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

  SESSION #7 (2025/12/08): CREATING DATABASE OBJECTS

  1. Understanding functions `CONVERT()`, `CAST()`, `DAY()`, `MONTH()`,
     `YEAR()` and `GETDATE()`
  2. Creating, dropping and altering views
 ************************************************************************

 1. LAB #5
    Write a query
    1.01. to call all columns and values shared by tables `AP1.ContactUpdates`
          and `AP1.Vendors` (`INNER JOIN`),
    1.02. retrieving only rows with `AP1.Vendors.VendorState` with values of
          `NY`, `NJ` and `CA`
    1.03. using `CASE` to replace `NY` to `New York`, `NJ` to `New Jersey`,
          `CA` to `California` and any other value to `Other`
    1.04. ordered first by `AP1.Vendors.VendorState` and then by
          `AP1.Vendors.VendorID`.
    BONUS: Make a view in schema `lab05` in database `labs`.
 ************************************************************************ */

SELECT AP1.ContactUpdates.VendorID,
  AP1.ContactUpdates.LastName,
  AP1.ContactUpdates.FirstName,
  -- AP1.Vendors.VendorID AS Expr1,             -- 1. duplicate column name
                                                --    commented out
  AP1.Vendors.VendorName,
  AP1.Vendors.VendorAddress1,
  AP1.Vendors.VendorAddress2,
  AP1.Vendors.VendorCity,
  CASE                                          -- 2. beginning of logic
    WHEN AP1.Vendors.VendorState = 'NY'         --    2.1. checking for value
      THEN 'New York'                           --         `NY` and return
                                                --         value `New York`
    WHEN AP1.Vendors.VendorState = 'NJ'         --    2.2. checking for value
      THEN 'New Jersey'                         --         `NY` and return
                                                --         value `New Jersey`
    WHEN AP1.Vendors.VendorState = 'CA'         --    2.3. checking for value
      THEN 'California'                         --         `NY` and return
                                                --         value `California`
    ELSE 'Other'                                --    2.4. checking for other
                                                --         values and return
                                                --         value `Other`
    END AS VendorState,
  AP1.Vendors.VendorZipCode,
  AP1.Vendors.VendorPhone,
  AP1.Vendors.VendorContactLName,
  AP1.Vendors.VendorContactFName,
  AP1.Vendors.DefaultTermsID,
  AP1.Vendors.DefaultAccountNo
FROM AP1.ContactUpdates
INNER JOIN AP1.Vendors
  ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
WHERE AP1.Vendors.VendorState IN (              -- 3. indicating what values we
    'NY',                                       --    query to return
    'NJ',
    'CA'
    );


/* ************************************************************************
    At this point, we make a view in schema `lab05` in database `labs`
    excluding the `ORDER BY` clause as it cannot be used when creating views.
 ************************************************************************ */
```

```
70
71  CREATE SCHEMA lab05;                                -- 1. creating schema `labs05`
72
73  CREATE VIEW lab05.ContactUpdatesVendorsVW       -- 2. creating view
74  AS                                                  --    `ContactUpdatesVendorsVW`
75  SELECT SF25SQL6172025.AP1.ContactUpdates.VendorID,--  in database `labs`
76    SF25SQL6172025.AP1.ContactUpdates.LastName,   --    hence calling database
77    SF25SQL6172025.AP1.ContactUpdates.FirstName,  --    `SF25SQL6172025` for each
78    -- SF25SQL6172025.AP1.Vendors.VendorID AS Expr1,--  table
79    SF25SQL6172025.AP1.Vendors.VendorName,
80    SF25SQL6172025.AP1.Vendors.VendorAddress1,
81    SF25SQL6172025.AP1.Vendors.VendorAddress2,
82    SF25SQL6172025.AP1.Vendors.VendorCity,
83    -- case to replace NY with New York...
84    CASE
85      WHEN SF25SQL6172025.AP1.Vendors.VendorState = 'NY'
86        THEN 'New York'
87      WHEN SF25SQL6172025.AP1.Vendors.VendorState = 'NJ'
88        THEN 'New Jersey'
89      WHEN SF25SQL6172025.AP1.Vendors.VendorState = 'CA'
90        THEN 'California'
91      ELSE 'Other'
92      END AS VendorState,
93    SF25SQL6172025.AP1.Vendors.VendorZipCode,
94    SF25SQL6172025.AP1.Vendors.VendorPhone,
95    SF25SQL6172025.AP1.Vendors.VendorContactLName,
96    SF25SQL6172025.AP1.Vendors.VendorContactFName,
97    SF25SQL6172025.AP1.Vendors.DefaultTermsID,
98    SF25SQL6172025.AP1.Vendors.DefaultAccountNo
99  FROM SF25SQL6172025.AP1.ContactUpdates
100 INNER JOIN SF25SQL6172025.AP1.Vendors
101   ON SF25SQL6172025.AP1.ContactUpdates.VendorID = SF25SQL6172025.AP1.Vendors.VendorID
102 WHERE SF25SQL6172025.AP1.Vendors.VendorState IN (
103     'NY',
104     'NJ',
105     'CA'
106     );
107
108
109 /* ************************************************************************
110           As with previous examples, we can use an alias for each table, which
111           in this case, allows us to present neater code.
112
113                     `c` for `SF25SQL6172025.AP1.ContactUpdates`
114                     `v` for `SF25SQL6172025.AP1.Vendors`
115  ************************************************************************ */
116
117 CREATE VIEW lab05.ContactUpdatesVendorsVW
118 AS
119 SELECT c.VendorID,
120   c.LastName,
121   c.FirstName,
122   -- v.VendorID AS Expr1,
123   v.VendorName,
124   v.VendorAddress1,
125   v.VendorAddress2,
126   v.VendorCity,
127   CASE
128     WHEN v.VendorState = 'NY'
129       THEN 'New York'
130     WHEN v.VendorState = 'NJ'
131       THEN 'New Jersey'
132     WHEN v.VendorState = 'CA'
133       THEN 'California'
134     ELSE 'Other'
135     END AS VendorState,
136   v.VendorZipCode,
137   v.VendorPhone,
138   v.VendorContactLName,
```

```
    v.VendorContactFName,
    v.DefaultTermsID,
    v.DefaultAccountNo
FROM SF25SQL6172025.AP1.ContactUpdates AS c
INNER JOIN SF25SQL6172025.AP1.Vendors AS v
  ON c.VendorID = v.VendorID
WHERE v.VendorState IN (
    'NY',
    'NJ',
    'CA'
    );


/* ************************************************************************
 2. LAB #6
    Write a query without duplicate rows (`SELECT DISTINCT`)
    2.01. to call all shared values from tables `AP1.Invoices` and `AP1.Terms`,
    2.02. to format all dates as `yyyy-MM-dd` and currency as pounds sterling
          (culture `en-gb`)
    2.03. where `AP1.Invoices.PaymentTotal` is greater than the average value
          of `AP1.Invoices.InvoiceTotal` (sub-query within the `WHERE` clause)
          and `AP1.Invoices.PaymentDate` is not null.
    HINT: `AVG(AP1.Invoices.InvoiceTotal) FROM AP1.Invoices` for the nested
          query value
    BONUS: Make a view in schema `lab06` in database `labs` (already created in
          lab #5).
 ************************************************************************ */

