```
/* ************************************************************************
          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 #7 (2023/04/24): 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,
```

```
  53    AP1.Vendors.VendorContactFName,
  54    AP1.Vendors.DefaultTermsID,
  55    AP1.Vendors.DefaultAccountNo
  56  FROM AP1.ContactUpdates
  57  INNER JOIN AP1.Vendors
  58    ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
  59  WHERE AP1.Vendors.VendorState IN (              -- 3. indicating what values we
  60      'NY',                                       --    query to return
  61      'NJ',
  62      'CA'
  63      );
  64
  65
  66  /* ************************************************************************
  67      At this point, we make a view in schema `lab05` in database `labs`
  68      excluding the `ORDER BY` clause as it cannot be used when creating views.
  69   ************************************************************************ */
  70
  71  CREATE SCHEMA lab05;                            -- 1. creating schema `labs05`
  72
  73  CREATE VIEW lab05.ContactUpdatesVendorsVW       -- 2. creating view
  74  AS                                              --    `ContactUpdatesVendorsVW`
  75  SELECT WS23SQL1001.AP1.ContactUpdates.VendorID, --    in database `labs`
  76    WS23SQL1001.AP1.ContactUpdates.LastName,      --    hence calling database
  77    WS23SQL1001.AP1.ContactUpdates.FirstName,     --    `WS23SQL1001` for each
  78    -- WS23SQL1001.AP1.Vendors.VendorID AS Expr1, --    table
  79    WS23SQL1001.AP1.Vendors.VendorName,
  80    WS23SQL1001.AP1.Vendors.VendorAddress1,
  81    WS23SQL1001.AP1.Vendors.VendorAddress2,
  82    WS23SQL1001.AP1.Vendors.VendorCity,
  83    -- case to replace NY with New York...
  84    CASE
  85      WHEN WS23SQL1001.AP1.Vendors.VendorState = 'NY'
  86        THEN 'New York'
  87      WHEN WS23SQL1001.AP1.Vendors.VendorState = 'NJ'
  88        THEN 'New Jersey'
  89      WHEN WS23SQL1001.AP1.Vendors.VendorState = 'CA'
  90        THEN 'California'
  91      ELSE 'Other'
  92      END AS VendorState,
  93    WS23SQL1001.AP1.Vendors.VendorZipCode,
  94    WS23SQL1001.AP1.Vendors.VendorPhone,
  95    WS23SQL1001.AP1.Vendors.VendorContactLName,
  96    WS23SQL1001.AP1.Vendors.VendorContactFName,
  97    WS23SQL1001.AP1.Vendors.DefaultTermsID,
  98    WS23SQL1001.AP1.Vendors.DefaultAccountNo
  99  FROM WS23SQL1001.AP1.ContactUpdates
 100  INNER JOIN WS23SQL1001.AP1.Vendors
 101    ON WS23SQL1001.AP1.ContactUpdates.VendorID = WS23SQL1001.AP1.Vendors.VendorID
 102  WHERE WS23SQL1001.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 `WS23SQL1001.AP1.ContactUpdates`
114                    `v` for `WS23SQL1001.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,
139    v.VendorContactFName,
140    v.DefaultTermsID,
141    v.DefaultAccountNo
142  FROM WS23SQL1001.AP1.ContactUpdates AS c
143  INNER JOIN WS23SQL1001.AP1.Vendors AS v
144    ON c.VendorID = v.VendorID
145  WHERE v.VendorState IN (
146      'NY',
147      'NJ',
148      'CA'
149      );
150
151
152  /* ********************************************************************
153   2. LAB #6
154      Write a query without duplicate rows (`SELECT DISTINCT`)
155      2.01. to call all shared values from tables `AP1.Invoices` and `AP1.Terms`,
156      2.02. to format all dates as `yyyy-MM-dd` and currency as pounds sterling
```

```
157           (culture `en-gb`)
158      2.03. where `AP1.Invoices.PaymentTotal` is greater than the average value
159           of `AP1.Invoices.InvoiceTotal` (sub-query within the `WHERE` clause)
160           and `AP1.Invoices.PaymentDate` is not null.
