```
/* ************************************************************************
                    DATABASE ADMINISTRATION FUNDAMENTALS:
                  INTRODUCTION TO STRUCTURED QUERY LANGUAGE
                    SF21SQL1001, 2021/11/02 - 2021/12/09
                  https://folvera.commons.gc.cuny.edu/?cat=29
 ************************************************************************

  SESSION #10 (2021/12/07): FINAL EXAMINATION

  1. Final examination
 ************************************************************************

 1. Change the tables in schema `AP3` using the following structure.

           +----------------------+--------------+--------------+---------+
           | SCHEMA & TABLE       | FIELD        | DATA TYPE    | NULL    |
           +----------------------+--------------+--------------+---------+
           | AP3.Employees        | EmpID        | VARCHAR(11)  | NOT NULL|
           |                      | FName        | VARCHAR(100) | NOT NULL|
           |                      | MName        | VARCHAR(100) |         |
           |                      | LName        | VARCHAR(100) | NOT NULL|
           |                      | Gender       | VARCHAR(1)   | NOT NULL|
           |                      | Address1     | VARCHAR(100) |         |
           |                      | Address2     | VARCHAR(100) |         |
           |                      | City         | VARCHAR(100) |         |
           |                      | State        | VARCHAR(2)   |         |
           |                      | PhoneNumber  | VARCHAR(10)  |         |
           |                      | ZipCode      | VARCHAR(10)  |         |
           |                      | JobDeptID    | INT          |         |
           |                      | JobTitleID   | INT          |         |
           |                      | JobSalary    | FLOAT        |         |
           |                      | StartDate    | DATE         | NOT NULL|
           +----------------------+--------------+--------------+---------+
           | AP3.JobDept          | JobDeptID    | INT          | NOT NULL|
           |                      | JobDept      | VARCHAR(100) | NOT NULL|
           |                      | Comments     | VARCHAR(255) |         |
           +----------------------+--------------+--------------+---------+
           | AP3.JobTitle         | JobTitleID   | INT          | NOT NULL|
           |                      | JobTitle     | VARCHAR(100) | NOT NULL|
           |                      | JobDept      | INT          | NOT NULL|
           +----------------------+--------------+--------------+---------+

 2. Update the `AP3.Employees` table.
 ************************************************************************ */

ALTER TABLE AP3.Employees                       -- 1. beginning of table
  ALTER COLUMN EmpID VARCHAR(11) NOT NULL;      --    `AP3.Employees`

ALTER TABLE AP3.Employees
  ALTER COLUMN FName VARCHAR(50) NOT NULL;

ALTER TABLE AP3.Employees
```

```
53    ALTER COLUMN MName VARCHAR(50);
54
55  ALTER TABLE AP3.Employees
56    ALTER COLUMN LName VARCHAR(50) NOT NULL;
57
58  ALTER TABLE AP3.Employees
59    ALTER COLUMN Address1 VARCHAR(100);
60
61  ALTER TABLE AP3.Employees
62    ALTER COLUMN Address2 VARCHAR(100);
63
64  ALTER TABLE AP3.Employees
65    ALTER COLUMN City VARCHAR(50);
66
67  ALTER TABLE AP3.Employees
68    ALTER COLUMN State VARCHAR(2);
69
70  ALTER TABLE AP3.Employees
71    ALTER COLUMN ZipCode VARCHAR(10);
72
73  ALTER TABLE AP3.Employees
74    ALTER COLUMN PhoneNumber VARCHAR(10);
75
76  ALTER TABLE AP3.Employees
77    ALTER COLUMN JobDeptID INT;
78
79  ALTER TABLE AP3.Employees
80    ALTER COLUMN JobTitleID INT;
81
82  ALTER TABLE AP3.Employees
83    ALTER COLUMN JobSalary FLOAT;
84
85  ALTER TABLE AP3.JobDept                         -- 2. beginning of table
86    ALTER COLUMN JobDeptID INT NOT NULL;          --    `AP3.JobDept`
87
88  ALTER TABLE AP3.JobDept
89    ALTER COLUMN JobDept VARCHAR(100) NOT NULL;
90
91  ALTER TABLE AP3.JobDept
92    ALTER COLUMN Comments VARCHAR(255);
93
94  ALTER TABLE AP3.JobTitle                        -- 3. beginning of table
95    ALTER COLUMN JobTitleID INT NOT NULL;         --    `AP3.JobTitle`
96
97  ALTER TABLE AP3.JobTitle
98    ALTER COLUMN JobTitle VARCHAR(100) NOT NULL;
99
100 ALTER TABLE AP3.JobTitle
101   ALTER COLUMN JobDeptID INT NOT NULL;
102
103
104 /* ***************************************************************************
```

```
105      2.1. Remove all empty spaces in fields `FName`, `LName`, `MName`, `City`
106           and `State`.
107
108      2.2. Use proper case in fields `FName`, `LName`, `MName` and `City`.