SELECT AP1.Invoices.InvoiceID,
  AP1.Invoices.VendorID,
  AP1.Invoices.InvoiceNumber,
  FORMAT(AP1.Invoices.InvoiceDate,              -- 1. formatting column
                                                --    `InvoiceDate` as
    'yyyy-MM-dd', 'en-gb')                      --    `yyyy-MM-dd` date with
                                                --    culture `en-gb` using
     AS InvoiceDate,                            --    alias `InvoiceDate`
  FORMAT(AP1.Invoices.InvoiceTotal,             -- 2. formatting column
                                                --    `InvoiceTotal` as
    'c', 'en-gb')                               --    `c` (currency) with
                                                --    culture `en-gb` using
    AS InvoiceTotal,                            --    alias `InvoiceTotal`
  FORMAT(AP1.Invoices.PaymentTotal,             -- 3. formatting column
    'c', 'en-gb')                               --    as `c` (currency) with
                                                --    culture `en-gb` using
    AS PaymentTotal,                            --    alias `PaymentTotal`
  FORMAT(AP1.Invoices.CreditTotal,              -- 4. formatting column
    'c', 'en-gb')                               --    as `c` (currency) with
                                                --    culture `en-gb` using
    AS CreditTotal,                             --    alias `CreditTotal`
  AP1.Invoices.TermsID,
  FORMAT(AP1.Invoices.InvoiceDueDate,           -- 5. formatting column
    'yyyy-MM-dd', 'en-gb')                      --    as `yyyy-MM-dd` date with
                                                --    culture `en-gb` using
    AS InvoiceDueDate,                          --    alias `InvoiceDueDate`
  FORMAT(AP1.Invoices.PaymentDate,              -- 6. formatting column
    'yyyy-MM-dd', 'en-gb')                      --    as `yyyy-MM-dd` date with
                                                --    culture `en-gb` using
    AS PaymentDate,                             --    alias `PaymentDate`
  AP1.Terms.TermsDescription,
  AP1.Terms.TermsDueDays
FROM AP1.Invoices                               -- 7. from table `AP1.Invoices`
INNER JOIN AP1.Terms                            --    using `INNER JOIN` to
                                                --    retrieve all shared data
  ON AP1.Invoices.TermsID = AP1.Terms.TermsID   --    connecting both tables on
                                                --    shared field `TermsID`
WHERE (                                         -- 8. where the value of
    AP1.Invoices.PaymentTotal > (               --    `PaymentTotal` is larger
                                                --    than (`>`) the single
      SELECT AVG(PaymentTotal)                  --    value of sub-query
```

```sql
      FROM AP1.Invoices                                 --        `(SELECT
                                                        --         AVG(PaymentTotal)
                                                        --         FROM AP1.Invoices)`
                                                        --    that returns 1879.7413
      )                                                 --    8.1. sub-query always in
    )                                                   --         parenthesis, just
                                                        --         like in algebra
                                                        --    8.2. no need for
                                                        --         `ORDER BY` since
                                                        --         aggregate function
                                                        --         `AVG()` affects only
                                                        --         one column and it
                                                        --         does not affect the
                                                        --         main query
  AND AP1.Invoices.PaymentDate IS NOT NULL;             -- 9. and [where] value of
                                                        --    `PaymentDate` is not null
                                                        --    (must have a value)


/* ***************************************************************************
    At this point, we make a view in schema `lab06` in database `labs`
    excluding the `ORDER BY` clause as it cannot be used when creating views.
 *************************************************************************** */

CREATE SCHEMA lab06;                                    -- 1. creating schema `labs06`

CREATE VIEW lab06.InvoicesTermsVW                       -- 2. creating view
AS                                                      --    `InvoicesTermsVW` in
SELECT SF25SQL6172025.AP1.Invoices.InvoiceID,           --   database `labs` hence
  SF25SQL6172025.AP1.Invoices.VendorID,                 --   calling database
  SF25SQL6172025.AP1.Invoices.InvoiceNumber,            --   `SF25SQL6172025` for each
  FORMAT(SF25SQL6172025.AP1.Invoices.InvoiceDate,       --   table
    'yyyy-MM-dd', 'en-gb') AS InvoiceDate,
  FORMAT(SF25SQL6172025.AP1.Invoices.InvoiceTotal,
    'c', 'en-gb') AS InvoiceTotal,
  FORMAT(SF25SQL6172025.AP1.Invoices.PaymentTotal,
    'c', 'en-gb') AS PaymentTotal,
  FORMAT(SF25SQL6172025.AP1.Invoices.CreditTotal,
    'c', 'en-gb') AS CreditTotal,
  SF25SQL6172025.AP1.Invoices.TermsID,
  FORMAT(SF25SQL6172025.AP1.Invoices.InvoiceDueDate,
    'yyyy-MM-dd', 'en-gb') AS InvoiceDueDate,
  FORMAT(SF25SQL6172025.AP1.Invoices.PaymentDate,
    'yyyy-MM-dd', 'en-gb') AS PaymentDate,
  SF25SQL6172025.AP1.Terms.TermsDescription,
  SF25SQL6172025.AP1.Terms.TermsDueDays
FROM SF25SQL6172025.AP1.Invoices
INNER JOIN SF25SQL6172025.AP1.Terms
  ON SF25SQL6172025.AP1.Invoices.TermsID = SF25SQL6172025.AP1.Terms.TermsID
WHERE (
    SF25SQL6172025.AP1.Invoices.PaymentTotal > (
      SELECT AVG(PaymentTotal)
      FROM SF25SQL6172025.AP1.Invoices
      )
    )
  AND SF25SQL6172025.AP1.Invoices.PaymentDate IS NOT NULL;


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

                    `i` for `SF25SQL6172025.AP1.Invoices`
                    `t` for `SF25SQL6172025.AP1.Terms`
 *************************************************************************** */

CREATE VIEW lab06.InvoicesTermsVW
AS
SELECT i.InvoiceID,
```

```
    i.VendorID,
    i.InvoiceNumber,
    FORMAT(i.InvoiceDate, 'yyyy-MM-dd', 'en-gb') AS InvoiceDate,
    FORMAT(i.InvoiceTotal, 'c', 'en-gb') AS InvoiceTotal,
    FORMAT(i.PaymentTotal, 'c', 'en-gb') AS PaymentTotal,
    FORMAT(i.CreditTotal, 'c', 'en-gb') AS CreditTotal,
    i.TermsID,
    FORMAT(i.InvoiceDueDate, 'yyyy-MM-dd', 'en-gb') AS InvoiceDueDate,
    FORMAT(i.PaymentDate, 'yyyy-MM-dd', 'en-gb') AS PaymentDate,
    t.TermsDescription,
    t.TermsDueDays
FROM SF25SQL6172025.AP1.Invoices AS i
INNER JOIN SF25SQL6172025.AP1.Terms AS t
  ON i.TermsID = t.TermsID
WHERE (
    i.PaymentTotal > (
      SELECT AVG(PaymentTotal)                  -- no need for a table alias as
      FROM SF25SQL6172025.AP1.Invoices          -- the table is called once in
      )                                         -- the sub-query
    )
  AND i.PaymentDate IS NOT NULL;


/* ***************************************************************************
 3. As a quick review, SQL is the language to interact with a relational
    database.
    * to request data (`SELECT`) from database objects like databases,
      schemata, tables and views
    * to create (`CREATE`) where to store data, database objects like
      databases, schemata, tables including columns, etc.`
    * to modify (`ALTER`) database objects
    * to delete (`DROP`) database objects, automatic `COMMIT` in SQL Server
      hence no `ROLLBACK` (no way to rescue the data or objects)
    * and to manipulate data either affecting the data or not (showing data).

                    CREATE obj_type object_name
                    [other_code]

                    DROP obj_type object_name
                    [other_code]

                    ALTER obj_type object_name
                    ALTER|ADD|DROP obj_type obj_name data_type [other_code]

                    INSERT INTO table_name
                    VALUES
                      (
                        field1 datatype1,
                        field2 datatype2
                        ...
                      )

                    DELETE FROM table_name
                    [other_code]

                    TRUNCATE TABLE table_name

                    UPDATE table_name
                    SET field = new_value

    We use SQL to return data to any person or program that needs data.