161      HINT:  `AVG(AP1.Invoices.InvoiceTotal) FROM AP1.Invoices` for the nested
162             query value
163      BONUS: Make a view in schema `lab06` in database `labs`.
164   ************************************************************************** */
165
166  SELECT AP1.Invoices.InvoiceID,
167    AP1.Invoices.VendorID,
168    AP1.Invoices.InvoiceNumber,
169    FORMAT(AP1.Invoices.InvoiceDate,              -- 1. formatting column
170                                                  --    `InvoiceDate` as
171      'yyyy-MM-dd', 'en-gb')                      --    `yyyy-MM-dd` date with
172                                                  --    culture `en-gb` using
173       AS InvoiceDate,                            --    alias `InvoiceDate`
174    FORMAT(AP1.Invoices.InvoiceTotal,             -- 2. formatting column
175                                                  --    `InvoiceTotal` as
176      'c', 'en-gb')                               --    `c` (currency) with
177                                                  --    culture `en-gb` using
178      AS InvoiceTotal,                            --    alias `InvoiceTotal`
179    FORMAT(AP1.Invoices.PaymentTotal,             -- 3. formatting column
180      'c', 'en-gb')                               --    as `c` (currency) with
181                                                  --    culture `en-gb` using
182      AS PaymentTotal,                            --    alias `PaymentTotal`
183    FORMAT(AP1.Invoices.CreditTotal,              -- 4. formatting column
184      'c', 'en-gb')                               --    as `c` (currency) with
185                                                  --    culture `en-gb` using
186      AS CreditTotal,                             --    alias `CreditTotal`
187    AP1.Invoices.TermsID,
188    FORMAT(AP1.Invoices.InvoiceDueDate,           -- 5. formatting column
189      'yyyy-MM-dd', 'en-gb')                      --    as `yyyy-MM-dd` date with
190                                                  --    culture `en-gb` using
191      AS InvoiceDueDate,                          --    alias `InvoiceDueDate`
192    FORMAT(AP1.Invoices.PaymentDate,              -- 6. formatting column
193      'yyyy-MM-dd', 'en-gb')                      --    as `yyyy-MM-dd` date with
194                                                  --    culture `en-gb` using
195      AS PaymentDate,                             --    alias `PaymentDate`
196    AP1.Terms.TermsDescription,
197    AP1.Terms.TermsDueDays
198  FROM AP1.Invoices                               -- 7. from table `AP1.Invoices`
199  INNER JOIN AP1.Terms                            --    using `INNER JOIN` to
200                                                  --    retrieve all shared data
201    ON AP1.Invoices.TermsID = AP1.Terms.TermsID   --    connecting both tables on
202                                                  --    shared field `TermsID`
203  WHERE (                                         -- 8. where the value of
204      AP1.Invoices.PaymentTotal > (               --    `PaymentTotal` is larger
205                                                  --    than (`>`) the single
206        SELECT AVG(PaymentTotal)                  --    value of sub-query
207        FROM AP1.Invoices                         --        `(SELECT
208                                                  --         AVG(PaymentTotal)
```

```
209                                                 --         FROM AP1.Invoices)`
210                                                 --    that returns 1879.7413
211       )                                         --    8.1. sub-query always in
212     )                                           --         parenthesis, just
213                                                 --         like in algebra
214                                                 --    8.2. no need for
215                                                 --         `ORDER BY` since
216                                                 --         aggregate function
217                                                 --         `AVG()` affects only
218                                                 --         one column and it
219                                                 --         does not affect the
220                                                 --         main query
221   AND AP1.Invoices.PaymentDate IS NOT NULL;     -- 9. and [where] value of
222                                                 --    `PaymentDate` is not null
223                                                 --    (must have a value)
224
225
226 /* ***************************************************************************
227     At this point, we make a view in schema `lab06` in database `labs`
228     excluding the `ORDER BY` clause as it cannot be used when creating views.
229  *************************************************************************** */
230
231 CREATE SCHEMA lab06;                            -- 1. creating schema `labs06`
232
233 CREATE VIEW lab06.InvoicesTermsVW               -- 2. creating view
234 AS                                              --    `InvoicesTermsVW` in
235 SELECT WS23SQL1001.AP1.Invoices.InvoiceID,      --    database `labs` hence
236   WS23SQL1001.AP1.Invoices.VendorID,            --    calling database
237   WS23SQL1001.AP1.Invoices.InvoiceNumber,       --    `WS23SQL1001` for each
238   FORMAT(WS23SQL1001.AP1.Invoices.InvoiceDate,  --    table
239     'yyyy-MM-dd', 'en-gb') AS InvoiceDate,
240   FORMAT(WS23SQL1001.AP1.Invoices.InvoiceTotal,
241     'c', 'en-gb') AS InvoiceTotal,
242   FORMAT(WS23SQL1001.AP1.Invoices.PaymentTotal,
243     'c', 'en-gb') AS PaymentTotal,
244   FORMAT(WS23SQL1001.AP1.Invoices.CreditTotal,
245     'c', 'en-gb') AS CreditTotal,
246   WS23SQL1001.AP1.Invoices.TermsID,
247   FORMAT(WS23SQL1001.AP1.Invoices.InvoiceDueDate,
248     'yyyy-MM-dd', 'en-gb') AS InvoiceDueDate,
249   FORMAT(WS23SQL1001.AP1.Invoices.PaymentDate,
250     'yyyy-MM-dd', 'en-gb') AS PaymentDate,
251   WS23SQL1001.AP1.Terms.TermsDescription,
252   WS23SQL1001.AP1.Terms.TermsDueDays
253 FROM WS23SQL1001.AP1.Invoices
254 INNER JOIN WS23SQL1001.AP1.Terms
255   ON WS23SQL1001.AP1.Invoices.TermsID = WS23SQL1001.AP1.Terms.TermsID
256 WHERE (
257     WS23SQL1001.AP1.Invoices.PaymentTotal > (
258       SELECT AVG(PaymentTotal)
259       FROM WS23SQL1001.AP1.Invoices
260       )
```

```
    )
  AND WS23SQL1001.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 `WS23SQL1001.AP1.Invoices`
                    `t` for `WS23SQL1001.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 WS23SQL1001.AP1.Invoices AS i
INNER JOIN WS23SQL1001.AP1.Terms AS t
  ON i.TermsID = t.TermsID
WHERE (
    i.PaymentTotal > (
      SELECT AVG(PaymentTotal)
      FROM WS23SQL1001.AP1.Invoices
      )
    )
  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]
```

```
313
314                     DROP obj_type object_name
315                     [other_code]
316
317                     ALTER obj_type object_name
318                     ALTER|ADD|DROP obj_type obj_name data_type [other_code]
319
320                     INSERT INTO table_name
321                     VALUES
322                       (
323                         field1 datatype1,
324                         field2 datatype2
325                         ...
326                       )
327
328                     DELETE FROM table_name
329                     [other_code]
330
331                     TRUNCATE TABLE table_name
332
333                     UPDATE table_name
334                     SET field = new_value
335
336     We use SQL to return data to any person or program that needs data.
337
338     We can use functions to change the output of data as well as the data
339     itself, which we will see later in the course.