109
110           2.2.1. Since `MName` is only a character long, there is no need for
111                  `LEFT()` or `SUBSTRING()`.
112
113           2.2.2. Since the only value in `City` is `new york` in lower case, we
114                  can simply replace the original value for `New York` with the
115                  correct case.
116   ************************************************************************ */
117
118  UPDATE AP3.Employees
119    SET FName = RTRIM(LTRIM(
120    CONCAT(
121    UPPER(LEFT(FName, 1)),
122    LOWER(SUBSTRING(FName, 2, LEN(FName)-1))
123    )
124    ));
125
126  UPDATE AP3.Employees
127    SET LName = RTRIM(LTRIM(
128    CONCAT(
129    UPPER(LEFT(LName, 1)),
130    LOWER(SUBSTRING(LName, 2, LEN(LName)-1))
131    )
132    ));
133
134
135  /* ************************************************************************
136           2.2.3. Note that we cannot make `AP3.Employees.StartDate` `NOT NULL`
137                  till we push values into the column if any field is NULL.
138   ************************************************************************ */
139
140  UPDATE AP3.Employees                          -- 1. passing values of to
141  SET StartDate = GETDATE()                     --    columns where `StartDate`
142  WHERE StartDate IS NULL                       --    is NULL or has no value,
143    OR StartDate = '';                          --    in this case passing the
144                                                --    value retrieved from
145                                                --    `GETDATE()`
146
147  ALTER TABLE AP3.Employees                     -- 2. making column `NOT NULL`
148  ALTER COLUMN StartDate DATE NOT NULL;         --    now that `StartDate` has
149                                                --    values from `GETDATE()`
150
151
152  /* ************************************************************************
153      2.3. The updates above (#2.1 & #2.2) to `AP3.Employees` (same table) can
154           also be done in one (1) `UPDATE` statement with only one (1) `SET` and
155           commas separating each column.
156
```

```
157          2.3.1. Note that functions are called in the same order (innermost to
158                 outermost) as specified in each section (#2.1 & #2.2).
159
160          2.3.2. Since `MName` is only a character long, there is no need for
161                 `LEFT()` or `SUBSTRING()`.
162  ********************************************************************** */
163
164  UPDATE AP3.Employees
165    SET FName = RTRIM(LTRIM(FName)),
166        LName = RTRIM(LTRIM(LName)),
167        MName = RTRIM(LTRIM(MName)),
168        City  = RTRIM(LTRIM(City)),
169        State = UPPER(RTRIM(LTRIM(State)));
170
171
172  /* **********************************************************************
173          2.3.3. Since the only value in `City` is `new york` in lower case, we
174                 can simply replace the original value for `New York` with the
175                 correct case.
176   ********************************************************************** */
177
178  UPDATE AP3.Employees
179    SET City = 'New York'
180    WHERE City = 'new york';
181
182
183  /* **********************************************************************
184      2.3. Capitalize States.
185   ********************************************************************** */
186
187  UPDATE AP3.Employees
188    SET State = UPPER(RTRIM(LTRIM(State)));
189
190
191  /* **********************************************************************
192      2.4. Change data type of `JobSalary` from FLOAT to MONEY.
193   ********************************************************************** */
194
195  ALTER TABLE AP3.Employees
196    ALTER COLUMN JobSalary MONEY;
197
198
199  /* **********************************************************************
200   3. Add column `LastUpdateDate` to table `AP3.Employees` with data type DATE
201      and NOT NULL with today's DATE.
202
203      3.1. When you UPDATE a table, you cannot assign `NOT NULL`.  There is
204           always a way to do things in SQL and other programming.  You might
205           have to first create a new table and then populate it with the data
206           from the original table.
207
208      3.2. One alternative is using DATE function `GETDATE()`.
```

```
209
210     3.3. After adding column `LastUpdateDate` to table `AP3.Employees`, we pass
211          the value of the result of built-in function `GETDATE()`.
212
213     3.4. `GETDATE()` must be followed by an opening and closing parenthesis
214          (without parameters) to tell the system that `GETDATE()` is a function
215          and not confuse the system into believing that `GETDATE()` is a column
216          in a table).
217  ********************************************************************** */
218
219 ALTER TABLE AP3.Employees
220   ADD LastUpdateDate DATE;
221
222 UPDATE AP3.Employees
223   SET LastUpdateDate = GETDATE();
224
225
226 /* **********************************************************************
227  4. Create a view named `AP3.EmployeesVW` with all employee information with
228     the following fields without extra spaces.