    We can use functions to change the output of data as well as the data
    itself, which we will see later in the course.

 4. Although using a custom format like `yyyy-MM-dd` overrides the culture
    (`en-us`) and there is no longer need to include this culture, it is a good
    idea to include it as good practice.
 *************************************************************************** */
```

```
346
347  SELECT FORMAT(InvoiceTotal, 'yyyy-MM-dd')       -- no culture (`en-us`) needed
348  FROM AP1.Invoices;                              -- because of the custom format
349
350  SELECT FORMAT(InvoiceTotal,                     -- but good practice to include
351    'yyyy-MM-dd', 'en-us')                        -- the culture (`en-us`) even
352  FROM AP1.Invoices;                              -- when overridden by custom
353                                                  -- a format like `yyyy-MM-dd`
354
355
356  /* ************************************************************************
357   5. As mentioned several times, `FORMAT()` changes numeric values to strings.
358      We can also use `CONVERT()` to change ``an expression from a data type to
359      another data type`` -- in other words, numeric values to strings or vice
360      versa (https://techonthenet.com/sql_server/functions/convert.php).
361
362                      CONVERT(new_data_type, column)
363
364      `CONVERT()` does not change the currency sign or adds commas to divide
365      thousands or millions as `FORMAT()` does.
366
367      5.01. In the example below, we change the data type of `InvoiceTotal` to
368            VARCHAR(50) -- an allocation in RAM to hold a variable character
369            value with a maximum size of fifty (50) characters.
370   ************************************************************************ */
371
372  SELECT CONVERT(VARCHAR(50), InvoiceTotal)       -- changing data type of column
373    AS InvoiceTotal                               -- `InvoiceTotal` (`FLOAT`) to
374  FROM AP1.Invoices;                              -- `VARCHAR(50)`
375
376
377  /* ************************************************************************
378      5.02. In the example below, we use `CONVERT()` to return the value of
379            `AP1.Invoices.InvoiceTotal` as a dollar amount concatenating the
380            dollar sign (`$`) at the beginning.
381   ************************************************************************ */
382
383  SELECT CONCAT (
384          '$',                                    -- concatenating `$` to the
385          CONVERT(VARCHAR(50), InvoiceTotal)      -- output of data conversion
386          ) AS InvoiceTotal                       --    `CONVERT(VARCHAR(50),
387  FROM AP1.Invoices;                              --    InvoiceTotal)`
388
389
390  /* ************************************************************************
391            We could also use `CONVERT()` to return the value of
392            `AP1.Invoices.InvoiceTotal` as a dollar amount with `USD` rather than
393            the dollar sign (`$`).
394   ************************************************************************ */
395
396  SELECT CONCAT (
397          'USD ',                                 -- concatenating `USD ` to the
398          CONVERT(VARCHAR(50), InvoiceTotal)      -- output of data conversion
399          ) AS InvoiceTotal                       --    `CONVERT(VARCHAR(50),
400  FROM AP1.Invoices;                              --    InvoiceTotal)`
401
402
403  /* ************************************************************************
404            Of course, if you are ``dressing up`` a numeric value like
405            `AP1.Invoices.InvoiceTotal` as currency, it is better to just use
406            `FORMAT()` to keep your code simple.
407   ************************************************************************ */
408
409  SELECT FORMAT(InvoiceTotal, 'c', 'en-us') AS InvoiceTotal
410  FROM AP1.Invoices;
411
412
413  /* ************************************************************************
414      5.03. In the example below, we use `CONVERT()` to change the data type of
```

```
           `AP1.Invoices.InvoiceID` and `AP1.Invoices.VendorID` from FLOAT to
           `VARCHAR(50)` before concatenating these values to a string.
 ************************************************************************** */

SELECT CONCAT (                                 -- 1. concatenating string
                                                --    values of
    'Invoice ',                                 --    1.1. `Invoice `
                                                --         (hard-coded),
    CONVERT(VARCHAR(3), InvoiceID),             --    1.2. the conversion of
                                                --         `InvoiceID` to
                                                --         `VARCHAR(3)`,
    ' from vendor ',                            --    1.3. ` from vendor `
                                                --         (hard-coded) and
    CONVERT(VARCHAR(3), VendorID)               --    1.4. the conversion of
                                                --         `VendorID` to
                                                --         `VARCHAR(3)`
    ) AS InvoiceVendor                          --    1.5. using alias
                                                --         `InvoiceVendor`
FROM AP1.Invoices;


/* **************************************************************************
 6. We use the `WHERE` (https://techonthenet.com/sql_server/where.php)
    clause to ``filter the results from a SELECT, INSERT, UPDATE, or DELETE
    statement.``

                    SELECT table1.field1, table1.field2 ...
                      table2.field1, table2.field2 ...
                    FROM table1
                      INNER|LEFT|RIGHT JOIN table2
                      ON table1.shared_field1 = table2.shared_field1
                        AND table1.shared_field2 = table2.shared_field2
                      ...
                    WHERE condition1
                      AND|OR condition2
                      ...

    6.01. We use conditions in order to filter data.

          AND       to test for two or more conditions
                    https://techonthenet.com/sql_server/and.php

          OR        to test multiple conditions where records are returned when
                    any one of the conditions are met
                    https://techonthenet.com/sql_server/or.php

    6.02. We use operators to compare values.

          =         equal to
                    https://techonthenet.com/sql_server/comparison_operators.php

          <>        not equal to
                    https://techonthenet.com/sql_server/comparison_operators.php

          !=        not equal to
                    https://techonthenet.com/sql_server/comparison_operators.php

          <         less than
                    https://techonthenet.com/sql_server/comparison_operators.php

          >         greater than
                    https://techonthenet.com/sql_server/comparison_operators.php

          <=        less than or equal to
                    https://techonthenet.com/sql_server/comparison_operators.php

          >=        greater than or equal to
                    https://techonthenet.com/sql_server/comparison_operators.php
```

```
            !>          not greater than (same as <=)
                        https://techonthenet.com/sql_server/comparison_operators.php

            !<          not less than (same as >=)
                        https://techonthenet.com/sql_server/comparison_operators.php

            LIKE        allows wild cards to be used in the WHERE clause of a
                        SELECT, INSERT, UPDATE, or DELETE statement [allowing] you
                        to perform pattern matching
                        https://techonthenet.com/sql_server/like.php

            IN          to help reduce the need to use multiple OR conditions in a
                        SELECT, INSERT, UPDATE, or DELETE statement
                        https://techonthenet.com/sql_server/in.php

            BETWEEN     used to retrieve values within a range in a SELECT, INSERT,
                        UPDATE, or DELETE statement
                        https://techonthenet.com/sql_server/between.php

            IS NULL     condition... used to test for a NULL no value
                        https://techonthenet.com/sql_server/is_null.php

            NOT         to negate a condition in a SELECT, INSERT, UPDATE, or
                        DELETE statement
                        https://techonthenet.com/sql_server/not.php

                        * NOT LIKE
                        * NOT IN
                        * NOT BETWEEN
                        * IS NOT NULL
                          https://techonthenet.com/sql_server/is_not_null.php

        6.03. In the example, below, we retrieve all values from table
              `AP1.Vendors` where `VendorState` is equal to `CA` and `VendorCity`
              could either be `Fresno` or `Sacramento`.