340
341  4. Although using a custom format like `yyyy-MM-dd` overrides the culture
342     (`en-us`) and there is no longer need to include this culture, it is a good
343     idea to include it as good practice.
344  ********************************************************************** */
345
346  SELECT FORMAT(InvoiceTotal, 'yyyy-MM-dd')       -- no culture (`en-us`) needed
347  FROM AP1.Invoices;                              -- because of the custom format
348
349  SELECT FORMAT(InvoiceTotal,                     -- good practice to include the
350    'yyyy-MM-dd', 'en-us')                        -- culture (`en-us`) even when
351  FROM AP1.Invoices;                              -- overridden by custom format
352
353
354  /* ********************************************************************
355   5. As mentioned several times, `FORMAT()` changes numeric values to strings.
356      We can also use `CONVERT()` to change ``an expression from a data type to
357      another data type`` -- in other words, numeric values to strings or vice
358      versa (https://techonthenet.com/sql_server/functions/convert.php).
359
360                      CONVERT(new_data_type, column)
361
362     `CONVERT()` does not change the currency sign or adds commas to divide
363     thousands or millions as `FORMAT()` does.
364
```

```
365      5.01. In the example below, we change the data type of `InvoiceTotal` to
366            VARCHAR(50) -- an allocation in RAM to hold a variable character
367            value with a maximum size of fifty (50) characters.
368   ************************************************************************ */
369
370  SELECT CONVERT(VARCHAR(50), InvoiceTotal)       -- changing data type of column
371    AS InvoiceTotal                               -- `InvoiceTotal` (`FLOAT`) to
372  FROM AP1.Invoices;                              -- `VARCHAR(50)`
373
374
375  /* ************************************************************************
376      5.02. In the example below, we use `CONVERT()` to return the value of
377            `AP1.Invoices.InvoiceTotal` as a dollar amount concatenating the
378            dollar sign (`$`) at the beginning.
379   ************************************************************************ */
380
381  SELECT CONCAT (
382          '$',                                    -- concatenating `$` to the
383          CONVERT(VARCHAR(50), InvoiceTotal)      -- output of
384          ) AS InvoiceTotal                       --    `CONVERT(VARCHAR(50),
385  FROM AP1.Invoices;                              --    `InvoiceTotal)`
386
387
388  /* ************************************************************************
389            We could also use `CONVERT()` to return the value of
390            `AP1.Invoices.InvoiceTotal` as a dollar amount with `USD` rather than
391            the dollar sign (`$`).
392   ************************************************************************ */
393
394  SELECT CONCAT (
395          'USD ',                                 -- concatenating `USD ` to the
396          CONVERT(VARCHAR(50), InvoiceTotal)      -- output of
397          ) AS InvoiceTotal                       --    `CONVERT(VARCHAR(50),
398  FROM AP1.Invoices;                              --    `InvoiceTotal)`
399
400
401  /* ************************************************************************
402            Of course, if you are ``dressing up`` a numeric value like
403            `AP1.Invoices.InvoiceTotal` as currency, it is better to just use
404            `FORMAT()` to keep your code simple.
405   ************************************************************************ */
406
407  SELECT FORMAT(InvoiceTotal, 'c', 'en-us') AS InvoiceTotal
408  FROM AP1.Invoices;
409
410
411  /* ************************************************************************
412      5.03. In the example below, we use `CONVERT()` to change the data type of
413            `AP1.Invoices.InvoiceID` and `AP1.Invoices.VendorID` from FLOAT to
414            `VARCHAR(50)` before concatenating these values to a string.
415   ************************************************************************ */
416
```

```
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
```

```
469
470           <         less than
471                     https://techonthenet.com/sql_server/comparison_operators.php
472
473           >         greater than
474                     https://techonthenet.com/sql_server/comparison_operators.php
475
476           <=        less than or equal to
477                     https://techonthenet.com/sql_server/comparison_operators.php
478
479           >=        greater than or equal to
480                     https://techonthenet.com/sql_server/comparison_operators.php
481
482           !>        not greater than (same as <=)
483                     https://techonthenet.com/sql_server/comparison_operators.php
484
485           !<        not less than (same as >=)
486                     https://techonthenet.com/sql_server/comparison_operators.php
487
488           LIKE      allows wild cards to be used in the WHERE clause of a
489                     SELECT, INSERT, UPDATE, or DELETE statement [allowing] you
490                     to perform pattern matching
491                     https://techonthenet.com/sql_server/like.php
492
493           IN        to help reduce the need to use multiple OR conditions in a
494                     SELECT, INSERT, UPDATE, or DELETE statement
495                     https://techonthenet.com/sql_server/in.php
496
497           BETWEEN   used to retrieve values within a range in a SELECT, INSERT,
498                     UPDATE, or DELETE statement
499                     https://techonthenet.com/sql_server/between.php
500
501           IS NULL   condition... used to test for a NULL no value
502                     https://techonthenet.com/sql_server/is_null.php
503
504           NOT       to negate a condition in a SELECT, INSERT, UPDATE, or
505                     DELETE statement
506                     https://techonthenet.com/sql_server/not.php
507
508                     * NOT LIKE
509                     * NOT IN
510                     * NOT BETWEEN
511                     * IS NOT NULL
512                       https://techonthenet.com/sql_server/is_not_null.php
513
514      6.03. In the example, below, we retrieve all values from table
515           `AP1.Vendors` where `VendorState` is equal to `CA` and `VendorCity`
516           could either be `Fresno` or `Sacramento`.
