```
/* *****************************************************************************
          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
```

```sql
            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;

```

```sql

/* ****************************************************************************
          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
              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`

  CASE
    WHEN (AP1.ContactUpdates.FirstName <> ''    -- 4. checking if
      OR AP1.ContactUpdates.FirstName <> ' '    --    `ContactUpdates` is not
      OR AP1.ContactUpdates.FirstName IS NOT NULL)--    an empty string or not
                                                --    equal to a space or
                                                --    `IS NOT NULL` (in other
                                                --    words, not no-value; must
                                                --    have a value) using
                                                --    parenthesis to make a
                                                --    block of three conditions
      AND (AP1.ContactUpdates.LastName <> ''    --    and that `ContactUpdates`
        OR AP1.ContactUpdates.LastName <> ' '   --    is not an empty string or
```

```
        OR AP1.ContactUpdates.LastName IS NOT NULL)-- not equal to a space or
                                                --    `IS NOT NULL` (in other
                                                --    words, not no-value; must
                                                --    have a value) using
                                                --    parenthesis to make a
                                                --    block of three conditions
    THEN CONCAT (                               -- 4. action to be taken if
        LEFT(AP1.ContactUpdates.FirstName, 1),  --    previous condition is
        AP1.ContactUpdates.LastName,            --    true is the concatenation
        '@',                                    --    of `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
        )
    ELSE ''
    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,
```

```sql
    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(
                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. 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

      2.05. 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

      2.06. 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

      2.07. 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

      2.08. 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
```

```

     2.09. 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

     2.10. 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.

  3. 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

    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                   | converted to text;  no longer   |
                    | DECIMAL               | numeric data and cannot be used |
                    | DATETIME  to  VARCHAR | in mathematical calculations    |
                    | DATE          NVARCHAR|                                 |
                    | TIME                  |                                 |
                    +-----------------------+---------------------------------+
                    |               INT     | straight conversion to proper   |
                    |               DECIMAL | data type as long as the string |
                    | VARCHAR   to  DATETIME| field only has numbers and      |
                    | NVARCHAR      DATE    | structure is correct (for       |
                    |               TIME    | 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
```

```
                      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,
                      ...

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

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


/* ****************************************************************************
    4.02. 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)


/* ****************************************************************************
    4.02. 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)


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

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

    `+`             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

    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

 6. 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


/* ****************************************************************************
    6.01. When using an aggregate function, we must use `GROUP BY` and list all
          columns not in affected by any aggregate function.

          In the example below, we retrieve `VendorState` plus the count of
          column `VendorState` for each `VendorState` (`COUNT(VendorState)`).

          We can use `DISTINCT` to make sure that duplicate values (rows) are
          not included in the output of a query.

          We can use `ORDER BY` to organize output by a specific column or list
          of columns.

          The default option for `ORDER BY` is ascending (`ASC`), which can be
          omitted (1, 2, 3... a, b, c...).

          The opposite option for `ORDER BY` is descending (`DESC`), which must
          be used if needed (...3, 2, 1 ...c, b, a).
 *************************************************************************** */

SELECT DISTINCT                                 -- 1. to avoid duplicates
  VendorState,                                  -- 2. column not in aggregate
                                                --    function
  COUNT(VendorState)                            -- 3. column in aggregate
                                                --    function (calculation)
FROM AP1.Vendors                                -- 4. from table `AP1.Vendors`
GROUP BY VendorState                            -- 5. must use `GROUP BY` when
                                                --    using any aggregate
                                                --    function, listing all
                                                --    columns not in the
                                                --    aggregate function
ORDER BY VendorState ASC;                       -- 6. organizing results by
                                                --    column `VendorState` in
                                                --    ascending order


/* ****************************************************************************
    6.02. In the example below, we retrieve `VendorID` plus the sum of column
          `PaymentTotal` for each `VendorID` (`SUM(PaymentTotal)`).
 *************************************************************************** */

SELECT DISTINCT                                 -- 1. to avoid duplicates
  VendorID,                                     -- 2. column not in aggregate
                                                --    function
  SUM(PaymentTotal)                             -- 3. column in aggregate
                                                --    function (calculation)
FROM AP1.Invoices                               -- 4. from table `AP1.Invoices`
GROUP BY VendorID                               -- 5. must use `GROUP BY` when
                                                --    using any aggregate
                                                --    function, listing all
                                                --    columns not in the
                                                --    aggregate function
ORDER BY VendorID DESC;                         -- 6. organizing results by
                                                --    column `VendorID` in
                                                --    descending order


/* ****************************************************************************
 7. In the example below, the query returns all values from the `AP1.Vendors`
    table with all related values from table `AP1.Invoices`,
    `AP1.InvoiceLineItems` and `AP1.Terms`.