              Use parenthesis for SQL (regardless of vendor/distribution) to
              process the inner condition first

                        (
                          VendorCity = 'Fresno'
                          OR VendorCity = 'Sacramento'
                        )

              before the outer condition.
     ********************************************************************** */

    SELECT *
    FROM AP1.Vendors
    WHERE VendorState = 'CA'                        -- 1. inner criterion that must
                                                    --    be true (satisfied)
      AND (                                         -- 2. outer criterion that must
                                                    --    be true composed of two
                                                    --    sections where either
                                                    --    could be true (satisfied)
        VendorCity = 'Fresno'                       --    3.1. first criteria that
                                                    --         could be met
        OR VendorCity = 'Sacramento'                --    3.2. second criteria that
                                                    --         could be met
        );


    /* **********************************************************************
        6.04. In the example below, we retrieve all values from table `AP1.Vendors`
              where `VendorState` is not (`<>` or `!=`) `NY`.
     ********************************************************************** */

    SELECT *
    FROM AP1.Vendors                                -- can also be written as
```

```
553   WHERE VendorState <> 'NY';                          -- `VendorState != `NY``
554
555
556   /* ************************************************************************
557       6.05. In the example below, we retrieve all values from table `AP1.Vendors`
558             where `VendorState` is either `DC` or `IA`.
559    ************************************************************************ */
560
561   SELECT *
562   FROM AP1.Vendors
563   WHERE VendorState = 'DC'                            -- checking if either criterion
564     OR VendorState = 'IA';                            -- is true
565
566
567   /* ************************************************************************
568       6.06. In the example below, we retrieve all values from table `AP1.Vendors`
569             where `VendorAddress2` is NULL (no-value) using `NOT` as it negates
570             operators `LIKE` as `NOT LIKE`, `IN` as `NOT IN`, `BETWEEN` as
571             `NOT BETWEEN` and `IS NULL` as `IS NOT NULL`.
572    ************************************************************************ */
573
574   SELECT *
575   FROM AP1.Vendors
576   WHERE VendorAddress2 IS NULL;                       -- asking for no-value
577
578
579   /* ************************************************************************
580       6.07. In the example below, we retrieve all values from table `AP1.Vendors`
581             where `VendorAddress2` is not NULL (not a no-value).  Refer to
582             https://techonthenet.com/sql_server/is_not_null.php for more
583             information.
584    ************************************************************************ */
585
586   SELECT *
587   FROM AP1.Vendors
588   WHERE VendorAddress2 IS NOT NULL;                   -- asking for not `NOT NULL`
589                                                       -- (no no-value)
590
591
592   /* ************************************************************************
593       6.08. In the example below, we rewrite #6.3 in a cleaner fashion to
594             retrieve all values from table `AP1.Vendors` where `VendorState` is
595             equal to `CA` and `VendorCity` could either be `Fresno` or
596             `Sacramento`.  We use operator `IN`
597             (https://techonthenet.com/sql_server/in.php) to specify the list of
598             values that can be true for `VendorCity`.
599    ************************************************************************ */
600
601   SELECT *
602   FROM AP1.Vendors
603   WHERE VendorState = 'CA'                            -- 1. first condition as in
604                                                       --    original example
605     AND VendorCity IN (                               -- 2. second condition using
606       'Fresno',                                       --    `IN` to list all possible
607       'Sacramento'                                    --    values that can be true
608       );                                              --    (satisfied)
609
610
611   /* ************************************************************************
612       6.09. In the example below, we retrieve all values from table `AP1.Vendors`
613             where `VendorState` could either be `CA` or `NJ` and `VendorCity`
614             could either be `Fresno` or `Sacramento`.
615
616             This query looks for the combination of
617
618                       `CA` and `Fresno`     (true)
619                       `CA` and `Sacramento` (true)
620
621             as well as
```

```
                    `NJ` and `Fresno`     (false)
                    `NJ` and `Sacramento` (false)

          The query only returns only the first set of values since we do not
          have any records where `VendorCity` is `NJ` and VendorCity` is either
          `Fresno` or `Sacramento`.
 ************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE (
    VendorState IN (                          -- 1. first condition using
      'CA',                                   --    `IN` to list all possible
      'NJ'                                    --    values that can be true
      )                                       --    (satisfied)
    AND VendorCity IN (                       -- 2. second condition using
      'Fresno',                               --    `IN` to list all possible
      'Sacramento'                            --    values that can be true
      )                                       --    (satisfied)
    )
ORDER BY VendorState,
  VendorCity;


/* ************************************************************************
    6.10. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorState` could either be `CA` and `VendorCity` could
          either be `Fresno` or `Sacramento` as one condition or `VendorState`
          is `NJ` as another condition.
 ************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE (                                       -- 1. first condition where
    VendorState IN ('CA')                     --    `VendorState` could be
    AND VendorCity IN (                       --    `CA` and `VendorCity`
      'Fresno',                               --    could either be `Fresno`
      'Sacramento'                            --    or `Sacramento`
      )                                       --    looking the combinations
    )                                         --    of `CA` and `Fresno` or
                                              --    `CA` and `Sacramento`
  OR VendorState IN ('NJ')                    -- 2. second condition starting
                                              --    with `OR` to specify that
                                              --    `VendorState` could also
                                              --    be `NJ`
ORDER BY VendorState,                         -- 3. ordering results first by
  VendorCity;                                 --    `VendorState` and then by
                                              --    `VendorCity`


/* ************************************************************************
    6.11. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorName` has as a value starting with `am` (not case
          sensitive) using wild card `%` to represent any character or group of
          after `am`.
 ************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorName LIKE 'am%';                  -- returns values
                                              -- `American Booksellers Assoc`
                                              -- and `American Express`