517
518           Use parenthesis for SQL (regardless of vendor/distribution) to
519           process the inner condition first
520
```

```
                    (
                      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
WHERE VendorState <> 'NY';                      -- `VendorState != `NY``


/* ************************************************************************
    6.05. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorState` is either `DC` or `IA`.
 ************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorState = 'DC'                        -- checking if either criterion
  OR VendorState = 'IA';                        -- is true


/* ************************************************************************
    6.06. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorAddress2` is NULL (no-value) using `NOT` as it negates
          operators `LIKE` as `NOT LIKE`, `IN` as `NOT IN`, `BETWEEN` as
          `NOT BETWEEN` and `IS NULL` as `IS NOT NULL`.
 ************************************************************************** */

SELECT *
```

```
573  FROM AP1.Vendors
574  WHERE VendorAddress2 IS NULL;                    -- asking for no-value
575
576
577  /* ************************************************************************
578      6.07. In the example below, we retrieve all values from table `AP1.Vendors`
579            where `VendorAddress2` is not NULL (not a no-value).  Refer to
580            https://techonthenet.com/sql_server/is_not_null.php for more
581            information.
582   ************************************************************************ */
583
584  SELECT *
585  FROM AP1.Vendors
586  WHERE VendorAddress2 IS NOT NULL;                -- asking for not `NOT NULL`
587                                                   -- (no no-value)
588
589
590  /* ************************************************************************
591      6.08. In the example below, we rewrite #6.3 in a cleaner fashion to
592            retrieve all values from table `AP1.Vendors` where `VendorState` is
593            equal to `CA` and `VendorCity` could either be `Fresno` or
594            `Sacramento`.  We use operator `IN`
595            (https://techonthenet.com/sql_server/in.php) to specify the list of
596            values that can be true for `VendorCity`.
597   ************************************************************************ */
598
599  SELECT *
600  FROM AP1.Vendors
601  WHERE VendorState = 'CA'                         -- 1. first condition as in
602                                                   --    original example
603    AND VendorCity IN (                            -- 2. second condition using
604      'Fresno',                                    --    `IN` to list all possible
605      'Sacramento'                                 --    values that can be true
606      );                                           --    (satisfied)
607
608
609  /* ************************************************************************
610      6.09. In the example below, we retrieve all values from table `AP1.Vendors`
611            where `VendorState` could either be `CA` or `NJ` and `VendorCity`
612            could either be `Fresno` or `Sacramento`.
613
614            This query looks for the combination of
615
616                      `CA` and `Fresno`     (true)
617                      `CA` and `Sacramento` (true)
618
619            as well as
620
621                      `NJ` and `Fresno`     (false)
622                      `NJ` and `Sacramento` (false)
623
624            The query only returns only the first set of values since we do not
```

```
625           have any records where `VendorCity` is `NJ` and VendorCity` is either
626           `Fresno` or `Sacramento`.
627  ************************************************************************ */
628
629  SELECT *
630  FROM AP1.Vendors
631  WHERE (
632      VendorState IN (                          -- 1. first condition using
633        'CA',                                   --    `IN` to list all possible
634        'NJ'                                    --    values that can be true
635        )                                       --    (satisfied)
636      AND VendorCity IN (                       -- 2. second condition using
637        'Fresno',                               --    `IN` to list all possible
638        'Sacramento'                            --    values that can be true
639        )                                       --    (satisfied)
640      )
641  ORDER BY VendorState,
642    VendorCity;
643
644
645  /* ************************************************************************
646      6.10. In the example below, we retrieve all values from table `AP1.Vendors`
647            where `VendorState` could either be `CA` and `VendorCity` could
648            either be `Fresno` or `Sacramento` as one condition or `VendorState`
649            is `NJ` as another condition.
650   ************************************************************************ */
651
652  SELECT *
653  FROM AP1.Vendors
654  WHERE (                                       -- 1. first condition where
655      VendorState IN ('CA')                     --    `VendorState` could be
656      AND VendorCity IN (                       --    `CA` and `VendorCity`
657        'Fresno',                               --    could either be `Fresno`
658        'Sacramento'                            --    or `Sacramento`
659        )                                       --    looking the combinations
660      )                                         --    of `CA` and `Fresno` or
661                                                --    `CA` and `Sacramento`
662    OR VendorState IN ('NJ')                    -- 2. second condition starting
663                                                --    with `OR` to specify that
664                                                --    `VendorState` could also
665                                                --    be `NJ`
666  ORDER BY VendorState,                         -- 3. ordering results first by
667    VendorCity;                                 --    `VendorState` and then by
668                                                --    `VendorCity`
669
670
671  /* ************************************************************************
672      6.11. In the example below, we retrieve all values from table `AP1.Vendors`
673            where `VendorName` has as a value starting with `am` (not case
674            sensitive) using wild card `%` to represent any character or group of
675            after `am`.
676   ************************************************************************ */
```

```sql

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
```

```
729 WHERE InvoiceDueDate BETWEEN '2012-01-01'       -- range between `2012-01-01`
730     AND '2012-01-30';                           -- and `2012-01-30`
731
732
733 /* ***************************************************************************
734     6.16. In the example below, we retrieve all values from table `AP1.Vendors`
735           where InvoiceTotal has values within 100 and 1000.  Then we organize
736           the results in descending order using an `ORDER BY` clause
737           (https://techonthenet.com/sql/order_by.php).
738
739           The default option for `ORDER BY` is `ASC` (ascending), which can be
740           omitted.