    7.01. The relation between related tables `AP1.Invoices`,
          `AP1.InvoiceLineItems` and `AP1.Terms` is `INNER JOIN` since the
          value (row ID) of one table in referenced in another.

    7.02. 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.

    7.03. 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    | 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`                |
              |           | the week,     |                             |
              |           | first letter  |                             |
              |           | capitalized;  |                             |
              |           | whole month,  |                             |
              |           | first letter  |                             |
              |           | capitalized;  |                             |
              |           | day without   |                             |
              |           | leading zero  |                             |
              |           | and complete  |                             |
              |           | year (Monday, |                             |
              |           | April 17,     |                             |
              |           | 2023)         |                             |
              +-----------+---------------+-----------------------------+

              +-----------+---------------+-----------------------------+
              | DATEPART  | OUTPUT        | FORMAT                      |
              +-----------+---------------+-----------------------------+
              | dw        | whole day of  | `dw MMMM dd, yyyy`          |
              |           | the week,     | `dw MMMM d, yyyy`           |
              |           | first letter  | `dw MMMM dd, yy`            |
              |           | capitalized   | `dw MMMM d, yy`             |
              |           | (Monday)      |                             |
              +-----------+---------------+-----------------------------+
              | MMMM      | whole month,  | `MMMM dd, yyyy`             |
              |           | first letter  | `MMMM d, yyyy`              |
              |           | capitalized   | `MMMM dd, yy`               |
              |           | (April)       | `MMMM d, yy`                |
              +-----------+---------------+-----------------------------+
              | MMM       | month in      | `MMM dd, yyyy`              |
              |           | abbreviation, | `MMM d, yyyy`               |
              |           | first letter  | `MMM dd, yy`                |
              |           | capitalized   | `MMM d, yy`                 |
              |           | (Apr)         | `dd-MMM-yy`(default Oracle) |
              |           |               | `d-MMM-yy` (default Oracle) |
              +-----------+---------------+-----------------------------+
              | MM        | month number  | `MM/dd/yyyy`                |
              |           | with leading  | `MM/d/yyyy`                 |
              |           | zero (04)     | `MM/dd/yy`                  |
              |           |               | `MM/d/yy`                   |
              +-----------+---------------+-----------------------------+
              | M         | month number  | `M/dd/yyyy`                 |
              |           | without       | `M/d/yyyy`                  |
              |           | leading zero  | `M/dd/yy`                   |
              |           | (4)           | `M/d/yy`                    |
              +-----------+---------------+-----------------------------+
              | dddd      | day of week   | `dddd, MMM d, yyyy`         |
              |           | (Monday)      | `dddd, MMMM d, yyyy`        |
              +-----------+---------------+-----------------------------+
              | ddd       | day of week   | `ddd, MMM d, yyyy`          |
              |           | abbreviation  } `ddd, MMMM d, yyyy`         |
              |           | (Mon)         |                             |
              +-----------+---------------+-----------------------------+
              | dd        | day with      | `MM/dd/yyyy`                |
              |           | leading zero  | `M/dd/yyyy`                 |
              |           | (17)          | `MM/dd/yy`                  |
              |           |               | `M/dd/yy`                   |
              +-----------+---------------+-----------------------------+
              | d         | day without   | `MM/d/yyyy`                 |
              |           | leading zero  | `M/d/yyyy`                  |
              |           | (17)          | `MM/d/yy`                   |
              |           |               | `M/d/yy`                    |
              +-----------+---------------+-----------------------------+
              | yy        | last two      | `M/dd/yy`                   |
              |           | digits of year| `M/d/yy`                    |
              |           | (23)          | `MM/d/yy`                   |
              |           |               | `M/d/yy`                    |
              +-----------+---------------+-----------------------------+
              | yyyy      | complete year | `M/dd/yyyy`                 |
              |           | (2023)        | `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
SUBSTRING(AP1.Vendors.VendorPhone, 4, 3),--   a space, the substring
                                     --    from `VendorPhone`
                                     --    starting with character 4
                                     --    taking 3 characters