/* ************************************************************************
    6.12. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorName` has as a value with pattern `data` (not case
          sensitive) using wild card `%` before and after the given string.
```

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

SELECT *
FROM AP1.Vendors
WHERE VendorName LIKE '%data%';                 -- returns various values like
                                                -- `Expedata Inc`,
                                                -- `California Data Marketing`
                                                -- and `Quality Education Data`


/* **********************************************************************
    6.13. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorPhone` has as a value starting with `800` (string, not a
          numeric value).
 ********************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorPhone LIKE '800%';


/* **********************************************************************
    6.14. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorPhone` has as a value NOT starting with `800`.
 ********************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorPhone NOT LIKE '800%';


/* **********************************************************************
    6.15. In the example below, we retrieve all values from table
          `AP1.Invoices` where `InvoiceDueDate` has values within the range of
          two dates -- `2012-01-01` and `2012-01-30` (dates always in single
          quotes).
 ********************************************************************** */

SELECT *
FROM AP1.Invoices
WHERE InvoiceDueDate BETWEEN '2012-01-01'       -- range between `2012-01-01`
    AND '2012-01-30';                           -- and `2012-01-30`


/* **********************************************************************
    6.16. In the example below, we retrieve all values from table `AP1.Vendors`
          where InvoiceTotal has values within 100 and 1000.  Then we organize
          the results in descending order using an `ORDER BY` clause
          (https://techonthenet.com/sql/order_by.php).

          The default option for `ORDER BY` is `ASC` (ascending), which can be
          omitted.

          The opposite option for `ORDER BY` is `DESC` (descending), which
          needs to be specified for each field.
 ********************************************************************** */

SELECT *
FROM AP1.Invoices
WHERE InvoiceTotal BETWEEN 100                  -- range between 100 and 1000
    AND 1000
ORDER BY InvoiceTotal DESC,                     -- organizing results
                                                -- 1. first by `InvoiceTotal`
                                                --    in descending order
                                                --    (`DESC`)
  PaymentTotal DESC,                            -- 2. then by `PaymentTotal` in
                                                --    descending order
                                                --    (`DESC`)
  TermsID DESC;                                 -- 3. then finally by `TermsID`
```

```
                                                    --    also in descending order
                                                    --    (`DESC`)


/* ********************************************************************************
 7. As we have mentioned several times, when calling multiple tables, we need
    to `JOIN` them (https://techonthenet.com/sql_server/joins.php).

    7.01. `INNER JOIN` returns ``all rows from multiple tables where the join
          condition is met.``

          In the example below, we retrieve all records shared in tables
          `AP1.Invoices` and `AP1.Invoices`.
 ******************************************************************************** */

SELECT *
FROM AP1.Vendors
INNER JOIN AP1.Invoices
  ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;


/* ********************************************************************************
    7.02. `LEFT JOIN` returns ``all rows from the LEFT-hand table specified in
          the ON condition and only those rows from the other table where the
          joined fields are equal (join conditions met).``

          In the example below, we retrieve all records in `AP1.Vendors` (left
          table) and any records in `AP1.Invoices` (if any in the right table).
 ******************************************************************************** */

SELECT *
FROM AP1.Vendors                                    -- retrieves all records from
LEFT JOIN AP1.Invoices                              -- the left table/dataset
  ON AP1.Vendors.VendorID=AP1.Invoices.VendorID;-- (first table/dataset
                                                    -- called in the statement,
                                                    -- `AP1.Vendors`) and related
                                                    -- records from the right
                                                    -- table/dataset (second
                                                    -- table/dataset called in the
                                                    -- statement, `AP1.Invoices`);
                                                    -- returns 202 records


/* ********************************************************************************
          In the example below, we retrieve all records in `AP1.Invoices` (left
          table) and any records in `AP1.Vendors` (if any in the right table).
 ******************************************************************************** */

SELECT *
FROM AP1.Invoices                                   -- retrieves all records from
LEFT JOIN AP1.Vendors                               -- the left table/dataset
  ON AP1.Vendors.VendorID=AP1.Invoices.VendorID;-- (first table/dataset called
                                                    -- in the statement,
                                                    -- `AP1.Invoices`) and related
                                                    -- records from the right
                                                    -- table/dataset (second
                                                    -- table/dataset called in the
                                                    -- statement, `AP1.Vendors`)


/* ********************************************************************************
    7.03. `RIGHT JOIN` returns ``all rows from the RIGHT-hand table specified in
          the ON condition and only those rows from the other table where the
          joined fields are equal (join condition is met).``

          In the example below, we retrieve all records in `AP1.Invoices`
          (right table) and any records in `AP1.Vendors` (if any in the left
          table).
 ******************************************************************************** */
```

```
829
830   SELECT *
831   FROM AP1.Vendors                                    -- retrieves all records from
832   RIGHT JOIN AP1.Invoices                             -- the right table/dataset
833     ON AP1.Invoices.VendorID=AP1.Vendors.VendorID;-- (second table/dataset called
834                                                       -- in the statement,
835                                                       -- `AP1.Invoices`) and related
836                                                       -- records from the left
837                                                       -- table/dataset (first
838                                                       -- table/dataset called in the
839                                                       -- statement, `AP1.Invoices`)
840
841
842   /* ************************************************************************
843       7.04. `FULL JOIN` returns ``all rows from the LEFT-hand table and RIGHT-
844             hand table with nulls in place where the join condition is not met.``
845             Note that depending on the size of the tables, this query might make
846             the server run slowly or crash it.
847
848             In the example below, we retrieve all records in `AP1.Invoices` (left
849             table) and all records in `AP1.Vendors` (right table).
850    ************************************************************************ */
851
852   SELECT *
853   FROM AP1.Invoices                                   -- retrieves all records from
854   FULL JOIN AP1.Vendors                               -- the left table/dataset
855     ON AP1.Vendors.VendorID=AP1.Invoices.VendorID;-- (first table/dataset called
856                                                       -- in the statement,
857                                                       -- `AP1.Vendors`) and all
858                                                       -- records from the right
859                                                       -- table/dataset (second
860                                                       -- table/dataset called in the
861                                                       -- statement, `AP1.Invoices`)
862
863
864   /* ************************************************************************
865    8. Now that we have reviewed most of the material so far, we start views.
866
867           ``In a database management system, a view is a way of portraying
868           information in the database.  This can be done by arranging the data
869           items in a specific order, by highlighting certain items, or by
870           showing only certain items.  For any database, there are a number of
871           possible views that may be specified.  Databases with many items tend
872           to have more possible views than databases with few items.  Often
873           thought of as a virtual table, the view doesn't actually store
874           information itself, but just pulls it out of one or more existing
875           tables.  Although impermanent, a view may be accessed repeatedly by
876           storing its criteria in a query.``
877           http://searchsqlserver.techtarget.com/definition/view
878