741
742           The opposite option for `ORDER BY` is `DESC` (descending), which
743           needs to be specified.
744  *************************************************************************** */
745
746 SELECT *
747 FROM AP1.Invoices
748 WHERE InvoiceTotal BETWEEN 100                  -- range between 100 and 1000
749     AND 1000
750 ORDER BY InvoiceTotal DESC,                    -- organizing results first by
751                                                 -- `InvoiceTotal` in descending
752                                                 -- order,
753   PaymentTotal DESC,                            -- then by `PaymentTotal` in
754                                                 -- descending order
755   TermsID DESC;                                 -- and finally by `TermsID`
756                                                 -- also in descending order
757
758
759 /* ***************************************************************************
760  7. As we have mentioned several times, when calling multiple tables, we need
761     to `JOIN` them (https://techonthenet.com/sql_server/joins.php).
762
763     7.01. `INNER JOIN` returns ``all rows from multiple tables where the join
764           condition is met.``
765
766           In the example below, we retrieve all records shared in tables
767           `AP1.Invoices` and `AP1.Invoices`.
768  *************************************************************************** */
769
770 SELECT *
771 FROM AP1.Vendors
772 INNER JOIN AP1.Invoices
773   ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;
774
775
776 /* ***************************************************************************
777     7.02. `LEFT JOIN` returns ``all rows from the LEFT-hand table specified in
778           the ON condition and only those rows from the other table where the
779           joined fields are equal (join conditions met).``
780
```

```
781           In the example below, we retrieve all records in `AP1.Vendors` (left
782           table) and any records in `AP1.Invoices` (if any in the right table).
783    ********************************************************************** */
784
785   SELECT *
786   FROM AP1.Vendors                                -- retrieves all records from
787   LEFT JOIN AP1.Invoices                          -- the left table/dataset
788     ON AP1.Vendors.VendorID = AP1.Invoices.VendorID; -- (first table/dataset
789                                                   -- called in the statement,
790                                                   -- `AP1.Vendors`) and related
791                                                   -- records from the right
792                                                   -- table/dataset (second
793                                                   -- table/dataset called in the
794                                                   -- statement, `AP1.Invoices`);
795                                                   -- returns 202
796
797
798   /* **********************************************************************
799           In the example below, we retrieve all records in `AP1.Invoices` (left
800           table) and any records in `AP1.Vendors` (if any in the right table).
801    ********************************************************************** */
802
803   SELECT *
804   FROM AP1.Invoices                               -- retrieves all records from
805   LEFT JOIN AP1.Vendors                           -- the left table/dataset
806     ON AP1.Vendors.VendorID = AP1.Invoices.VendorID; -- (first table/dataset
807                                                   -- called in the statement,
808                                                   -- `AP1.Invoices`) and related
809                                                   -- records from the right
810                                                   -- table/dataset (second
811                                                   -- table/dataset called in the
812                                                   -- statement, `AP1.Vendors`)
813
814
815   /* **********************************************************************
816       7.03. `RIGHT JOIN` returns ``all rows from the RIGHT-hand table specified in
817           the ON condition and only those rows from the other table where the
818           joined fields are equal (join condition is met).``
819
820           In the example below, we retrieve all records in `AP1.Invoices`
821           (right table) and any records in `AP1.Vendors` (if any in the left
822           table).
823    ********************************************************************** */
824
825   SELECT *
826   FROM AP1.Vendors                                -- retrieves all records from
827   RIGHT JOIN AP1.Invoices                         -- the right table/dataset
828     ON AP1.Invoices.VendorID = AP1.Vendors.VendorID; -- (second table/dataset
829                                                   -- called in the statement,
830                                                   -- `AP1.Invoices`) and related
831                                                   -- records from the left
832                                                   -- table/dataset (first
```

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

```
885  ADD CustomerID INT NULL;
886
887  UPDATE AP1.Invoices
888  SET CustomerID = 1
889  WHERE VendorID = 34;
890
891  UPDATE AP1.Invoices
892  SET CustomerID = 2
893  WHERE VendorID = 37;
894
895  UPDATE AP1.Invoices
896  SET CustomerID = 3
897  WHERE VendorID = 89;
898
899
900  /* ************************************************************************
901      8.02. Now that relationship has been created, we can now query tables
902            `AP1.Invoices` and `AP2.Customers` (each tables in a different
903            databases).
904   ************************************************************************ */
905
906  SELECT DISTINCT AP1.Invoices.InvoiceID,
907    AP1.Invoices.VendorID,
908    AP1.Invoices.InvoiceNumber,
909    FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
910    FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
911    FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
912    FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
913    AP1.Invoices.TermsID,
914    FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
915    FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
916    AP1.Invoices.CustomerID,
917    AP2.Customers.LastName,
918    AP2.Customers.FirstName,
919    AP2.Customers.Address,
920    AP2.Customers.City,
921    AP2.Customers.STATE,
922    AP2.Customers.ZipCode,
923    AP2.Customers.Email
924  FROM AP1.Invoices
925  INNER JOIN AP2.Customers
926    ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
927  ORDER BY AP1.Invoices.VendorID;
928
929
930  /* ************************************************************************
931      8.03. In the example below, we can create a view using the query in the
932             example above using tables `AP1.Invoices` and `AP2.Customers`
933             without `ORDER BY`, which would return an error when creating the
934             view.
935
936             Tables and views cannot share names since both data objects are of
```

```
937             the same hierarchy.
938
939             We can query, alter and/or drop a view just like a table.
940
941             In most relational databases, we cannot update data using a view
942             since this action only take place in tables.
