.VendorState,
  v.VendorZipCode,
  v.VendorPhone,
  CONCAT (
    v.VendorContactFName,
    ' ',
    v.VendorContactLName
    ) AS ContactName,
  CONCAT (
    LEFT(v.VendorContactFName, 1),
    v.VendorContactLName,
    '@',
    REPLACE(
      REPLACE(
        REPLACE(
          REPLACE(
            REPLACE(v.VendorName, ' ', '')
          , '.', '')
        , ',', '')
      , '/', '')
    , '&', '')
  , '''', ''),
    '.foo'
    ) AS VendorContactEmail,
  v.DefaultTermsID,
  v.DefaultAccountNo,
  -- c.VendorID AS Expr1,
  CONCAT (
    c.FirstName,
    ' ',
    c.LastName
    ) AS ContactName,
  CASE
    WHEN (
        c.FirstName <> ''
        OR c.FirstName <> ' '
        OR c.FirstName IS NOT NULL
        )
      AND (
        c.LastName <> ''
        OR c.LastName <> ' '
        OR c.LastName IS NOT NULL
        )
      THEN CONCAT (
          LEFT(c.FirstName, 1),
          c.LastName,
          REPLACE(
            REPLACE(
```

```
                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. The relation of `UNION`, `EXCEPT` and `INTERSECT` is to affect the output
    between datasets (output of queries).  They both work if the datasets have
    the same number of columns with the same or compatible data types (how data
    is saved in the database) for each column.

    For example, dataset one (1) has three (3) columns with an integer (whole
    number), a `VARCHAR(25)` (variable character text with a maximum capacity
    for 25 characters) and a float (floating point decimal number).  This means
    that dataset two (2) must also have three (3) columns with a number
    (integer or compatible), a string (preferable a `VARCHAR(25)` or similar
    file type that will not allow data loss) and another number that may allow
    a floating point calculation (another float, integer, money, etc.) -- in
    this order, of course.  If you have multiple datasets, the new dataset must
    comply with the structure of the previous datasets.  If any dataset has
    different number of columns as the previous, you have to include fake
    columns (for example, `'no value' AS some_column_name`) or the statement
    will break.  If the data types are different and/or can't be converted
    during execution, the statement will break. For example, text `apple`
    cannot be converted into a number, while text `1` can be converted into a
    number.

    Row identifiers might be considered as part of the unique value for each
    row.  For example, a row with values `1 | 'apple' | 'red'` in dataset one
    (1) would be considered different from `2 | 'apple' | 'red'` in dataset two
    (2).  I would recommend not calling row IDs when using `UNION`, `EXCEPT` or
    `INTERSECT` to avoid logic errors.

    2.01. `UNION` simply merges two (2) datasets without duplicates.

          In the example below, we retrieve all contact names from
          `AP1.Vendors` and those found in `ContactUpdates`.  Since these are
          separate datasets, we can use `UNION` to return a single output.
 ******************************************************************************** */

SELECT VendorName,
  VendorContactFName,
  VendorContactLName
FROM AP1.Vendors
WHERE VendorContactFName IS NOT NULL
  AND VendorContactLName IS NOT NULL            -- returns 119 records

UNION

SELECT VendorName,
  FirstName,
  LastName
FROM AP1.ContactUpdates
INNER JOIN AP1.Vendors
  ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
                                                -- returns 7 records
```

```sql
ORDER BY VendorName;


/* ********************************************************************************
    2.02. `EXCEPT` returns the values in dataset one (1) without any value
          found in dataset two (2).  This means that dataset two (2) acts like
          a filter and is not returned in the output.

          In the example below, since the first query returned that some
          contacts may be on both datasets with different first names, we
          exclude these contacts from the first dataset.
 ******************************************************************************** */

SELECT VendorName,
  -- VendorContactFName,
  VendorContactLName
FROM AP1.Vendors
WHERE VendorContactFName IS NOT NULL
  AND VendorContactLName IS NOT NULL            -- would return 119 records

EXCEPT

SELECT VendorName,
  -- FirstName,
  LastName
FROM AP1.ContactUpdates
INNER JOIN AP1.Vendors
  ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
                                                -- has 2 values present in the
                                                -- first dataset
ORDER BY VendorName;                            -- only returns 117 records


/* ********************************************************************************
    2.03. `INTERSECT` returns the common values in dataset one (1) and two (2).
          It behaves similarly to `INNER JOIN`.