                                     --    (branch exchange), a
'-',                                    --    hyphen and the 4 four
RIGHT(AP1.Vendors.VendorPhone, 4)       --    characters of
                                     --    `VendorPhone`
                                     --    (subscriber number) using
)                                       --    alias `VendorPhone`
ELSE ''
END
AS VendorPhone,
LTRIM(RTRIM(                                  -- 5. trimming the output of
```

```sql
                                                --    the concatenation of
      CONCAT(AP1.Vendors.VendorContactLName,    --    `VendorContactLName`, a
      ', ',                                     --    comma with a space and
      AP1.Vendors.VendorContactFName))          --    `VendorContactFName`
  ) AS VendorContactName,                       --    using alias
                                                --    `VendorContactName`
  AP1.Vendors.DefaultAccountNo,
  AP1.Invoices.InvoiceID,
  AP1.Invoices.InvoiceNumber,
  FORMAT(AP1.Invoices.InvoiceDate,              -- 5. formatting column as
    'MM/dd/yyyy', 'en-us')                      --    `MM/dd/yyyy` (date) with
                                                --    culture `en-us` as
  AS InvoiceDate,                               --    `InvoiceDate`
  FORMAT(AP1.Invoices.InvoiceTotal,             -- 7. formatting column as
    'c', 'en-us')                               --    `c` (currency) with
                                                --    culture `en-us` with
  AS InvoiceTotal,                              --    alias `InvoiceTotal`
  (
    SELECT                                      -- 8. embedded query calling
      FORMAT(AVG(AP1.Invoices.InvoiceTotal),    --    `AVG(InvoiceTotal)`
        'c', 'en-us')                           --    formatted as `c`
                                                --    (currency) with culture
                                                --    `en-us`
    FROM AP1.Invoices                           --    from all values in table
                                                --    `AP1.Invoices` as
  ) AS AvgInvoiceTotal,                         --    `AvgInvoiceTotal`
  FORMAT(AP1.Invoices.PaymentTotal,             -- 9. formatting column as `c`
    'c', 'en-us')                               --    (currency) with culture
  AS PaymentTotal,                              --    `en-us` as `PaymentTotal`
  FORMAT(AP1.Invoices.CreditTotal,              -- 10. formatting column as `c`
    'c', 'en-us')                               --     (currency) with culture
  AS CreditTotal,                               --     `en-us` as `CreditTotal`
  FORMAT(AP1.Invoices.InvoiceDueDate,           -- 11. formatting column as
    'MM/dd/yyyy', 'en-us')                      --     `MM/dd/yyyy` (date) with
                                                --     culture `en-us` as
  AS InvoiceDueDate,                            --     `InvoiceDueDate`
  FORMAT(AP1.Invoices.PaymentDate,              -- 12. formatting column as
    'MM/dd/yyyy', 'en-us')                      --     `MM/dd/yyyy` (date) with
                                                --     culture `en-us` as
  AS PaymentDate,                               --     `PaymentDate`
  AP1.InvoiceLineItems.InvoiceSequence,
  AP1.InvoiceLineItems.AccountNo,
  FORMAT(AP1.InvoiceLineItems.InvoiceLineItemAmount,
                                                -- 13. formatting column as
    'c', 'en-us')                               --    `c` (currency) with
                                                --    culture `en-us` as
  AS InvoiceLineItemAmount,                     --    `InvoiceLineItemAmount`
  AP1.InvoiceLineItems.InvoiceLineItemDescription,
  AP1.Terms.TermsDescription,
  AP1.Terms.TermsDueDays
FROM AP1.InvoiceLineItems                       -- 14. from
                                                --     `AP1.InvoiceLineItems`
```

```
INNER JOIN AP1.Invoices                         --     using `INNER JOIN` to
                                                --     to connect to
                                                --     `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 <> ''
```

```sql
      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
    ) 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;


/* ***************************************************************************
 8. 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).

    8.01. In the example below, we use `01/01/2017` as the starting date and
          `04/17/2023` as the end date.
 **************************************************************************** */

SELECT DATEDIFF(DAY,'01/01/2017','04/17/2023') AS DatediffDays, -- 2,297 days
  DATEDIFF(MONTH,'01/01/2017','04/17/2023') AS DatediffMonths,  --    75 months
  DATEDIFF(YEAR,'01/01/2017','04/17/2023') AS DatediffYears;    --     6 years


/* *****************************************************************************
    8.02. Instead of hard-coding today's date, we can use function `GETDATE()`
          to retrieve the local system datetime.
 **************************************************************************** */

SELECT DATEDIFF(DAY, '01/01/2017', GETDATE()) AS DatediffDays,  -- 2,297 days
  DATEDIFF(MONTH, '01/01/2017', GETDATE()) AS DatediffMonths,   --    75 months
  DATEDIFF(YEAR, '01/01/2017', GETDATE()) AS DatediffYears;     --     6 years


/* *****************************************************************************
 https://folvera.commons.gc.cuny.edu/?p=1223
 **************************************************************************** */
```