943
944             In SQL Server (T-SQL), we can update data from the base table.
945
946                     ``Requires UPDATE, INSERT, or DELETE permissions on the
947                     target table, depending on the action being performed.``
948                     https://msdn.microsoft.com/en-us/library/ms180800.aspx
949  ************************************************************************** */
950
951 CREATE VIEW AP1.InvoicesCustomersVW
952 AS
953 (
954     SELECT DISTINCT AP1.Invoices.InvoiceID,
955       AP1.Invoices.VendorID,
956       AP1.Invoices.InvoiceNumber,
957       FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
958       FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
959       FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
960       FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
961       AP1.Invoices.TermsID,
962       FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
963       FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
964       AP1.Invoices.CustomerID,
965       AP2.Customers.LastName,
966       AP2.Customers.FirstName,
967       AP2.Customers.Address,
968       AP2.Customers.City,
969       AP2.Customers.STATE,
970       AP2.Customers.ZipCode,
971       AP2.Customers.Email
972     FROM AP1.Invoices
973     INNER JOIN AP2.Customers
974       ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
975     );
976
977
978 /* ************************************************************************
979     8.04. We can modify a view simply changing `CREATE` for `ALTER`.
980  ************************************************************************** */
981
982 ALTER VIEW AP1.InvoicesCustomersVW
983 AS
984 (
985     SELECT DISTINCT AP1.Invoices.InvoiceID,
986       AP1.Invoices.VendorID,
987       AP1.Invoices.InvoiceNumber,
988       FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us')
```

```
989           AS InvoiceDate,
990         FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
991         FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
992         FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
993         AP1.Invoices.TermsID,
994         FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
995         FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
996         AP1.Invoices.CustomerID,
997         AP2.Customers.LastName,
998         AP2.Customers.FirstName,
999         AP2.Customers.Address,
1000        AP2.Customers.City,
1001        AP2.Customers.STATE,
1002        AP2.Customers.ZipCode,
1003        AP2.Customers.Email,
1004        GETDATE() AS SystemDate -- change in query
1005    FROM AP1.Invoices
1006    INNER JOIN AP2.Customers
1007      ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
1008    );
1009
1010
1011 /* ************************************************************************
1012    8.05. In the example below, we create view `AP1.InvoicesVW` only from table
1013          `AP1.Invoices` formatting the date and currency fields accordingly.
1014          This way we do not need to format the columns again and again every
1015          time we need to call them.
1016  ************************************************************************ */
1017
1018 CREATE VIEW AP1.InvoicesVW
1019 AS
1020 (
1021    SELECT DISTINCT InvoiceID,
1022      VendorID,
1023      InvoiceNumber,
1024      FORMAT(InvoiceDate, 'd', 'en-us') AS InvoiceDate,
1025      FORMAT(InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1026      FORMAT(PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1027      FORMAT(CreditTotal, 'c', 'en-us') AS CreditTotal,
1028      TermsID,
1029      FORMAT(InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
1030      FORMAT(PaymentDate, 'd', 'en-us') AS PaymentDate,
1031      CustomerID
1032   FROM AP1.Invoices
1033   );
1034
1035
1036 /* ************************************************************************
1037    8.06. In the example below, we create view `AP1.InvoicesVendorsVW` from
1038          tables `AP1.Invoices` and `AP1.Vendors`.
1039
1040          Unless we indicate in which database to store the view, it would most
```

```
1041           likely be in the same database where the previous view was stored
1042           (`AP2`).
1043    ********************************************************************** */
1044
1045  CREATE VIEW AP1.InvoicesVendorsVW
1046  AS
1047  (
1048      SELECT DISTINCT AP1.Invoices.InvoiceID,
1049        AP1.Invoices.VendorID,
1050        AP1.Invoices.InvoiceNumber,
1051        AP1.Invoices.InvoiceDate,
1052        AP1.Invoices.InvoiceTotal,
1053        AP1.Invoices.PaymentTotal,
1054        AP1.Invoices.CreditTotal,
1055        AP1.Invoices.TermsID,
1056        AP1.Invoices.InvoiceDueDate,
1057        AP1.Invoices.PaymentDate,
1058        AP1.Vendors.VendorName,
1059        CASE
1060          WHEN AP1.Vendors.VendorAddress2 IS NOT NULL
1061            THEN CONCAT (
1062                AP1.Vendors.VendorAddress1,
1063                ' ',
1064                AP1.Vendors.VendorAddress2
1065                )
1066          WHEN AP1.Vendors.VendorAddress1 IS NULL
1067            AND AP1.Vendors.VendorAddress2 IS NULL
1068            THEN 'No Address'
1069          ELSE AP1.Vendors.VendorAddress1
1070          END AS VendorAddress,
1071        AP1.Vendors.VendorCity,
1072        AP1.Vendors.VendorState,
1073        AP1.Vendors.VendorZipCode,
1074        AP1.Vendors.DefaultAccountNo
1075      FROM AP1.Invoices
1076      LEFT JOIN AP1.Vendors
1077        ON AP1.Invoices.VendorID = AP1.Vendors.VendorID
1078      );
1079
1080
1081  /* **********************************************************************
1082      8.07. In the example below, we create view
1083           `AP1.Invoices_Customers_Vendors_VW` from views (like we would do with
1084           tables) `AP1.InvoicesCustomersVW` and `AP1.InvoicesVendorsVW`.
1085
1086           As mentioned, unless we indicate in which database to store the new
1087           view, it is saved in `AP2`.
1088
1089           We do not need to call the database and schema (`dbo`), but it is
1090           always a good idea -- good practice.