          In the example below, since the first query returned that some
          contacts may be on both datasets with different first names, we only
          call the last name of the contact persons.
 ******************************************************************************** */

SELECT VendorName,
  -- VendorContactFName,
  VendorContactLName
FROM AP1.Vendors
WHERE VendorContactFName IS NOT NULL
  AND VendorContactLName IS NOT NULL

INTERSECT

SELECT VendorName,
  -- FirstName,
  LastName
FROM AP1.ContactUpdates
INNER JOIN AP1.Vendors
  ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
ORDER BY VendorName;                            -- intersects two (2) rows
                                                -- 1. `VendorName` with value
                                                --    `Courier Companies, Inc`
                                                --    and `VendorContactLName`
                                                --    with value `Antavius`
                                                -- 2. `VendorName` with value
                                                --    `Federal Express
                                                --    Corporation` and
                                                --    `LasName` with value
                                                --    `Bucket`


```

```
/* ************************************************************************
 3. 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).

    3.01. 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

    3.02. 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

    3.03. 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

    3.04. 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

    3.05. Microsoft SQL Server is a relational database management system, or
          RDBMS, that supports a wide variety of transaction processing,
          business intelligence and analytics applications in corporate IT
          environments.  It's one of the three market-leading database
          technologies, along with Oracle Database and IBM's DB2.
          Like other RDBMS software, Microsoft SQL Server is built on top of
          SQL, a standardized programming language that database administrators
          (DBAs) and other IT professionals use to manage databases and query
          the data they contain.  SQL Server is tied to Transact-SQL (T-SQL),
          an implementation of SQL from Microsoft that adds a set of
          proprietary programming extensions to the standard language.  The
          original SQL Server code was developed in the 1980s by the former
          Sybase Inc., which is now owned by SAP.  Sybase initially built the
          software to run on Unix systems and minicomputer platforms.  It,
          Microsoft and Ashton-Tate Corp., then the leading vendor of PC
          databases, teamed up to produce the first version of what became
          Microsoft SQL Server, designed for the OS/2 operating system and
          released in 1989.
          https://searchsqlserver.techtarget.com/definition/SQL-Server

    3.06. Another form of flat file is one in which table data is gathered in
          lines of ASCII text with the value from each table cell separated by
          a comma and each row represented with a new line.  This type of flat
          file is also known as a comma-separated values file (CSV) file.
          http://searchsqlserver.techtarget.com/definition/flat-file

