```
/* ************************************************************************
        CUNY ACE UPSKILLING:  INTRODUCTION TO STRUCTURED QUERY LANGUAGE
                      SF21JOB#2, 2021/11/08 to 2021/12/13
                  https://folvera.commons.gc.cuny.edu/?cat=30
 ************************************************************************

  SESSION #10 (2021/12/08): 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
  ALTER COLUMN MName VARCHAR(50);
```

```
 53
 54 ALTER TABLE AP3.Employees
 55   ALTER COLUMN LName VARCHAR(50) NOT NULL;
 56
 57 ALTER TABLE AP3.Employees
 58   ALTER COLUMN Address1 VARCHAR(100);
 59
 60 ALTER TABLE AP3.Employees
 61   ALTER COLUMN Address2 VARCHAR(100);
 62
 63 ALTER TABLE AP3.Employees
 64   ALTER COLUMN City VARCHAR(50);
 65
 66 ALTER TABLE AP3.Employees
 67   ALTER COLUMN State VARCHAR(2);
 68
 69 ALTER TABLE AP3.Employees
 70   ALTER COLUMN ZipCode VARCHAR(10);
 71
 72 ALTER TABLE AP3.Employees
 73   ALTER COLUMN PhoneNumber VARCHAR(10);
 74
 75 ALTER TABLE AP3.Employees
 76   ALTER COLUMN JobDeptID INT;
 77
 78 ALTER TABLE AP3.Employees
 79   ALTER COLUMN JobTitleID INT;
 80
 81 ALTER TABLE AP3.Employees
 82   ALTER COLUMN JobSalary FLOAT;
 83
 84 ALTER TABLE AP3.JobDept                          -- 2. beginning of table
 85   ALTER COLUMN JobDeptID INT NOT NULL;          --    `AP3.JobDept`
 86
 87 ALTER TABLE AP3.JobDept
 88   ALTER COLUMN JobDept VARCHAR(100) NOT NULL;
 89
 90 ALTER TABLE AP3.JobDept
 91   ALTER COLUMN Comments VARCHAR(255);
 92
 93 ALTER TABLE AP3.JobTitle                         -- 3. beginning of table
 94   ALTER COLUMN JobTitleID INT NOT NULL;         --    `AP3.JobTitle`
 95
 96 ALTER TABLE AP3.JobTitle
 97   ALTER COLUMN JobTitle VARCHAR(100) NOT NULL;
 98
 99 ALTER TABLE AP3.JobTitle
100   ALTER COLUMN JobDeptID INT NOT NULL;
101
102
103 /* ***************************************************************************
104     2.1. Remove all empty spaces in fields `FName`, `LName`, `MName`, `City`
```

```
105          and `State`.
106
107     2.2. Use proper case in fields `FName`, `LName`, `MName` and `City`.
108
109          2.2.1. Since `MName` is only a character long, there is no need for
110                 `LEFT()` or `SUBSTRING()`.
111
112          2.2.2. Since the only value in `City` is `new york` in lower case, we
113                 can simply replace the original value for `New York` with the
114                 correct case.
115  ************************************************************************ */
116
117 UPDATE AP3.Employees
118   SET FName = RTRIM(LTRIM(
119   CONCAT(
120   UPPER(LEFT(FName, 1)),
121   LOWER(SUBSTRING(FName, 2, LEN(FName)-1))
122   )
123   ));
124
125 UPDATE AP3.Employees
126   SET LName = RTRIM(LTRIM(
127   CONCAT(
128   UPPER(LEFT(LName, 1)),
129   LOWER(SUBSTRING(LName, 2, LEN(LName)-1))
130   )
131   ));
132
133
134 /* ************************************************************************
135          2.2.3. Note that we cannot make `AP3.Employees.StartDate` `NOT NULL`
136                 till we push values into the column if any field is NULL.
137  ************************************************************************ */
138
139 UPDATE AP3.Employees                          -- 1. passing values of to
140 SET StartDate = GETDATE()                     --    columns where `StartDate`
141 WHERE StartDate IS NULL                       --    is NULL or has no value,
142   OR StartDate = '';                          --    in this case passing the
143                                               --    value retrieved from
144                                               --    `GETDATE()`
145
146 ALTER TABLE AP3.Employees                     -- 2. making column `NOT NULL`
147 ALTER COLUMN StartDate DATE NOT NULL;         --    now that `StartDate` has
148                                               --    values from `GETDATE()`
149
150
151 /* ************************************************************************
152     2.3. The updates above (#2.1 & #2.2) to `AP3.Employees` (same table) can
153          also be done in one (1) `UPDATE` statement with only one (1) `SET` and
154          commas separating each column.
155
156          2.3.1. Note that functions are called in the same order (innermost to
```