229
230             +----------------------+----------------------------------------+
231             | FIELD(S)             | FORMATTING                             |
232             +----------------------+----------------------------------------+
233             | EmpID                | first seven (7) characters masked with |
234             |                      | `XXX-XX-`                              |
235             +----------------------+----------------------------------------+
236             | LName, FName MName   | returned in one field with a single    |
237             |                      | space between them, avoid NULLs if     |
238             |                      | `MName` is NULL;  no multiple spaces;  |
239             |                      | trimmed if needed;  capitalized with a |
240             |                      | comma between `LName` and `FName`      |
241             |                      | replacing empty spaces                 |
242             +----------------------+----------------------------------------+
243             | Gender               | capitalized                            |
244             +----------------------+----------------------------------------+
245             | Address1 Address2    | returned in one field with a single    |
246             |                      | space between them;  trimmed if needed |
247             +----------------------+----------------------------------------+
248             | City                 | first letter capitalized;  trimmed if  |
249             |                      | needed                                 |
250             +----------------------+----------------------------------------+
251             | State                | capitalized; trimmed if needed         |
252             +----------------------+----------------------------------------+
253             | ZipCode              | with missing Plus 4 as `-0001`         |
254             +----------------------+----------------------------------------+
255             | PhoneNumber          | formatted as `(XXX) XXX-XXXX`          |
256             +----------------------+----------------------------------------+
257             | JobDept              | using corresponding value from table   |
258             |                      | `JobDept`, not `JobDeptID`             |
259             +----------------------+----------------------------------------+
260             | JobTitle             | using corresponding value from table   |
```

```
            |                       | `JobTitle`, not `JobTitleID`           |
            +-----------------------+----------------------------------------+
            | JobSalary             | formatted as currency (`c`)            |
            +-----------------------+----------------------------------------+
            | StartDate             | formatted as date (`d`)                |
            +-----------------------+----------------------------------------+
            | LastUpdateDate        | new column updated with today's date;  |
            |                       | formatted as date (`d`)                |
            +-----------------------+----------------------------------------+
            | Comments              | no formatting                          |
            +-----------------------+----------------------------------------+

    4.1. When creating the view, only show employees hired after 12/30/1999 and
         before 6/1/2015 sorting the output by last name, first name, middle
         name and employee ID.

         4.1.1. When using an alias after concatenating fields, you would use
                the alias in `ORDER BY` (no `ORDER BY` in views).

         4.1.2. Before creating the view, it is good practice to first create
                the query.

         4.1.3. Add `XXX-XX-` to the last four characters in `EmpID`.

                4.1.3.1. The latter can be done using a user-defined function.

         4.1.4. Concatenate `LName`, `FName` and `MName` with spaces in between
                if we have a value for `MName` or `LName` and `FName` with a
                space in between if we have no value for `MName`.

                4.1.4.1. We can use a `CASE` clause to add added logic.

                4.1.4.2. The latter can be done using a user-defined function.

         4.1.5. Concatenate (put strings together) with space (` `) between
                `Address1` and `Address2` if we have a value in `Address2` or
                only call `Address1` if we have no value in `Address2`.

                4.1.5.1. We can use a `CASE` clause for added logic.

                4.1.5.2. The latter can be done using a user-defined function.

         4.1.6. Once again we have to concatenate strings.  In this case, we
                add dummy Plus 4 `-0001` (non-existent string in the database,
                hard-coded) before field `ZipCode`.

                4.1.6.1. The latter can be done using a user-defined function.

         4.1.7. We have to separate the parts of the phone number logically and
                present the phone number as `(XXX) XXX-XXXX`.

                4.1.7.1. We add `(` to surround the area code part of
```

```
313                         `PhoneNumber`.
314
315                 4.1.7.2. We call the first three (3) characters of
316                         `PhoneNumber` using `LEFT()` calling three (3) spaces.
317
318                 4.1.7.3. Then we add `) ` to close the first parenthesis and
319                         complete the way area codes are commonly displayed.
320
321                 4.1.7.4. Now we have to call the inside of `PhoneNumber` using
322                         `SUBSTRING()` calling first where we want to start
323                         (fourth characters) and how many characters from the
324                         first value we want to call -- next three (3)
325                         characters.
326
327                 4.1.7.5. We add a hyphen (`-`) to continue the way how phone
328                         numbers are normally presented.
329
330                 4.1.7.6. We finish by calling the last four (4) characters
331                         using `RIGHT()` and calling four (4) places.
332
333                 4.1.7.7. The latter can be done using a user-defined function.
334
335           4.1.8. Format all monetary values (`c`) with culture `en-us`
336                 (https://msdn.microsoft.com/en-us/library/hh213525.aspx).
337
338                 4.1.8.1. Note that formatting any numeric value returns a
339                         string.
340
341                 4.1.8.2. The latter can be done using a user-defined function.
342
343           4.1.9. Format all dates (`d`) with culture `en-us`
344                 (https://msdn.microsoft.com/en-us/library/hh213525.aspx).
345
346                 4.1.9.1. Note that formatting any numeric value returns a
347                         string.
348
349                 4.1.9.2. The latter can be done using a user-defined function.
350
351           4.1.10. Retrieve values for `AP3.Employees.StartDate` greater or equal
352                  to (`>=`) to `12/30/1999` and less than or equal to (`<=`) to
353                  `6/1/2015` (range of values).
354
355                           WHERE AP3.Employees.StartDate >= `12/30/1999`
356                             AND AP3.Employees.StartDate <= `6/1/2015`
357
358                  The better option is using `BETWEEN`, which is cleaner and
359                  easier to read.