879                     CREATE VIEW view_name AS
880                       SELECT columns
881                       FROM tables
882                       [WHERE conditions];
883
884       8.01. In the example below, we modify table `AP1.Invoices` adding column
885             `CustomerID` in order to establish a relation between this table and
886             `AP2.Customers`.
887    ************************************************************************ */
888
889   ALTER TABLE AP1.Invoices
890   ADD CustomerID INT NULL;
891
892   UPDATE AP1.Invoices
893   SET CustomerID = 1
894   WHERE VendorID = 34;
895
896   UPDATE AP1.Invoices
897   SET CustomerID = 2
```

```
WHERE VendorID = 37;

UPDATE AP1.Invoices
SET CustomerID = 3
WHERE VendorID = 89;


/* ***************************************************************************
    8.02. Now that relationship has been created, we can now query tables
          `AP1.Invoices` and `AP2.Customers` (each tables in a different
          databases).
 *************************************************************************** */

SELECT DISTINCT AP1.Invoices.InvoiceID,
  AP1.Invoices.VendorID,
  AP1.Invoices.InvoiceNumber,
  FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
  FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
  FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
  FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
  AP1.Invoices.TermsID,
  FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
  FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
  AP1.Invoices.CustomerID,
  AP2.Customers.LastName,
  AP2.Customers.FirstName,
  AP2.Customers.Address,
  AP2.Customers.City,
  AP2.Customers.STATE,
  AP2.Customers.ZipCode,
  AP2.Customers.Email
FROM AP1.Invoices
INNER JOIN AP2.Customers
  ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
ORDER BY AP1.Invoices.VendorID;


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

                    `i` for `AP1.Invoices`
                    `c` for `AP2.Customers`
 *************************************************************************** */

SELECT DISTINCT i.InvoiceID,
  i.VendorID,
  i.InvoiceNumber,
  FORMAT(i.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
  FORMAT(i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
  FORMAT(i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
  FORMAT(i.CreditTotal, 'c', 'en-us') AS CreditTotal,
  i.TermsID,
  FORMAT(i.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
  FORMAT(i.PaymentDate, 'd', 'en-us') AS PaymentDate,
  i.CustomerID,
  c.LastName,
  c.FirstName,
  c.Address,
  c.City,
  c.STATE,
  c.ZipCode,
  c.Email
FROM AP1.Invoices AS i
INNER JOIN AP2.Customers AS c
  ON i.CustomerID = c.CustomerID
ORDER BY i.VendorID;
```

```
/* ************************************************************************
    8.03. In the example below, we can create a view using the query in the
          example above using tables `AP1.Invoices` and `AP2.Customers`
          without `ORDER BY`, which would return an error when creating the
          view.

          Tables and views cannot share names since both data objects are of
          the same hierarchy.

          We can query, alter and/or drop a view just like a table.

          In most relational databases, we cannot update data using a view
          since this action only take place in tables.

          In SQL Server (T-SQL), we can update data from the base table.

                    ``Requires UPDATE, INSERT, or DELETE permissions on the
                    target table, depending on the action being performed.``
                    https://msdn.microsoft.com/en-us/library/ms180800.aspx
 ************************************************************************ */

CREATE VIEW AP1.InvoicesCustomersVW
AS
(
    SELECT DISTINCT AP1.Invoices.InvoiceID,
      AP1.Invoices.VendorID,
      AP1.Invoices.InvoiceNumber,
      FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
      FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
      FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
      FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
      AP1.Invoices.TermsID,
      FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
      FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
      AP1.Invoices.CustomerID,
      AP2.Customers.LastName,
      AP2.Customers.FirstName,
      AP2.Customers.Address,
      AP2.Customers.City,
      AP2.Customers.STATE,
      AP2.Customers.ZipCode,
      AP2.Customers.Email
    FROM AP1.Invoices
    INNER JOIN AP2.Customers
      ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
    );


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

                    `i` for `AP1.Invoices`
                    `c` for `AP2.Customers`
 ************************************************************************ */

CREATE VIEW AP1.InvoicesCustomersVW
AS
(
    SELECT DISTINCT i.InvoiceID,
      i.VendorID,
      i.InvoiceNumber,
      FORMAT(i.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
      FORMAT(i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
      FORMAT(i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
      FORMAT(i.CreditTotal, 'c', 'en-us') AS CreditTotal,
      i.TermsID,
      FORMAT(i.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
      FORMAT(i.PaymentDate, 'd', 'en-us') AS PaymentDate,
```

```
1036        i.CustomerID,
1037        c.LastName,
1038        c.FirstName,
1039        c.Address,
1040        c.City,
1041        c.STATE,
1042        c.ZipCode,
1043        c.Email
1044      FROM AP1.Invoices AS i
1045      INNER JOIN AP2.Customers AS c
1046        ON i.CustomerID = c.CustomerID
1047      );
1048
1049
1050  /* ************************************************************************
1051      8.04. We can modify a view simply changing `CREATE` for `ALTER`.
1052   ************************************************************************ */
1053
1054  ALTER VIEW AP1.InvoicesCustomersVW
1055  AS
1056  (
1057      SELECT DISTINCT AP1.Invoices.InvoiceID,
1058        AP1.Invoices.VendorID,
1059        AP1.Invoices.InvoiceNumber,
1060        FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us')
1061          AS InvoiceDate,
1062        FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1063        FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1064        FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
1065        AP1.Invoices.TermsID,
1066        FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
1067        FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
1068        AP1.Invoices.CustomerID,
1069        AP2.Customers.LastName,
1070        AP2.Customers.FirstName,
1071        AP2.Customers.Address,
1072        AP2.Customers.City,
1073        AP2.Customers.STATE,
1074        AP2.Customers.ZipCode,
1075        AP2.Customers.Email,
1076        GETDATE() AS SystemDate                    -- change from previous query
1077      FROM AP1.Invoices
1078      INNER JOIN AP2.Customers
1079        ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
1080      );
1081
1082
1083  /* ************************************************************************
1084            As with previous examples, we can use an alias for each table, which
1085            in this case, allows us to present neater code.
1086
1087                      `i` for `AP1.Invoices`
1088                      `c` for `AP2.Customers`
1089   ************************************************************************ */
1090
1091  ALTER VIEW AP1.InvoicesCustomersVW
1092  AS
1093  (
1094      SELECT DISTINCT i.InvoiceID,
1095        i.VendorID,
1096        i.InvoiceNumber,
1097        FORMAT(i.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
1098        FORMAT(i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1099        FORMAT(i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1100        FORMAT(i.CreditTotal, 'c', 'en-us') AS CreditTotal,
1101        i.TermsID,
1102        FORMAT(i.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
1103        FORMAT(i.PaymentDate, 'd', 'en-us') AS PaymentDate,
1104        i.CustomerID,
```

```
        c.LastName,
        c.FirstName,
        c.Address,
        c.City,
        c.STATE,
        c.ZipCode,
        c.Email,
        GETDATE() AS SystemDate                    -- no table alias since value
                                                   -- is returned by function
                                                   -- `GETDATE()`, which calls the
                                                   -- system DATETIME
      FROM AP1.Invoices AS i
      INNER JOIN AP2.Customers AS c
        ON i.CustomerID = c.CustomerID
      );


/* ************************************************************************
    8.05. In the example below, we create view `AP1.InvoicesVW` only from table
          `AP1.Invoices` formatting the date and currency fields accordingly.
          This way we do not need to format the columns again and again every
          time we need to call them.
 ************************************************************************ */

CREATE VIEW AP1.InvoicesVW
AS
(
    SELECT DISTINCT InvoiceID,
      VendorID,
      InvoiceNumber,
      FORMAT(InvoiceDate, 'd', 'en-us') AS InvoiceDate,
      FORMAT(InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
      FORMAT(PaymentTotal, 'c', 'en-us') AS PaymentTotal,
      FORMAT(CreditTotal, 'c', 'en-us') AS CreditTotal,
      TermsID,
      FORMAT(InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
      FORMAT(PaymentDate, 'd', 'en-us') AS PaymentDate,
      CustomerID
    FROM AP1.Invoices                              -- no need for a table alias as
    );                                             -- all values come from the one
                                                   -- table (`AP1.Invoices`)


/* ************************************************************************
    8.06. In the example below, we create view `AP1.InvoicesVendorsVW` from
          tables `AP1.Invoices` and `AP1.Vendors`.

          Unless we indicate in which database to store the view, it would most