1091    ********************************************************************** */
1092
```

```
1093  CREATE VIEW AP1.Invoices_Customers_Vendors_VW
1094  AS
1095  (
1096      SELECT DISTINCT AP1.InvoicesCustomersVW.InvoiceID,
1097        AP1.InvoicesCustomersVW.VendorID,
1098        AP1.InvoicesCustomersVW.InvoiceNumber,
1099        AP1.InvoicesCustomersVW.InvoiceDate,
1100        AP1.InvoicesCustomersVW.InvoiceTotal,
1101        AP1.InvoicesCustomersVW.PaymentTotal,
1102        AP1.InvoicesCustomersVW.CreditTotal,
1103        AP1.InvoicesCustomersVW.TermsID,
1104        AP1.InvoicesCustomersVW.InvoiceDueDate,
1105        AP1.InvoicesCustomersVW.PaymentDate,
1106        AP1.InvoicesCustomersVW.CustomerID,
1107        AP1.InvoicesCustomersVW.LastName,
1108        AP1.InvoicesCustomersVW.FirstName,
1109        AP1.InvoicesCustomersVW.Address,
1110        AP1.InvoicesCustomersVW.City,
1111        AP1.InvoicesCustomersVW.STATE,
1112        AP1.InvoicesCustomersVW.ZipCode,
1113        AP1.InvoicesCustomersVW.Email,
1114        AP1.InvoicesVendorsVW.VendorName,
1115        AP1.InvoicesVendorsVW.VendorAddress,
1116        AP1.InvoicesVendorsVW.VendorCity,
1117        AP1.InvoicesVendorsVW.VendorState,
1118        AP1.InvoicesVendorsVW.VendorZipCode,
1119        AP1.InvoicesVendorsVW.DefaultAccountNo
1120      FROM AP1.InvoicesCustomersVW
1121      LEFT OUTER JOIN AP1.InvoicesVendorsVW
1122        ON AP1.InvoicesCustomersVW.VendorID = AP1.InvoicesVendorsVW.VendorID
1123      );
1124
1125
1126  /* ***************************************************************************
1127   9. Depending on the relational database management system (RDBMS) and even the
1128      product related to each RDBMS, the date format might vary.  In SQL Server,
1129      we can query data using format `YYYY/MM/DD` (including quotes) although the
1130      system returns format `YYYY-MM-DD` plus time in format `hh:mm:ss.nnnnnnn`.
1131      Refer to https://msdn.microsoft.com/en-us/library/bb630352.aspx and
1132      https://msdn.microsoft.com/en-us/library/bb677243.aspx for information on
1133      date and time respectively.
1134
1135      9.01. The most common date functions are the following.
1136
1137            DAY()     returns the day of the month (1 to 31) given a date value
1138                      https://techonthenet.com/sql_server/functions/day.php
1139
1140            MONTH()   returns the month (1 to 12) given a date value
1141                      https://techonthenet.com/sql_server/functions/month.php
1142
1143            YEAR()    returns a four-digit year (as a number) given a date value
1144                      https://techonthenet.com/sql_server/functions/year.php
```

```
1145
1146            GETDATE() returns the current date and time
1147                      https://techonthenet.com/sql_server/functions/getdate.php
1148   ********************************************************************** */
1149
1150  SELECT DAY('2021/09/15') AS Day,                -- 1. returns `15` from
1151                                                  --    `2021/09/15` without
1152                                                  --    leading zeros (`d`)
1153    MONTH('2021/09/15') AS Month,                 -- 2. returns `9` from
1154                                                  --    `2021/09/15` without
1155                                                  --    leading zeros (`M`)
1156    YEAR('2021/09/15') AS Year;                   -- 3. returns `2021` from
1157                                                  --    `2021/09/15 (`yyyy`)
1158
1159  SELECT GETDATE() AS CurrentDateTime;            -- returns
1160                                                  -- `2021-09-15 20:20:34.053`
1161                                                  -- from `GETDATE()` that calls
1162                                                  -- system date and time
1163
1164  SELECT DAY(GETDATE()) AS Day,                   -- 1. returns `15` from system
1165                                                  --    DATETIME without leading
1166                                                  --    zeros (`d`)
1167    MONTH(GETDATE()) AS Month,                    -- 2. returns `9` from system
1168                                                  --    DATETIME without leading
1169                                                  --    zeros (`M`)
1170    YEAR(GETDATE()) AS Year;                      -- 3. returns `2021` from
1171                                                  --    system DATETIME (`yyyy`)
1172
1173  SELECT FORMAT(GETDATE(), 'd', 'en-us')          -- returns system date and time
1174    AS FormattedCurrentDateTime;                  -- formatted as `9/15/2021`
1175
1176
1177  /* ************************************************************************
1178      9.02. Instead of hard-coding the date in the example above (#3.1), we can
1179            use parameter `@date` in all instances that we need to pass the value
1180            returned by `GETDATE()`.
1181
1182            We must declare each parameter with its proper data type.
1183
1184            We can then have to pass (`SET`) a value for each parameter.
1185   ********************************************************************** */
1186
1187  DECLARE @date DATETIME = GETDATE()              -- 1. declaring parameter
1188                                                  --    `@date` as DATETIME (the
1189                                                  --    proper data type) and
1190                                                  --    passing value of
1191                                                  --    `GETDATE()`
1192
1193  SELECT DAY(@date) AS Day,                       -- 2. returns `9` from system
1194                                                  --    DATETIME without leading
1195                                                  --    zeros (`d`)
1196    MONTH(@date) AS Month,                        -- 3. returns `15` from system
```

```
                                                --    DATETIME without leading
                                                --    zeros (`M`)
  YEAR(@date) AS Year;                          -- 4. returns `2021` from
                                                --    system DATETIME (`yyyy`)


/* ************************************************************************
    9.03. We can also use date function `GETDATE()` to calculate age in months,
          days and years.