    3.07. A hierarchical database is a design that uses a one-to-many
          relationship for data elements.  Hierarchical database models use a
          tree structure that links a number of disparate elements to one
          `owner,` or `parent,` primary record.
          https://www.techopedia.com/definition/19782/hierarchical-database
```

3.08. Data Manipulation Language (DML) is the ``vocabulary used to retrieve and work with data... to add, modify, query, or remove data`` (https://msdn.microsoft.com/en-us/library/ff848766.aspx).

SELECT to retrieve records from one or more tables
https://techonthenet.com/sql/select.php

INSERT to insert a one or more records into a table
https://techonthenet.com/sql/insert.php

UPDATE to update existing records in the tables
https://techonthenet.com/sql/update.php

DELETE to delete a one or more records from a table
https://techonthenet.com/sql/delete.php

MERGE to insert, update, or delete operations on a target table based on the results of a join with a source table
https://msdn.microsoft.com/en-us/library/bb510625.aspx

3.09. Data Definition Language (DDL) is the ``vocabulary used to define data structures... to create, alter, or drop data structures`` (https://msdn.microsoft.com/en-us/library/ff848799.aspx).

USE to select any existing database in SQL schema [or output from another query]
http://tutorialspoint.com/sql/sql-select-database.htm

CREATE to create and define a table [or other database object]
https://techonthenet.com/sql/tables/create_table.php

ALTER to add a column, modify a column, drop a column, rename a column or rename a table [or other database object]
https://techonthenet.com/sql/tables/alter_table.php

DROP to remove or delete a table [or other database object]
https://techonthenet.com/sql/tables/drop_table.php

TRUNCATE to remove all records from a table
https://techonthenet.com/sql/truncate.php

DELETE to delete a one or more records from a table
https://techonthenet.com/sql/delete.php

3.10. Note that some of these statements can do more than what is covered in these notes for our first sessions.

For example, the `CREATE` statement is also used to create other database objects as well as access management, but we will not cover these other statements yet. Refer to https://msdn.microsoft.com/en-us/library/cc879262.aspx for more information on the `CREATE` statement.

On a personal note, when looking for information and/or explanation on how to use Microsoft technologies, in this case SQL Server, go to https://techonthenet.com/ or http://tutorialspoint.com/ as https://msdn.microsoft.com/ and other Microsoft websites often seem to be written for advanced users.

We will use DML and DDL in detail later in the course.

4. There are several data types (https://msdn.microsoft.com/en-us/library/ms187752.aspx) that you need to know if you are interested in taking the certification exam for Database Fundamentals. In everyday use, these are the most often used data types in T-SQL (http://searchsqlserver.techtarget.com/definition/T-SQL) -- the version of SQL (http://searchsqlserver.techtarget.com/definition/SQL) used in SQL Server (http://searchsqlserver.techtarget.com/definition/SQL-Server) -- are the following.

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

4.01. 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	converted to text; no longer numeric data and cannot be used in mathematical calculations

	DATE TIME	NVARCHAR	
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

4.02. 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.

5. 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
                      AND|OR condition2         --    rows/records you want to
                      AND|OR condition3         --    retrieve
                      ...

                    GROUP BY table1.field1,     -- 4. grouping fields not in an
                      table1.field2,            --    aggregate function
                      ...
                      table2.field1,
                      table2.field2,
                      ...
                      table3.field1,
                      table3.field2,
                      ...

                    ORDER BY                    -- 5. organizing rows/records
                      table1.field1 ASC|DESC,   --    (output) in ascending
                      table1.field2 ASC|DESC,   --    (`ASC`) or descending
                      ...                       --    (`DESC`) order
                      table2.field1 ASC|DESC,
                      table2.field2 ASC|DESC,
                      ...
                      table3.field1 ASC|DESC,
                      table3.field2 ASC|DESC,
                      ...

    5.01. In the example below, we retrieve all (`*`) columns from table
          `AP1.Vendors`.
 ************************************************************************ */

SELECT *
FROM AP1.Vendors;                               -- retrieves all values from
                                                -- table `AP1.Vendors`


/* ************************************************************************
    5.02. The only time you can use `SELECT` without `FROM` is when you want
          the machine to return a value, similar to `PRINT`.
 ************************************************************************ */

SELECT 9 * 8;                                   -- returns integer 72 (a
                                                -- mathematical equation)

SELECT 'Hello there';                           -- returns string `Hello there`
                                                -- (a simple string)


/* ************************************************************************
    5.03. As you can see in the examples above, we are not retrieving data from
```

```
           any table.  You can get the same results using `PRINT`.
 ************************************************************************** */

PRINT 9 * 8;                                    -- prints integer 72 (a
                                                -- mathematical equation)

PRINT 'Hello there';                            -- prints string `Hello there`
                                                -- (a simple string)


/* ************************************************************************
 6. We have covered built-in functions that affect strings.

    CONCAT()        allows you to concatenate strings together
                    https://techonthenet.com/sql_server/functions/concat.php

    + (plus)        allows you to concatenate 2 or more strings together
                    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

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

    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

    RTRIM()         removes all space characters from the right-hand side of a
                    string
                    https://techonthenet.com/sql_server/functions/rtrim.php

    SUBSTRING()     allows you to extract a substring from a string
                    https://techonthenet.com/sql_server/functions/substring.php

    UPPER()         converts all letters in the specified string to uppercase
                    https://techonthenet.com/sql_server/functions/upper.php

    Now we will see functions used with numeric values.

    AVG()           returns the average value of an expression
                    https://techonthenet.com/sql_server/functions/avg.php

    CEILING()       returns the smallest integer value that is greater than or
                    equal to a number
                    https://techonthenet.com/sql_server/functions/ceiling.php

    COUNT()         returns the count of an expression
                    https://techonthenet.com/sql_server/functions/count.php

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

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