          likely be in the same database where the previous view was stored
          (`AP2`).
 ************************************************************************ */

CREATE VIEW AP1.InvoicesVendorsVW
AS
(
    SELECT DISTINCT AP1.Invoices.InvoiceID,
      AP1.Invoices.VendorID,
      AP1.Invoices.InvoiceNumber,
      AP1.Invoices.InvoiceDate,
      AP1.Invoices.InvoiceTotal,
      AP1.Invoices.PaymentTotal,
      AP1.Invoices.CreditTotal,
      AP1.Invoices.TermsID,
      AP1.Invoices.InvoiceDueDate,
      AP1.Invoices.PaymentDate,
      AP1.Vendors.VendorName,
      CASE
        WHEN AP1.Vendors.VendorAddress2 IS NOT NULL
          THEN CONCAT (
```

```
1174                AP1.Vendors.VendorAddress1,
1175                ' ',
1176                AP1.Vendors.VendorAddress2
1177                )
1178          WHEN AP1.Vendors.VendorAddress1 IS NULL
1179            AND AP1.Vendors.VendorAddress2 IS NULL
1180            THEN 'No Address'
1181          ELSE AP1.Vendors.VendorAddress1
1182          END AS VendorAddress,
1183        AP1.Vendors.VendorCity,
1184        AP1.Vendors.VendorState,
1185        AP1.Vendors.VendorZipCode,
1186        AP1.Vendors.DefaultAccountNo
1187      FROM AP1.Invoices
1188      LEFT JOIN AP1.Vendors
1189        ON AP1.Invoices.VendorID = AP1.Vendors.VendorID
1190      );
1191
1192
1193  /* ***************************************************************************
1194            As with previous examples, we can use an alias for each table, which
1195            in this case, allows us to present neater code.
1196
1197                      `i` for `AP1.Invoices`
1198                      `v` for `AP1.Vendors`
1199   *************************************************************************** */
1200
1201  CREATE VIEW AP1.InvoicesVendorsVW
1202  AS
1203  (
1204      SELECT DISTINCT i.InvoiceID,
1205        i.VendorID,
1206        i.InvoiceNumber,
1207        i.InvoiceDate,
1208        i.InvoiceTotal,
1209        i.PaymentTotal,
1210        i.CreditTotal,
1211        i.TermsID,
1212        i.InvoiceDueDate,
1213        i.PaymentDate,
1214        v.VendorName,
1215        CASE
1216          WHEN v.VendorAddress2 IS NOT NULL
1217            THEN CONCAT (
1218                v.VendorAddress1,
1219                ' ',
1220                v.VendorAddress2
1221                )
1222          WHEN v.VendorAddress1 IS NULL
1223            AND v.VendorAddress2 IS NULL
1224            THEN 'No Address'
1225          ELSE v.VendorAddress1
1226          END AS VendorAddress,
1227        v.VendorCity,
1228        v.VendorState,
1229        v.VendorZipCode,
1230        v.DefaultAccountNo
1231      FROM AP1.Invoices AS i
1232      LEFT JOIN AP1.Vendors AS v
1233        ON i.VendorID = v.VendorID
1234      );
1235
1236
1237  /* ***************************************************************************
1238      8.07. In the example below, we create view
1239            `AP1.Invoices_Customers_Vendors_VW` from views (like we would do with
1240            tables) `AP1.InvoicesCustomersVW` and `AP1.InvoicesVendorsVW` in
1241            schema `AP1`, which we must specify.
1242
```

```
1243            As mentioned, unless we indicate in which schema to store the new
1244            view, the new object would be saved in default schema `dbo`.  In this
1245            case, we do not need to call the schema (`dbo`), but it is always a
1246            good idea -- good practice.  Note that if we need to call the
1247            database name, we must call the schema (`db1.dbo.table1`).
1248     ********************************************************************** */
1249
1250   CREATE VIEW AP1.Invoices_Customers_Vendors_VW
1251   AS
1252   (
1253       SELECT DISTINCT AP1.InvoicesCustomersVW.InvoiceID,
1254         AP1.InvoicesCustomersVW.VendorID,
1255         AP1.InvoicesCustomersVW.InvoiceNumber,
1256         AP1.InvoicesCustomersVW.InvoiceDate,
1257         AP1.InvoicesCustomersVW.InvoiceTotal,
1258         AP1.InvoicesCustomersVW.PaymentTotal,
1259         AP1.InvoicesCustomersVW.CreditTotal,
1260         AP1.InvoicesCustomersVW.TermsID,
1261         AP1.InvoicesCustomersVW.InvoiceDueDate,
1262         AP1.InvoicesCustomersVW.PaymentDate,
1263         AP1.InvoicesCustomersVW.CustomerID,
1264         AP1.InvoicesCustomersVW.LastName,
1265         AP1.InvoicesCustomersVW.FirstName,
1266         AP1.InvoicesCustomersVW.Address,
1267         AP1.InvoicesCustomersVW.City,
1268         AP1.InvoicesCustomersVW.STATE,
1269         AP1.InvoicesCustomersVW.ZipCode,
1270         AP1.InvoicesCustomersVW.Email,
1271         AP1.InvoicesVendorsVW.VendorName,
1272         AP1.InvoicesVendorsVW.VendorAddress,
1273         AP1.InvoicesVendorsVW.VendorCity,
1274         AP1.InvoicesVendorsVW.VendorState,
1275         AP1.InvoicesVendorsVW.VendorZipCode,
1276         AP1.InvoicesVendorsVW.DefaultAccountNo
1277       FROM AP1.InvoicesCustomersVW
1278       LEFT OUTER JOIN AP1.InvoicesVendorsVW
1279         ON AP1.InvoicesCustomersVW.VendorID = AP1.InvoicesVendorsVW.VendorID
1280       );
1281
1282
1283   /* **********************************************************************
1284            As with previous examples, we can use an alias for each table, which
1285            in this case, allows us to present neater code.
1286
1287                     `icv` for `AP1.InvoicesCustomersVW`
1288                     `ivv` for `AP1.InvoicesVendorsVW`
1289    ********************************************************************** */
1290
1291   CREATE VIEW AP1.Invoices_Customers_Vendors_VW
1292   AS
1293   (
1294       SELECT DISTINCT ivv.InvoiceID,
1295         ivv.VendorID,
1296         ivv.InvoiceNumber,
1297         ivv.InvoiceDate,
1298         ivv.InvoiceTotal,
1299         ivv.PaymentTotal,
1300         ivv.CreditTotal,
1301         ivv.TermsID,
1302         ivv.InvoiceDueDate,
1303         ivv.PaymentDate,
1304         ivv.CustomerID,
1305         ivv.LastName,
1306         ivv.FirstName,
1307         ivv.Address,
1308         ivv.City,
1309         ivv.STATE,
1310         ivv.ZipCode,
1311         ivv.Email,
```

```
        icv.VendorName,
        icv.VendorAddress,
        icv.VendorCity,
        icv.VendorState,
        icv.VendorZipCode,
        icv.DefaultAccountNo
      FROM AP1.InvoicesCustomersVW AS icv
      LEFT OUTER JOIN AP1.InvoicesVendorsVW AS ivv
        ON ivv.VendorID = icv.VendorID
      );


/* ************************************************************************
 9. Depending on the relational database management system (RDBMS) and even the
    product related to each RDBMS, the date format might vary.  In SQL Server,
    we can query data using format `YYYY/MM/DD` (including quotes) although the
    system returns format `YYYY-MM-DD` plus time in format `hh:mm:ss.nnnnnnn`.
    Refer to https://msdn.microsoft.com/en-us/library/bb630352.aspx and
    https://msdn.microsoft.com/en-us/library/bb677243.aspx for information on
    date and time respectively.

    9.01. The most common date functions are the following.

          DAY()     returns the day of the month (1 to 31) given a date value
                    https://techonthenet.com/sql_server/functions/day.php

          MONTH()   returns the month (1 to 12) given a date value
                    https://techonthenet.com/sql_server/functions/month.php

          YEAR()    returns a four-digit year (as a number) given a date value
                    https://techonthenet.com/sql_server/functions/year.php

          GETDATE() returns the current date and time
                    https://techonthenet.com/sql_server/functions/getdate.php
 ************************************************************************ */

SELECT DAY('2021/09/15') AS Day,                -- 1. returns `15` from
                                                --    `2021/09/15` without
                                                --    leading zeros (`d`)
  MONTH('2021/09/15') AS Month,                 -- 2. returns `9` from
                                                --    `2021/09/15` without
                                                --    leading zeros (`M`)
  YEAR('2021/09/15') AS Year;                   -- 3. returns `2021` from
                                                --    `2021/09/15 (`yyyy`)

SELECT GETDATE() AS CurrentDateTime;            -- returns
                                                -- `2021-09-15 20:20:34.053`
                                                -- from `GETDATE()` that calls
                                                -- system date and time

SELECT DAY(GETDATE()) AS Day,                   -- 1. returns `15` from system
                                                --    DATETIME without leading
                                                --    zeros (`d`)
  MONTH(GETDATE()) AS Month,                    -- 2. returns `9` from system
                                                --    DATETIME without leading
                                                --    zeros (`M`)
  YEAR(GETDATE()) AS Year;                      -- 3. returns `2021` from
                                                --    system DATETIME (`yyyy`)

SELECT FORMAT(GETDATE(), 'd', 'en-us')          -- returns system date and time
  AS FormattedCurrentDateTime;                  -- formatted as `9/15/2021`


/* ************************************************************************
    9.02. Instead of hard-coding the date in the example above (#3.1), we can
          use parameter `@date` in all instances that we need to pass the value
          returned by `GETDATE()`.

          We must declare each parameter with its proper data type.
```

```
1381
1382            We can then have to pass (`SET`) a value for each parameter.
1383     ************************************************************************ */
1384
1385   DECLARE @date DATETIME = GETDATE()                -- 1. declaring parameter
1386                                                     --    `@date` as DATETIME (the
1387                                                     --    proper data type) and
1388                                                     --    passing value of