          The following example (9.03.01 to 9.03.14) is based on the answer
          found at http://stackoverflow.com/q/57599/, which is explained below
          in detail.

          9.03.01. We declare variables `@start_date`, `@end_date` and
                   `@tmp_date` as data type DATETIME
                   (https://msdn.microsoft.com/en-us/library/ms187819.aspx).

          9.03.02. It is good practice to use a second variable (in this case,
                   `@tmp_date`) for calculations or other forms of data
                   manipulation.

          9.03.03. We declare `@years`, `@months` and `@days` as INT
                   (https://msdn.microsoft.com/en-us/library/ms187745.aspx) for
                   date functions `DATEADD()` and `DATEDIFF()`.
 ************************************************************************ */

DECLARE @persons_name VARCHAR(100),             -- 1. person's first and last
                                                --    names
  @start_date DATETIME,                         -- 2. person's birthday
  @end_date DATETIME,                           --    today's date from system
                                                --    date and time
  @tmp_date DATETIME,                           -- 3. variable for calculations
  @years INT,                                   -- 4. variable for number of
                                                --    years
  @months INT,                                  -- 5. variable for number of
                                                --    months
  @days INT;                                    -- 6. variable for number of
                                                --    days


/* ************************************************************************
         9.03.04. We assign a value to `@start_date` as shown below since there
                  is no way for SQL Server to prompt the user to enter a value.
                  In this example, we are using the date of birth of Linus
                  Torvalds (creator of the Linux kernel;
                  http://searchenterpriselinux.techtarget.com/definition/Linus-
                   Torvalds).
                  We also assign `GETDATE()` to `@end_date`.  This way we can
                  change the end date as needed (change from original query).
 ************************************************************************ */

```

```
SET @persons_name = 'Linus Torvalds';          -- person's name
SET @start_date =   '12/28/1969';              -- person's date of birth
SET @end_date =     GETDATE();                 -- today's system date and time


/* ************************************************************************
        9.03.05. We assign the value of `@start_date` to `@tmp_date` to run
                 calculations against it and use `@start_date` as a constant.
 ************************************************************************ */

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
```

```
1300      WHEN (MONTH(@start_date) > MONTH(@end_date))
1301        OR (
1302          MONTH(@start_date) = MONTH(@end_date)
1303          AND DAY(@start_date) > DAY(@end_date)
1304          )
1305        THEN 1
1306      ELSE 0
1307      END;
1308
1309
1310  /* ***************************************************************************
1311           9.03.09. We add the value of `@years` (`yy`) to `@tmp_date` returning
1312                    1 or 0.
1313   *************************************************************************** */
1314
1315  SELECT @tmp_date = DATEADD(yy, @years, @tmp_date);
1316
1317
1318  /* ***************************************************************************
1319           9.03.10. Below `@months` is assigned the difference of `@tmp_date` and
1320                    `@end_date` in months when the month (`m`) in `@start_date`
1321                    is less than the month in `@end_date` or it is the same as
1322                    the month in `@end_date`.
1323
1324                      DAY(@start_date) > DAY(@end_date)
1325
1326                    If the condition is true, the query returns `1` (under a full
1327                    month).  Otherwise it returns `0` (full month).
1328   *************************************************************************** */
1329
1330  SELECT @months = DATEDIFF(m, @tmp_date, @end_date) - CASE
1331      WHEN DAY(@start_date) > DAY(@end_date)
1332        THEN 1
1333      ELSE 0
1334      END;
1335
1336
1337  /* ***************************************************************************
1338           9.03.11. We add the value of `@months` (`m`) to `@tmp_date` returning
1339                    1 or 0.
1340   *************************************************************************** */
1341
1342  SELECT @tmp_date = DATEADD(m, @months, @tmp_date);
1343
1344
1345  /* ***************************************************************************
1346           9.03.12. Below `@days` is assigned the difference of `@tmp_date` and
1347                    `@end_date` in days.
1348   *************************************************************************** */
1349
1350  SELECT @days = DATEDIFF(d, @tmp_date, @end_date);
1351
```

```
1352
1353 /* ************************************************************************
1354          9.03.13. We finally display the values for `@years`, `@months` and
1355                   `@days`.
1356
1357                    +---------------+-------+--------+------+
1358                    | Person's Name  | Years | Months | Days |
1359                    +---------------+-------+--------+------+
1360                    | Linus Torvalds | 53    |  3     | 28   |
1361                    +---------------+-------+--------+------+
1362
1363          9.03.14. You can also use the script to calculate your age or any
1364                   difference between any two dates by changing the values in
1365                   section #9.03.04.
1366
1367                   The value returned by `GETDATE()` when running this script
1368                   was 2021/11/29 and the end result will change according to
1369                   the current date when the script is run.
1370  ************************************************************************ */
1371
1372 SELECT @persons_name AS 'Person''s Name',       -- two single quotes (``) to
1373                                                 -- escape and show only one (`)
1374   @years AS 'Years',
1375   @months AS 'Months',
1376   @days AS 'Days';
1377
1378
1379 /* ************************************************************************
1380  https://folvera.commons.gc.cuny.edu/?p=1233
1381  ************************************************************************ */
```