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

      PRODUCT()       returns the product of an expression

                      https://www.sqlservercentral.com/articles/t-sql-in-sql-server-2025-the-operator

                      https://learn.microsoft.com/en-us/sql/t-sql/functions/product-aggregate-transact-sql?view=sql-server-ver17
                      * brand new T-SQL 2025

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

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

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

   7. In the examples below, we use each one of the numeric functions with the
      answer for each on the comment on the right.
   ********************************************************************** */

  SELECT SUM(InvoiceTotal) AS InvoiceTotalSUM,    -- returns 214290.51
    AVG(InvoiceTotal) AS InvoiceTotalAVG,         -- returns 1879.7413
    COUNT(InvoiceTotal) AS InvoiceTotalCOUNT,     -- returns 114
    ROUND(InvoiceTotal, 1) AS InvoiceTotalROUND,  -- returns 3813.30
                                                  --         40.20 ...
    FLOOR(InvoiceTotal) AS InvoiceTotalFLOOR,     -- returns 3813.00
                                                  --         40.00 ...
    CEILING(InvoiceTotal) AS InvoiceTotalCEILING, -- returns 3814.00
                                                  --         41.00 ...
    MAX(InvoiceTotal) AS InvoiceTotalMAX,         -- returns 37966.19
    MIN(InvoiceTotal) AS InvoiceTotalMIN,         -- returns 6.00
    RAND(InvoiceTotal) AS InvoiceTotalRAND,       -- returns 0.713591993212924
                                                  --         0.713610626184182...
    FORMAT(InvoiceTotal, 'c', 'en-us')            -- `c` for currency with
      AS InvoiceTotal,                            -- culture `en-us` (English US)
                                                  -- returns $3,813.33
                                                  --         $40.20 ...
    FORMAT(InvoiceDueDate, 'd', 'en-us')          -- `d` (lower case) for short
      AS InvoiceDueDate,                          -- date returning no leading
                                                  -- zeros with culture `en-us`
                                                  -- (English US);
                                                  -- returns 1/8/2012
                                                  --         1/10/2012 ...
    FORMAT(InvoiceDueDate, 'D', 'en-us')          -- `D` (upper case) for long
      AS InvoiceDueDate,                          -- date returning full day of
                                                  -- the week, full month, no
                                                  -- leading zeros with culture
                                                  -- `en-us` (English US);
                                                  -- returns
                                                  --    Sunday, January 8, 2012
                                                  --    Tuesday, January 10, 2012
                                                  --    ...
    FORMAT(InvoiceDueDate, 'MM/dd/yyyy', 'en-us') -- custom date using format
      AS InvoiceDueDate                           -- `MM/dd/yyyy` which overrides
                                                  -- culture `en-us` (English
                                                  -- US); returns 01/08/2012
                                                  --              01/10/2012 ...
  FROM AP1.Invoices
  GROUP BY InvoiceTotal,
    AP1.Invoices.InvoiceDueDate
```

```
1030
1031
1032  /* ********************************************************************
1033      7.01. When using an aggregate function, we must use `GROUP BY` and list all
1034            columns not in affected by any aggregate function.
1035
1036            In the example below, we retrieve `VendorState` plus the count of
1037            column `VendorState` for each `VendorState` (`COUNT(VendorState)`).
1038
1039            We can use `DISTINCT` to make sure that duplicate values (rows) are
1040            not included in the output of a query.
1041
1042            We can use `ORDER BY` to organize output by a specific column or list
1043            of columns.
1044
1045            The default option for `ORDER BY` is ascending (`ASC`), which can be
1046            omitted (1, 2, 3... a, b, c...).
1047
1048            The opposite option for `ORDER BY` is descending (`DESC`), which must
1049            be used if needed (...3, 2, 1 ...c, b, a).
1050   ******************************************************************** */
1051
1052  SELECT DISTINCT                                   -- 1. to avoid duplicates
1053    VendorState,                                    -- 2. column not in aggregate
1054                                                    --    function
1055    COUNT(VendorState)                              -- 3. column in aggregate
1056                                                    --    function (calculation)
1057  FROM AP1.Vendors                                  -- 4. from table `AP1.Vendors`
1058  GROUP BY VendorState                              -- 5. must use `GROUP BY` when
1059                                                    --    using any aggregate
1060                                                    --    function, listing all
1061                                                    --    columns not in the
1062                                                    --    aggregate function
1063  ORDER BY VendorState ASC;                         -- 6. organizing results by
1064                                                    --    column `VendorState` in
1065                                                    --    ascending order
1066
1067
1068  /* ********************************************************************
1069      7.02. In the example below, we retrieve `VendorID` plus the sum of column
1070            `PaymentTotal` for each `VendorID` (`SUM(PaymentTotal)`).
1071   ******************************************************************** */
1072
1073  SELECT DISTINCT                                   -- 1. to avoid duplicates
1074    VendorID,                                       -- 2. column not in aggregate
1075                                                    --    function
1076    SUM(PaymentTotal)                               -- 3. column in aggregate
1077                                                    --    function (calculation)
1078  FROM AP1.Invoices                                 -- 4. from table `AP1.Invoices`
1079  GROUP BY VendorID                                 -- 5. must use `GROUP BY` when
1080                                                    --    using any aggregate
1081                                                    --    function, listing all
1082                                                    --    columns not in the
1083                                                    --    aggregate function
1084  ORDER BY VendorID DESC;                           -- 6. organizing results by
1085                                                    --    column `VendorID` in
1086                                                    --    descending order
1087
1088
1089  /* ********************************************************************
1090   8. In the example below, the query returns all values from the `AP1.Vendors`
1091      table with all related values from table `AP1.Invoices`,
1092      `AP1.InvoiceLineItems` and `AP1.Terms`.
1093
1094      8.01. The relation between related tables `AP1.Invoices`,
1095            `AP1.InvoiceLineItems` and `AP1.Terms` is `INNER JOIN` since the
1096            value (row ID) of one table in referenced in another.
1097
1098      8.02. Dollar amounts are formatted as `c` (currency) with culture `en-us`
```