360
361                           WHERE AP3.Employees.StartDate BETWEEN `12/30/1999`
362                             AND `6/1/2015`
363
364                  It also avoids the possibility of errors when using operators
```

```
365                  `>=` and `<=`.
366   ************************************************************************ */
367
368  SELECT DISTINCT                                  -- start of query (#4.1)
369    REPLACE(AP3.Employees.EmpID, LEFT(AP3.Employees.EmpID, 7), 'xxx-xx-')
370      AS EmpID,
371    CASE                                           -- 1. concatenating last, first
372      WHEN AP3.Employees.MName IS NOT NULL         --    and middle names; if not
373        OR AP3.Employees.MName <> ''               --    concatenating only first
374        THEN CONCAT (                              --    and last names
375            AP3.Employees.LName,
376            ', ',
377            AP3.Employees.FName,
378            ' ',
379            AP3.Employees.MName
380            )
381      ELSE CONCAT (
382          AP3.Employees.LName,
383          ', ',
384          AP3.Employees.FName
385          )
386      END AS Name,
387    AP3.Employees.Gender,
388    CASE
389      WHEN AP3.Employees.Address2 IS NOT NULL      -- 2. concatenating `Address1`
390        OR AP3.Employees.Address2 <> ''            --    and `Address2` when
391        THEN CONCAT (                              --    `Address2` is NOT NULL or
392            AP3.Employees.Address1,                --    an empty string; if not
393            ' ',                                   --    showing only the original
394            AP3.Employees.Address2                 --    value of `Address1`
395            )
396      ELSE AP3.Employees.Address1
397      END AS Address,
398    AP3.Employees.City,
399    CASE                                           -- 3. concatenating `ZipCode`
400      WHEN AP3.Employees.ZipCode IS NULL           --    and dummy Plus 4 `-0001`
401        OR AP3.Employees.ZipCode = ''              --    is NOT NULL, an empty
402        OR LEN(AP3.Employees.ZipCode) <> 5         --    string or longer than 5;
403        THEN ''                                    --    otherwise showing only
404      WHEN LEN(AP3.Employees.ZipCode) = 5          --    the original value of
405        THEN CONCAT (                              --    `ZipCode`
406            AP3.Employees.ZipCode,
407            '-0001'
408            )
409      ELSE AP3.Employees.ZipCode
410      END AS ZipCode,
411    CASE                                           -- 4. concatenating an opening
412      WHEN PhoneNumber <> ''                       --    parenthesis (`(`), three
413        OR PhoneNumber <> 10                       --    characters from the left
414        THEN CONCAT (                              --    of `PhoneNumber`, the
415            '(',                                   --    substring starting from
416            LEFT(PhoneNumber, 3),                  --    the 4th character taking
```

```
417             ') ',                                   --    3 characters (4th to
418             SUBSTRING(PhoneNumber, 4, 3),           --    6th), a closing
419             '-',                                    --    parenthesis and space
420             RIGHT(PhoneNumber, 4)                   --    (`( `), a hyphen and 4
421             )                                       --    characters from the
422       ELSE PhoneNumber,                             --    right; if not showing
423       END AS PhoneNumber,                           --    only the original value
424                                                     --    of `PhoneNumber`
425     AP3.JobDept.JobDept,
426     AP3.JobTitle.JobTitle,
427     FORMAT(AP3.Employees.JobSalary, 'c', 'en-us') -- 5. formatting dollar amounts
428       AS JobSalary,                                 --    as currency (`c`) with
429                                                     --    culture `en-us`
430     FORMAT(AP3.Employees.StartDate, 'd', 'en-us') -- 6. formatting dates as short
431       AS StartDate,                                 --    date (`d`) with culture
432                                                     --    `en-us`
433     FORMAT(AP3.Employees.LastUpdateDate,          -- 7. formatting dates as short
434       'd', 'en-us') AS LastUpdateDate,              --    date (`d`) with culture
435                                                     --    `en-us`
436     AP3.JobDept.Comments
437   FROM AP3.Employees
438   LEFT JOIN AP3.JobDept
439     ON AP3.Employees.JobDeptID = AP3.JobDept.JobDeptID
440   LEFT JOIN AP3.JobTitle
441     ON AP3.Employees.JobTitleID = AP3.JobTitle.JobTitleID
442   WHERE AP3.Employees.StartDate BETWEEN '12/30/1999'
443       AND '6/1/2015'
444   ORDER BY Name,
445     EmpID;                                          -- end of query (#4.1)
446
447
448   /* ***************************************************************************
449       4.2. Now we can create view `final.EmployeesVW` in database `ace` using
450            the query above.
451
452            4.2.1. Do not forget to create schema `final` in in database `ace`
453                   and to add database `SF21SQL1001` before schema `AP3` and the
454                   name of the tables.
455
456            4.4.2. Note that views in T-SQL cannot be created using `ORDER BY` so
457                   we have to remove the clause.