1389                                                     --    `GETDATE()`
1390
1391   SELECT DAY(@date) AS Day,                         -- 2. returns `9` from system
1392                                                     --    DATETIME without leading
1393                                                     --    zeros (`d`)
1394     MONTH(@date) AS Month,                          -- 3. returns `15` from system
1395                                                     --    DATETIME without leading
1396                                                     --    zeros (`M`)
1397     YEAR(@date) AS Year;                            -- 4. returns `2021` from
1398                                                     --    system DATETIME (`yyyy`)
1399
1400
1401   /* ************************************************************************
1402       9.03. We can also use date function `GETDATE()` to calculate age in months,
1403             days and years.
1404
1405             The following example (9.03.01 to 9.03.14) is based on the answer
1406             found at http://stackoverflow.com/q/57599/, which is explained below
1407             in detail.
1408
1409             9.03.01. We declare variables `@start_date`, `@end_date` and
1410                      `@tmp_date` as data type DATETIME
1411                      (https://msdn.microsoft.com/en-us/library/ms187819.aspx).
1412
1413             9.03.02. It is good practice to use a second variable (in this case,
1414                      `@tmp_date`) for calculations or other forms of data
1415                      manipulation.
1416
1417             9.03.03. We declare `@years`, `@months` and `@days` as INT
1418                      (https://msdn.microsoft.com/en-us/library/ms187745.aspx) for
1419                      date functions `DATEADD()` and `DATEDIFF()`.
1420     ************************************************************************ */
1421
1422   DECLARE @persons_name VARCHAR(100),               -- 1. person's first and last
1423                                                     --    names
1424     @start_date DATETIME,                           -- 2. person's birthday
1425     @end_date DATETIME,                             --    today's date from system
1426                                                     --    date and time
1427     @tmp_date DATETIME,                             -- 3. variable for calculations
1428     @years INT,                                     -- 4. variable for number of
1429                                                     --    years
1430     @months INT,                                    -- 5. variable for number of
1431                                                     --    months
1432     @days INT;                                      -- 6. variable for number of
1433                                                     --    days
1434
1435
1436   /* ************************************************************************
1437             9.03.04. We assign a value to `@start_date` as shown below since there
1438                      is no way for SQL Server to prompt the user to enter a value.
1439                      In this example, we are using the date of birth of Linus
1440                      Torvalds (creator of the Linux kernel;
1441                      http://searchenterpriselinux.techtarget.com/definition/Linus-Torvalds).
1442                      We also assign `GETDATE()` to `@end_date`.  This way we can
1443                      change the end date as needed (change from original query).
1444     ************************************************************************ */
1445
1446   SELECT @persons_name = 'Linus Torvalds',          -- person's name
1447     @start_date = '12/28/1969',                     -- person's date of birth
1448     @end_date = GETDATE();                          -- today's system date and time
1449
```

```
/* ************************************************************************
                Note that we can declare the parameter and assign a value
                in-line instead of using a `SELECT` statement.

                  DECLARE @persons_name = 'Linus Torvalds',
                    @start_date = '12/28/1969',
                    @end_date = GETDATE(),
                    @tmp_date DATETIME,
                    @years INT,
                    @months INT,
                    @days INT;
 ************************************************************************ */

SELECT @tmp_date = @start_date;


/* ************************************************************************
        9.03.06. Date functions `DATEADD()` returns ``a specified date with
                the specified number interval (signed integer) added to a
                specified datepart of that date``
                (https://msdn.microsoft.com/en-us/library/ms186819.aspx) and
                `DATEDIFF()` returns ``the count (signed integer) of the
                specified datepart boundaries crossed between the specified
                start_date and end_date``
                (https://msdn.microsoft.com/en-us/library/ms189794.aspx).

                  `YEAR()` retrieves the year (`yy`) from the date.

                  `MONTH()` retrieves the month (`m`) from the date.

                  `DAY()` retrieves the day (`d`) from the date.

        9.03.07. The `CASE WHEN` statement uses a true value (situation we are
                looking for) coming from `WHEN... THEN` to trigger an action
                and an `ELSE` value to trigger an alternative action using
                the following syntax.

        9.03.08. Below `@years` is assigned the difference of `@tmp_date` and
                `@end_date` in years when the month in the year (`yy`) in
                `@start_date` is less than the month in `@end_date` or it is
                the same as the month in `@end_date`

                  MONTH(@start_date) > MONTH(@end_date))
                    OR (MONTH(@start_date) = MONTH(@end_date))

                and the day in `@start_date` is less than the day in
                `@end_date`.

                  AND DAY(@start_date) > DAY(@end_date)

                If both conditions are true, the query returns `1` (under a
                full year).  Otherwise it returns `0` (full year).
 ************************************************************************ */

SELECT @years = DATEDIFF(yy, @tmp_date, @end_date) - CASE
    WHEN (MONTH(@start_date) > MONTH(@end_date))
      OR (
        MONTH(@start_date) = MONTH(@end_date)
        AND DAY(@start_date) > DAY(@end_date)
        )
      THEN 1
    ELSE 0
    END;


/* ************************************************************************
        9.03.09. We add the value of `@years` (`yy`) to `@tmp_date` returning
                1 or 0.
 ************************************************************************ */
```

```
1519
1520  SELECT @tmp_date = DATEADD(yy, @years, @tmp_date);
1521
1522
1523  /* ************************************************************************
1524          9.03.10. Below `@months` is assigned the difference of `@tmp_date` and
1525                   `@end_date` in months when the month (`m`) in `@start_date`
1526                   is less than the month in `@end_date` or it is the same as
1527                   the month in `@end_date`.
1528
1529                     DAY(@start_date) > DAY(@end_date)
1530
1531                   If the condition is true, the query returns `1` (under a full
1532                   month).  Otherwise it returns `0` (full month).
1533   ************************************************************************ */
1534
1535  SELECT @months = DATEDIFF(m, @tmp_date, @end_date) - CASE
1536      WHEN DAY(@start_date) > DAY(@end_date)
1537        THEN 1
1538      ELSE 0
1539      END;
1540
1541
1542  /* ************************************************************************
1543          9.03.11. We add the value of `@months` (`m`) to `@tmp_date` returning
1544                   1 or 0.
1545   ************************************************************************ */
1546
1547  SELECT @tmp_date = DATEADD(m, @months, @tmp_date);
1548
1549
1550  /* ************************************************************************
1551          9.03.12. Below `@days` is assigned the difference of `@tmp_date` and
1552                   `@end_date` in days.
1553   ************************************************************************ */
1554
1555  SELECT @days = DATEDIFF(d, @tmp_date, @end_date);
1556
1557
1558  /* ************************************************************************
1559          9.03.13. We finally display the values for `@years`, `@months` and
1560                   `@days`.
1561
1562                     +----------------+-------+--------+------+
1563                     | Person's Name  | Years | Months | Days |
1564                     +----------------+-------+--------+------+
1565                     | Linus Torvalds | 55    | 11     | 10   |
1566                     +----------------+-------+--------+------+
1567
1568          9.03.14. You can also use the script to calculate your age or any
1569                   difference between any two dates by changing the values in
1570                   section #9.03.04.
1571
1572                   The value returned by `GETDATE()` when running this script
1573                   was 2021/11/29 and the end result will change according to
1574                   the current date when the script is run.
1575   ************************************************************************ */
1576
1577  SELECT @persons_name AS 'Person''s Name',       -- two single quotes (`''`) to
1578                                                  -- escape and show only one
1579                                                  -- (`'`)
1580    @years AS 'Years',
1581    @months AS 'Months',
1582    @days AS 'Days';
1583
1584
1585  /* ************************************************************************
1586   https://folvera.commons.gc.cuny.edu/?p=1401
1587   ************************************************************************ */
```