(English-United States). Dates are formatted as `MM/dd/yyyy` (two digits for month and day, four digits for year) and culture `en-us` (English-United States). Refer to https://msdn.microsoft.com/en-us/library/hh213506.aspx for more information. Note that formatting a numeric value changes it to an alpha-numeric value -- change in data type.

8.03. To include the average value of `InvoiceTotal` of all records from table `AP1.Invoices`, we use a sub-query (also referred to as nested query, http://tutorialspoint.com/sql/sql-sub-queries.htm). We use alias `AvgInvoiceTotal` to refer to this new column.

```
(
  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 Chinese	China	yuan
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 depending on culture (`$`)	`c`, `en-us`
d	day without leading zero, day without leading zero and complete year (03/25/2024)	`d`, `en-us`
D	whole day of the week, first letter capitalized;	`D`, `en-us`

	whole month, first letter capitalized; day without leading zero and complete year (Monday, April 17, 2023)	

DATEPART	OUTPUT	FORMAT

dw	whole day of the week, first letter capitalized (Monday)	`dw MMMM dd, yyyy`
		`dw MMMM d, yyyy`
		`dw MMMM dd, yy`
		`dw MMMM d, yy`

MMMM	whole month, first letter capitalized (April)	`MMMM dd, yyyy`
		`MMMM d, yyyy`
		`MMMM dd, yy`
		`MMMM d, yy`

MMM	month in abbreviation, first letter capitalized (Apr)	`MMM dd, yyyy`
		`MMM d, yyyy`
		`MMM dd, yy`
		`MMM d, yy`
		`dd-MMM-yy`(default Oracle)
		`d-MMM-yy` (default Oracle)

MM	month number with leading zero (04)	`MM/dd/yyyy`
		`MM/d/yyyy`
		`MM/dd/yy`
		`MM/d/yy`

M	month number without leading zero (4)	`M/dd/yyyy`
		`M/d/yyyy`
		`M/dd/yy`
		`M/d/yy`

dddd	day of week (Monday)	`dddd, MMM d, yyyy` `dddd, MMMM d, yyyy`
ddd	day of week abbreviation (Mon)	`ddd, MMM d, yyyy` `ddd, MMMM d, yyyy`

dd	day with leading zero (17)	`MM/dd/yyyy`
		`M/dd/yyyy`
		`MM/dd/yy`
		`M/dd/yy`

d	day without leading zero (17)	`MM/d/yyyy`
		`M/d/yyyy`
		`MM/d/yy`
		`M/d/yy`

yy	last two digits of year (23)	`M/dd/yy`
		`M/d/yy`
		`MM/d/yy`
		`M/d/yy`

yyyy	complete year (2023)	`M/dd/yyyy`
		`M/d/yyyy`
		`MM/d/yyyy`
		`M/d/yyyy`