458
459            4.2.3. Therefore we are forced to remove the `ORDER BY` clause from
460                   our query.
461    *************************************************************************** */
462
463   CREATE SCHEMA final;                              -- creating schema `final` in
464                                                     -- database `ace` where we are
465                                                     -- creating view `EmployeesVW`
466
467   CREATE VIEW final.EmployeesVW                     -- 1. start of view
468   AS                                                --    `final.EmployeesVW`
```

```
(
    SELECT DISTINCT                                 -- 2. start of query (#4.1)
      REPLACE(SF21SQL1001.AP3.Employees.EmpID,
        LEFT(SF21SQL1001.AP3.Employees.EmpID, 7), 'xxx-xx-') AS EmpID,
      CASE
        WHEN SF21SQL1001.AP3.Employees.MName IS NOT NULL
          OR SF21SQL1001.AP3.Employees.MName <> ''
          THEN CONCAT (
              SF21SQL1001.AP3.Employees.LName,
              ', ',
              SF21SQL1001.AP3.Employees.FName,
              ' ',
              SF21SQL1001.AP3.Employees.MName
              )
        ELSE CONCAT (
            SF21SQL1001.AP3.Employees.LName,
            ', ',
            SF21SQL1001.AP3.Employees.FName
            )
        END AS Name,
      SF21SQL1001.AP3.Employees.Gender,
      CASE
        WHEN SF21SQL1001.AP3.Employees.Address2 IS NOT NULL
          OR SF21SQL1001.AP3.Employees.Address2 <> ''
          THEN CONCAT (
              SF21SQL1001.AP3.Employees.Address1,
              ' ',
              SF21SQL1001.AP3.Employees.Address2
              )
        ELSE SF21SQL1001.AP3.Employees.Address1
        END AS Address,
      SF21SQL1001.AP3.Employees.City,
      CASE
        WHEN SF21SQL1001.AP3.Employees.ZipCode IS NULL
          OR SF21SQL1001.AP3.Employees.ZipCode = ''
          OR LEN(SF21SQL1001.AP3.Employees.ZipCode) <> 5
          THEN ''
        WHEN LEN(SF21SQL1001.AP3.Employees.ZipCode) = 5
          THEN CONCAT (
              SF21SQL1001.AP3.Employees.ZipCode,
              '-0001'
              )
        ELSE SF21SQL1001.AP3.Employees.ZipCode
        END AS ZipCode,
      CASE
        WHEN PhoneNumber IS NOT NULL                -- 1. checking for NULL
          OR PhoneNumber <> ''                      -- 2. checking that it is not
                                                    --    an empty string
          OR PhoneNumber <> 10                      -- 3. checking for length
          OR PhoneNumber NOT LIKE ('(%)%-%')        -- 4. checking for format
          THEN CONCAT (
              '(',
```

```
521                 LEFT(PhoneNumber, 3),
522                 ') ',
523                 SUBSTRING(PhoneNumber, 4, 3),
524                 '-',
525                 RIGHT(PhoneNumber, 4)
526                 )
527           ELSE PhoneNumber
528           END AS PhoneNumber,
529         SF21SQL1001.AP3.JobDept.JobDept,
530         SF21SQL1001.AP3.JobTitle.JobTitle,
531         FORMAT(SF21SQL1001.AP3.Employees.JobSalary,
532           'c', 'en-us') AS JobSalary,
533         FORMAT(SF21SQL1001.AP3.Employees.StartDate,
534           'd', 'en-us') AS StartDate,
535         FORMAT(SF21SQL1001.AP3.Employees.LastUpdateDate,
536           'd', 'en-us') AS LastUpdateDate,
537         SF21SQL1001.AP3.JobDept.Comments
538       FROM SF21SQL1001.AP3.Employees
539       LEFT JOIN SF21SQL1001.AP3.JobDept
540         ON SF21SQL1001.AP3.Employees.JobDeptID =
541           SF21SQL1001.AP3.JobDept.JobDeptID
542       LEFT JOIN SF21SQL1001.AP3.JobTitle
543         ON SF21SQL1001.AP3.Employees.JobTitleID =
544           SF21SQL1001.AP3.JobTitle.JobTitleID
545       WHERE SF21SQL1001.AP3.Employees.StartDate BETWEEN '12/30/1999'
546           AND '6/1/2015'
547       -- ORDER BY Name,                          -- 3. no `ORDER BY` in views
548         -- EmpID                                 --    3.1. end of query (#4.1)
549       );                                        --    3.2. end of view (#4.2)
550
551
552  /* ***********************************************************************
553      4.3. To make sure that the view displays the correct data, we can call all
554           the values referred in view `final.EmployeesVW`.
555   *********************************************************************** */
556
557  SELECT *
558  FROM final.EmployeesVW;
559
560
561  /* ***********************************************************************
562   5. As a bonus, you can use functions to format, concatenate and other tasks
563      when writing the query that will be part of your view.