```
                    | HH         | 24-hour,       | `HH:mm:ss`                  |
                    |            | military time  |                             |
                    |            | with leading   |                             |
                    |            | zero (20)      |                             |
                    +------------+----------------+-----------------------------+
                    | H          | 24-hour,       | `H:mm:ss`                   |
                    |            | military time  |                             |
                    |            | without        |                             |
                    |            | leading zero   |                             |
                    |            | (20)           |                             |
                    +------------+----------------+-----------------------------+
                    | hh         | 12-hour        | `hh:mm:ss`                  |
                    |            | (AM/PM), with  |                             |
                    |            | leading zero   |                             |
                    |            | (08 PM)        |                             |
                    +------------+----------------+-----------------------------+
                    | h          | 12-hour        | `h:mm:ss`                   |
                    |            | (AM/PM),       |                             |
                    |            | without        |                             |
                    |            | leading zero   |                             |
                    |            | (8 PM)         |                             |
                    +------------+----------------+-----------------------------+
                    | mm         | minutes (13)   | `HH:mm:ss`                  |
                    +------------+----------------+ `H:mm:ss`                   |
                    | ss         | seconds (58)   | `hh:mm:ss`                  |
                    |            |                | `h:mm:ss`                   |
                    +------------+----------------+-----------------------------+
                    | nnnnnnn    | six decimal    | `HH:mm:ss.nnnnnnn`          |
                    |            | spaces,        | `H:mm:ss.nnnnnnn`           |
                    |            | fractions of   | `hh:mm:ss.nnnnnnn`          |
                    |            | a second       | `h:mm:ss.nnnnnnn`           |
                    +------------+----------------+-----------------------------+

          Although we are using aggregate function `AVG()`, we do not need to
          use `GROUP BY` since the function is inside the sub-query.

          Go to https://docs.microsoft.com/en-us/sql/t-sql/functions/format-transact-sql
          for more information on `FORMAT()`.
 ************************************************************************** */

SELECT DISTINCT AP1.Vendors.VendorID,
  AP1.Vendors.VendorName,
  CONCAT (                                      -- 1. concatenating
    AP1.Vendors.VendorAddress1,                 --    `VendorAddress1`, an
    ' ',                                        --    empty space and
    AP1.Vendors.VendorAddress2                  --    `VendorAddress2`
    ) AS VendorAddress,                         --    as `VendorAddress`
  AP1.Vendors.VendorCity,
  AP1.Vendors.VendorState,
  CONCAT (                                      -- 2. concatenating
    AP1.Vendors.VendorZipCode,                  --    `VendorZipCode` and a
    '-0000'                                     --    dummy Plus4 as
    ) AS VendorZipCode,                         --    VendorZipCode


  CASE
    WHEN AP1.Vendors.VendorPhone <> ''          -- 3. checking that
      OR AP1.Vendors.VendorPhone <> ' '         --    `VendorPhone` is not an
      OR AP1.Vendors.VendorPhone IS NOT NULL    --    empty string or not a
                                                --    space or not `IS NOT
                                                --    NULL` (in other
                                                --    words, not no-value; must
                                                --    have a value) using
    THEN CONCAT (                               -- 4. concatenating an opening
        '(',                                    --    parenthesis, the first 3
        LEFT(AP1.Vendors.VendorPhone, 3),       --    characters of
                                                --    `VendorPhone` (area
                                                --    code), corresponding
        ') ',                                   --    closing parenthesis with
```