564
565      5.1. First we write the function to add `XXX-XX-` to the last four
566           characters in `EmpID` (#4.1.3).
567   *********************************************************************** */
568
569  CREATE FUNCTION final.EmpID_udf (@in_empid VARCHAR(15))
570  RETURNS VARCHAR(15)                           -- function returns...
571  AS
572    BEGIN
```

```
573     -- declaring output and passing value of `REPLACE` in-line
574     DECLARE @out_empid VARCHAR(15) = REPLACE(@in_empid,
575                                       LEFT(@in_empid, 7),
576                                       'xxx-xx-')
577     RETURN @out_empid
578   END;
579
580
581 /* ************************************************************************
582     5.2. Then we write the function to concatenate `LName`, `FName` and `MName`
583          with spaces in between if we have a value for `MName` or `LName` and
584          `FName` with a space in between if we have no value for `MName`
585          (#4.1.4).
586  ************************************************************************ */
587
588 CREATE FUNCTION final.fullname_udf (
589   @in_fname VARCHAR(50),
590   @in_mname VARCHAR(1),
591   @in_lname VARCHAR(50)
592   )                                          -- accepting three (3) values
593 RETURNS VARCHAR(101)                         -- function returns...
594 AS
595   BEGIN
596     DECLARE @out_fullname VARCHAR(101) = CASE
597         WHEN @in_mname IS NOT NULL
598           OR @in_mname <> ''
599           THEN CONCAT (
600               @in_lname,
601               ', ',
602               @in_fname,
603               ' ',
604               @in_mname
605               )
606         ELSE CONCAT (
607             @in_lname,
608             ', ',
609             @in_fname
610             )
611         END
612     RETURN @out_fullname
613   END;
614
615
616 /* ************************************************************************
617     5.3. Then we write the function to Concatenate with space between
618          `Address1` and `Address2` if we have a value in `Address2` or only
619          call `Address1` if we have no value in `Address2` (#4.1.5).
620  ************************************************************************ */
621
622 CREATE FUNCTION final.address2_udf (
623   @in_address1 VARCHAR(100),
624   @in_address2 VARCHAR(100)
```

```
625   )                                          -- accepting two (2) values
626 RETURNS VARCHAR(201)                         -- function returns...
627 AS
628   BEGIN
629     DECLARE @out_address VARCHAR(201) = CASE
630         WHEN @in_address2 IS NOT NULL
631           OR @in_address2 <> ''
632           THEN CONCAT (
633               @in_address1,
634               ' ',
635               @in_address2
636               )
637         ELSE @in_address1
638         END                                  -- closing `CASE`
639     RETURN @out_address
640   END;                                       -- closing `BEGIN` (function)
641
642
643 /* ***************************************************************************
644     5.4. Then we write the function to add dummy Plus 4 `-0001` (non-existent
645          string in the database, hard-coded) before field `ZipCode` (#4.1.6).
646  *************************************************************************** */
647
648 CREATE FUNCTION final.zipcode_udf (@in_zipcode VARCHAR(10))
649 RETURNS VARCHAR(10)                          -- function returns...
650 AS
651   BEGIN
652     DECLARE @out_zipcode VARCHAR(10) = CASE
653         WHEN @in_zipcode IS NULL
654           OR @in_zipcode = ''
655           OR LEN(@in_zipcode) <> 5
656           THEN ''
657         WHEN LEN(@in_zipcode) = 5
658           THEN CONCAT (
659               @in_zipcode,
660               '-0001'
661               )
662         ELSE @in_zipcode
663         END                                  -- closing `CASE`
664     RETURN @out_zipcode
665   END;                                       -- closing `BEGIN` (function)
666
667
668 /* ***************************************************************************
669     5.4. Then we write the function to to separate the parts of the phone
670          number logically and present the phone number as `(XXX) XXX-XXXX`
671          (#4.1.7).
672  *************************************************************************** */
673
674 CREATE FUNCTION final.phones_udf (@in_phone VARCHAR(15))
675 RETURNS VARCHAR(15)                          -- function returns...
676 AS
```

```
677   BEGIN
678     DECLARE @out_phone VARCHAR(15) = CASE
679         WHEN @in_phone IS NOT NULL
680           OR @in_phone <> ''
681           OR @in_phone <> 10
682           OR @in_phone NOT LIKE ('(%)%-%')
683           THEN CONCAT (
684               '(',
685               LEFT(@in_phone, 3),
686               ') ',
687               SUBSTRING(@in_phone, 4, 3),
688               '-',
689               RIGHT(@in_phone, 4)
690               )
691         ELSE @in_phone
692         END                                     -- closes `CASE`
693     RETURN @out_phone
694   END;                                          -- closes function
695
696
697 /* ***************************************************************************
698     5.5. Now we can re-write the view.