```
1304          SUBSTRING(AP1.Vendors.VendorPhone, 4,3),--    a space, the substring
1305                                                  --    from `VendorPhone`
1306                                                  --    starting with character 4
1307                                                  --    taking 3 characters
1308                                                  --    (branch exchange), a
1309          '-',                                    --    hyphen and the 4 four
1310          RIGHT(AP1.Vendors.VendorPhone, 4)       --    characters of
1311                                                  --    `VendorPhone`
1312                                                  --    (subscriber number) using
1313          )                                       --    alias `VendorPhone`
1314        ELSE ''
1315      END
1316        AS VendorPhone,
1317    LTRIM(RTRIM(                                  -- 5. trimming the output of
1318                                                  --    the concatenation of
1319        CONCAT(AP1.Vendors.VendorContactLName,    --    `VendorContactLName`, a
1320        ', ',                                     --    comma with a space and
1321        AP1.Vendors.VendorContactFName))          --    `VendorContactFName`
1322    ) AS VendorContactName,                       --    using alias
1323                                                  --    `VendorContactName`
1324    AP1.Vendors.DefaultAccountNo,
1325    AP1.Invoices.InvoiceID,
1326    AP1.Invoices.InvoiceNumber,
1327    FORMAT(AP1.Invoices.InvoiceDate,              -- 5. formatting column as
1328      'MM/dd/yyyy', 'en-us')                      --    `MM/dd/yyyy` (date) with
1329                                                  --    culture `en-us` as
1330    AS InvoiceDate,                               --    `InvoiceDate`
1331    FORMAT(AP1.Invoices.InvoiceTotal,             -- 7. formatting column as
1332      'c', 'en-us')                               --    `c` (currency) with
1333                                                  --    culture `en-us` with
1334    AS InvoiceTotal,                              --    alias `InvoiceTotal`
1335    (
1336      SELECT                                      -- 8. embedded query calling
1337        FORMAT(AVG(AP1.Invoices.InvoiceTotal),    --    `AVG(InvoiceTotal)`
1338          'c', 'en-us')                           --    formatted as `c`
1339                                                  --    (currency) with culture
1340                                                  --    `en-us`
1341      FROM AP1.Invoices                           --    from all values in table
1342                                                  --    `AP1.Invoices` as
1343    ) AS AvgInvoiceTotal,                         --    `AvgInvoiceTotal`
1344    FORMAT(AP1.Invoices.PaymentTotal,             -- 9. formatting column as `c`
1345      'c', 'en-us')                               --    (currency) with culture
1346    AS PaymentTotal,                              --    `en-us` as `PaymentTotal`
1347    FORMAT(AP1.Invoices.CreditTotal,              -- 10. formatting column as `c`
1348      'c', 'en-us')                               --     (currency) with culture
1349    AS CreditTotal,                               --     `en-us` as `CreditTotal`
1350    FORMAT(AP1.Invoices.InvoiceDueDate,           -- 11. formatting column as
1351      'MM/dd/yyyy', 'en-us')                      --     `MM/dd/yyyy` (date) with
1352                                                  --     culture `en-us` as
1353    AS InvoiceDueDate,                            --     `InvoiceDueDate`
1354    FORMAT(AP1.Invoices.PaymentDate,              -- 12. formatting column as
1355      'MM/dd/yyyy', 'en-us')                      --     `MM/dd/yyyy` (date) with
1356                                                  --     culture `en-us` as
1357    AS PaymentDate,                               --     `PaymentDate`
1358    AP1.InvoiceLineItems.InvoiceSequence,
1359    AP1.InvoiceLineItems.AccountNo,
1360    FORMAT(AP1.InvoiceLineItems.InvoiceLineItemAmount,
1361                                                  -- 13. formatting column as
1362      'c', 'en-us')                               --    `c` (currency) with
1363                                                  --    culture `en-us` as
1364    AS InvoiceLineItemAmount,                     --    `InvoiceLineItemAmount`
1365    AP1.InvoiceLineItems.InvoiceLineItemDescription,
1366    AP1.Terms.TermsDescription,
1367    AP1.Terms.TermsDueDays
1368  FROM AP1.InvoiceLineItems                       -- 14. from
1369                                                  --     `AP1.InvoiceLineItems`
1370  INNER JOIN AP1.Invoices                         --     using `INNER JOIN` to
1371                                                  --     to connect to
1372                                                  --     `AP1.Invoices` to get
```

```
                                                    --     all shared values from
  ON AP1.InvoiceLineItems.InvoiceID = AP1.Invoices.InvoiceID
                                                    --     `AP1.InvoiceLineItems`
                                                    --     and `AP1.Invoices`
INNER JOIN AP1.Terms                                --     using `INNER JOIN` to
                                                    --     connect to `AP1.Terms`
                                                    --     to get all shared values
                                                    --     from
  ON AP1.Invoices.TermsID = AP1.Terms.TermsID       --     (`AP1.InvoiceLineItems`
                                                    --     and `AP1.Invoices`) and
                                                    --     `AP1.Terms` using
RIGHT JOIN AP1.Vendors                              --     `RIGHT JOIN` to connect
                                                    --     to `AP1.Vendors` to get
                                                    --     values from
                                                    --     `AP1.Vendors` and
                                                    --     related data from
  ON AP1.Invoices.VendorID=AP1.Vendors.VendorID     --     (`AP1.InvoiceLineItems`
                                                    --     and `AP1.Invoices` and
                                                    --     `AP1.Terms`)
ORDER BY                                            -- 15. ordering results by
  AP1.Vendors.VendorName,                           --     `VendorName`first and
  AP1.Invoices.InvoiceID;                           --     then by `InvoiceID`


/* ***************************************************************************
          As with previous example, we can use an alias for each table, which
          in this case, allows us to present neater code.