699  ************************************************************************** */
700
701 CREATE VIEW final.EmployeesVW
702 AS
703 (
704     SELECT DISTINCT
705       final.EmpID_udf(SF21SQL1001.AP3.Employees.EmpID) AS EmpID,
706       final.fullname_udf(SF21SQL1001.AP3.Employees.LName,
707               SF21SQL1001.AP3.Employees.FName,
708               SF21SQL1001.AP3.Employees.MName
709               ) AS Name,
710       SF21SQL1001.AP3.Employees.Gender,
711       final.address2_udf(
712               SF21SQL1001.AP3.Employees.Address1,
713               SF21SQL1001.AP3.Employees.Address2
714               ) AS Address,
715       SF21SQL1001.AP3.Employees.City,
716       final.zipcode_udf(SF21SQL1001.AP3.Employees.ZipCode) AS ZipCode,
717       final.phones_udf(PhoneNumber) AS PhoneNumber,
718       SF21SQL1001.AP3.JobDept.JobDept,
719       SF21SQL1001.AP3.JobTitle.JobTitle,
720       FORMAT(SF21SQL1001.AP3.Employees.JobSalary,
721         'c', 'en-us') AS JobSalary,
722       FORMAT(SF21SQL1001.AP3.Employees.StartDate,
723         'd', 'en-us') AS StartDate,
724       FORMAT(SF21SQL1001.AP3.Employees.LastUpdateDate,
725         'd', 'en-us') AS LastUpdateDate,
726       SF21SQL1001.AP3.JobDept.Comments
727     FROM SF21SQL1001.AP3.Employees
728     LEFT JOIN SF21SQL1001.AP3.JobDept
```

```
729         ON SF21SQL1001.AP3.Employees.JobDeptID =
730           SF21SQL1001.AP3.JobDept.JobDeptID
731       LEFT JOIN SF21SQL1001.AP3.JobTitle
732         ON SF21SQL1001.AP3.Employees.JobTitleID =
733           SF21SQL1001.AP3.JobTitle.JobTitleID
734       WHERE SF21SQL1001.AP3.Employees.StartDate BETWEEN '12/30/1999'
735           AND '6/1/2015'
736       );
737
738
739   /* ************************************************************************
740       5.6. As a bonus, there are other objects that can be added to tables.
741            Constraints like keys restrict the possibility of losing data by
742            dropping the wrong object or even inserting the wrong data into a
743            field.
744
745              ``A primary key, also called a primary keyword, is a key in a
746              relational database that is unique for each record.  It is a unique
747              identifier, such as a driver license number, telephone number
748              (including area code), or vehicle identification number (VIN).  A
749              relational database must always have one and only one primary key.
750              Primary keys typically appear as columns in relational database
751              tables.
752              The choice of a primary key in a relational database often depends
753              on the preference of the administrator.  It is possible to change
754              the primary key for a given database when the specific needs of the
755              users changes.  For example, the people in a town might be uniquely
756              identified according to their driver license numbers in one
757              application, but in another situation it might be more convenient to
758              identify them according to their telephone numbers.``
759              https://searchsqlserver.techtarget.com/definition/primary-key
760
761            The syntax is similar to that of any structure change of a table.  In
762            this case, we add a primary key (PK) constraint.
763
764                       ALTER TABLE table_name
765                       ADD CONSTRAINT pk_name
766                         PRIMARY KEY (column_name)
767
768              ``A foreign key is a column or columns of data in one table that
769              connects to the primary key data in the original table.
770              To ensure the links between foreign key and primary key tables
771              aren't broken, foreign key constraints can be created to prevent
772              actions that would damage the links between tables and prevent
773              erroneous data from being added to the foreign key column.``
774              https://searchoracle.techtarget.com/definition/foreign-key
775
776            Just like in the case of primary keys, we need to alter the table to
777            add a foreign key (FK) constraint.  The big difference is that we need
778            to indicate the primary key (PK), on which the FK depends on.
779
780                       ALTER TABLE child_table
```

```
                    ADD CONSTRAINT fk_name
                      FOREIGN KEY child_table_column
                      REFERENCES parent_table (parent_column)

         These two constraints are different as noted above.  The main
         difference is that a table can have only one primary key (PK) making
         it the identifier for the row and/or multiple foreign keys (FK's')
         referrencing other PK's in other tables in order to maintain
         dependency to other tables.

           ``Differences between primary and foreign keys
           A primary key in the original table, or parent table, can be
           targeted by multiple foreign keys from other `child` tables.  But a
           primary key does not necessarily have to be the target of any
           foreign keys.  A primary key is a column or a set of columns that
           identify a row in a table.  A foreign key, however, is in a table
           that is different from the table that the primary key must match.``
           https://searchoracle.techtarget.com/definition/foreign-key

         In the example below, we make a PK from `AP1.Vendors.VendorID` and a
         FK from `AP1.ContactUpdates.ContactUpdateID`.
 ************************************************************************ */

ALTER TABLE AP1.Vendors                          -- must make `VendorID` NOT
  ALTER COLUMN VendorID INT NOT NULL;            -- NULL before making it a key

ALTER TABLE AP1.Vendors                          -- altering table by adding
  ADD CONSTRAINT VendorID_PK                     -- constraint PRIMARY KEY with
  PRIMARY KEY (VendorID);                        -- `VendorID` in parenthesis


ALTER TABLE AP1.ContactUpdates                   -- must make `ContactUpdateID`
  ALTER COLUMN ContactUpdateID INT NOT NULL;     -- NOT NULL before making it a
                                                 -- key

ALTER TABLE AP1.ContactUpdates                   -- altering table by adding
  ADD CONSTRAINT ContactUpdateID_FK              -- constraint FOREIGN KEY with
  FOREIGN KEY (VendorID)                         -- `VendorID` in parenthesis
  REFERENCES AP1.Vendors (VendorID);             -- indicating the reference to
                                                 -- the PK in parent table


/* ************************************************************************
 7. What do you do now that the `Introduction to SQL` course has ended?

    7.1. Go to Microsoft Virtual Academy (https://mva.microsoft.com/) and
         continue learning including application to write reports
         (https://docs.microsoft.com/en-us/sql/reporting-services/), visualize
         and/or analyze data (https://powerbi.microsoft.com/).

    7.2. Register for the `Intermediate to SQL` course
         (https://www.campusce.net/bmcc/course/course.aspx?catId=33).
```

```
833
834     7.3. If you are interested in a career in data science, learn Python
835          (https://python.org/) and the multiple libraries available for data
836          analysis (https://folvera.commons.gc.cuny.edu/?s=python).
837
838     7.4. Contact me if you ever need help.
839
840  8. Before you leave, figure out what the following prints.
841
842     8.1. Change the value `your_name` with your name before running the code
843          below to create the procedure.  Then execute it.
844
845     8.2. If curious, visit http://ascii.cl/ for more information on ASCII
846          (http://whatis.techtarget.com/definition/ASCII-American-Standard-Code-
             for-Information-Interchange).
847  ********************************************************************** */
848
849  CREATE PROCEDURE final.message_sp
850  AS
851  BEGIN
852    DECLARE @yourName VARCHAR(50) = 'your_name',     -- param to hold your name
853            @CourseCd VARCHAR(15) = 'SF21SQL1001';  -- code for the SQL course
854    PRINT CONCAT ( CHAR(084), CHAR(104), CHAR(097), CHAR(110), CHAR(107),
855        CHAR(032), CHAR(121), CHAR(111), CHAR(117), CHAR(044), CHAR(032),
856        @yourName, CHAR(044), CHAR(032), CHAR(102), CHAR(111), CHAR(114),
857        CHAR(032), CHAR(116), CHAR(097), CHAR(107), CHAR(105), CHAR(110),
858        CHAR(103), CHAR(032), CHAR(099), CHAR(108), CHAR(097), CHAR(115),
859        CHAR(115), CHAR(032), @CourseCd, CHAR(046), CHAR(013), CHAR(083),
860        CHAR(101), CHAR(101), CHAR(032), CHAR(121), CHAR(111), CHAR(117),
861        CHAR(032), CHAR(105), CHAR(110), CHAR(032), CHAR(116), CHAR(104),
862        CHAR(101), CHAR(032), CHAR(105), CHAR(110), CHAR(116), CHAR(101),
863        CHAR(114), CHAR(109), CHAR(101), CHAR(100), CHAR(105), CHAR(097),
864        CHAR(116), CHAR(101), CHAR(032), CHAR(099), CHAR(108), CHAR(097),
865        CHAR(115), CHAR(115), CHAR(046), CHAR(013), CHAR(013), CHAR(070),
866        CHAR(046), CHAR(079), CHAR(108), CHAR(118), CHAR(101), CHAR(114),
867        CHAR(097), CHAR(032), CHAR(040), CHAR(102), CHAR(111), CHAR(108),
868        CHAR(118), CHAR(101), CHAR(114), CHAR(097), CHAR(064), CHAR(098),
869        CHAR(109), CHAR(099), CHAR(099), CHAR(046), CHAR(099), CHAR(117),
870        CHAR(110), CHAR(121), CHAR(046), CHAR(101), CHAR(100), CHAR(117),
871        CHAR(041), CHAR(013), CHAR(104), CHAR(116), CHAR(116), CHAR(112),
872        CHAR(058), CHAR(047), CHAR(047), CHAR(102), CHAR(111), CHAR(108),
873        CHAR(118), CHAR(101), CHAR(114), CHAR(097), CHAR(046), CHAR(099),
874        CHAR(111), CHAR(109), CHAR(109), CHAR(111), CHAR(110), CHAR(115),
875        CHAR(046), CHAR(103), CHAR(099), CHAR(046), CHAR(099), CHAR(117),
876        CHAR(110), CHAR(121), CHAR(046), CHAR(101), CHAR(100), CHAR(117),
877        CHAR(047) );                                -- all characters in ASCII
878  END;
879
880  EXEC final.message_sp;
881  /* **********************************************************************
882   https://folvera.commons.gc.cuny.edu/?p=1065
883   ********************************************************************** */
```