                    `il` for `AP1.InvoiceLineItems`
                    `i`  for `AP1.Invoices`
                    `t`  for `AP1.Terms`
                    `v`  for `AP1.Vendors`
 *************************************************************************** */

SELECT DISTINCT v.VendorID,
  v.VendorName,
  CONCAT (
    v.VendorAddress1,
    ' ',
    v.VendorAddress2
    ) AS VendorAddress,
  v.VendorCity,
  v.VendorState,
  CONCAT (
    v.VendorZipCode,
    '-0000'
    ) AS VendorZipCode,
  CASE
    WHEN v.VendorPhone <> ''
      OR v.VendorPhone <> ' '
      OR v.VendorPhone IS NOT NULL
      THEN CONCAT (
          '(',
          LEFT(v.VendorPhone, 3),
          ') ',
          SUBSTRING(v.VendorPhone, 4, 3),
          RIGHT(v.VendorPhone, 4)
          )
    ELSE ''
    END AS VendorPhone,
  LTRIM(RTRIM(CONCAT (
        v.VendorContactLName,
        ', ',
        v.VendorContactFName
        ))) AS VendorContactName,
  v.DefaultAccountNo,
  i.InvoiceID,
  i.InvoiceNumber,
  FORMAT(i.InvoiceDate, 'MM/dd/yyyy', 'en-us') AS InvoiceDate,
```

```
    FORMAT(i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
    (
      SELECT FORMAT(AVG(i.InvoiceTotal), 'c', 'en-us')
      FROM AP1.Invoices AS i                        -- 1. alias `i` not needed as
                                                    --    this is a sub-query, but
                                                    --    solely included for
                                                    --    aesthetics in the example
                                                    -- 2. outer aliases needed
    ) AS AvgInvoiceTotal,
    FORMAT(i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
    FORMAT(i.CreditTotal, 'c', 'en-us') AS CreditTotal,
    FORMAT(i.InvoiceDueDate, 'MM/dd/yyyy', 'en-us') AS InvoiceDueDate,
    FORMAT(i.PaymentDate, 'MM/dd/yyyy', 'en-us') AS PaymentDate,
    il.InvoiceSequence,
    il.AccountNo,
    FORMAT(il.InvoiceLineItemAmount, 'c', 'en-us')
      AS InvoiceLineItemAmount,
    il.InvoiceLineItemDescription,
    t.TermsDescription,
    t.TermsDueDays
  FROM AP1.InvoiceLineItems AS il
  INNER JOIN AP1.Invoices AS i
    ON il.InvoiceID = i.InvoiceID
  INNER JOIN AP1.Terms AS t
    ON i.TermsID = t.TermsID
  RIGHT JOIN AP1.Vendors AS v
    ON i.VendorID = v.VendorID
  ORDER BY v.VendorName,
    i.InvoiceID;


  /* ************************************************************************
   9. To get the difference between two dates, we use `DATEDIFF()`, which
      ``returns the difference between two date values, based on the interval
      specified`` (https://techonthenet.com/sql_server/functions/datediff.php).

      We also call functions `DAY()`
      (https://techonthenet.com/sql_server/functions/day.php), `MONTH()`
      (https://techonthenet.com/sql_server/functions/month.php) and `YEAR()`
      (https://techonthenet.com/sql_server/functions/year.php).

      9.01. In the example below, we use `01/01/2017` as the starting date and
            `12/01/2025` as the end date.
   ************************************************************************ */

  SELECT DATEDIFF(DAY,'01/01/2017','12/01/2025') AS DatediffDays, -- 3,256 days
    DATEDIFF(MONTH,'01/01/2017','12/01/2025') AS DatediffMonths,  --   107 months
    DATEDIFF(YEAR,'01/01/2017','12/01/2025') AS DatediffYears;    --     8 years


  /* ************************************************************************
      9.02. Instead of hard-coding today's date, we can use function `GETDATE()`
            to retrieve the local system datetime -- `2025-12-01 21:52:16.327`.
   ************************************************************************ */

  SELECT DATEDIFF(DAY, '01/01/2017', GETDATE()) AS DatediffDays,  -- 3,256 days
    DATEDIFF(MONTH, '01/01/2017', GETDATE()) AS DatediffMonths,   --   107 months
    DATEDIFF(YEAR, '01/01/2017', GETDATE()) AS DatediffYears;     --     8 years


  /* ************************************************************************
   https://folvera.commons.gc.cuny.edu/?p=1351
   ************************************************************************ */
```