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

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

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

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

```
417             SUBSTRING(PhoneNumber, 4, 3),          --    6th), a closing
418             '-',                                   --    parenthesis and space
419             RIGHT(PhoneNumber, 4)                  --    (`( `), a hyphen and 4
420             )                                      --    characters from the
421       ELSE PhoneNumber,                            --    right; if not showing
422       END AS PhoneNumber,                          --    only the original value
423                                                    --    of `PhoneNumber`
424     AP3.JobDept.JobDept,
425     AP3.JobTitle.JobTitle,
426     FORMAT(AP3.Employees.JobSalary, 'c', 'en-us') -- 5. formatting dollar amounts
427       AS JobSalary,                                --    as currency (`c`) with
428                                                    --    culture `en-us`
429     FORMAT(AP3.Employees.StartDate, 'd', 'en-us') -- 6. formatting dates as short
430       AS StartDate,                                --    date (`d`) with culture
431                                                    --    `en-us`
432     FORMAT(AP3.Employees.LastUpdateDate,           -- 7. formatting dates as short
433       'd', 'en-us') AS LastUpdateDate,             --    date (`d`) with culture
434                                                    --    `en-us`
435     AP3.JobDept.Comments
436   FROM AP3.Employees
437   LEFT JOIN AP3.JobDept
438     ON AP3.Employees.JobDeptID = AP3.JobDept.JobDeptID
439   LEFT JOIN AP3.JobTitle
440     ON AP3.Employees.JobTitleID = AP3.JobTitle.JobTitleID
441   WHERE AP3.Employees.StartDate BETWEEN '12/30/1999'
442       AND '6/1/2015'
443   ORDER BY Name,
444     EmpID;                                         -- end of query (#4.1)
445
446
447   /* ***************************************************************************
448       4.2. Now we can create view `final.EmployeesVW` in database `ace` using
449            the query above.
450
451            4.2.1. Do not forget to create schema `final` in in database `ace`
452                   and to add database `SF21JOB2` before schema `AP3` and the
453                   name of the tables.
454
455            4.4.2. Note that views in T-SQL cannot be created using `ORDER BY` so
456                   we have to remove the clause.
457
458            4.2.3. Therefore we are forced to remove the `ORDER BY` clause from
459                   our query.
460    *************************************************************************** */
461
462   CREATE SCHEMA final;                             -- creating schema `final` in
463                                                    -- database `ace` where we are
464                                                    -- creating view `EmployeesVW`
465
466   CREATE VIEW final.EmployeesVW                    -- 1. start of view
467   AS                                               --    `final.EmployeesVW`
468   (
```

```
469     SELECT DISTINCT                                 -- 2. start of query (#4.1)
470       REPLACE(SF21JOB2.AP3.Employees.EmpID,
471         LEFT(SF21JOB2.AP3.Employees.EmpID, 7), 'xxx-xx-') AS EmpID,
472       CASE
473         WHEN SF21JOB2.AP3.Employees.MName IS NOT NULL
474           OR SF21JOB2.AP3.Employees.MName <> ''
475           THEN CONCAT (
476               SF21JOB2.AP3.Employees.LName,
477               ', ',
478               SF21JOB2.AP3.Employees.FName,
479               ' ',
480               SF21JOB2.AP3.Employees.MName
481               )
482         ELSE CONCAT (
483             SF21JOB2.AP3.Employees.LName,
484             ', ',
485             SF21JOB2.AP3.Employees.FName
486             )
487         END AS Name,
488       SF21JOB2.AP3.Employees.Gender,
489       CASE
490         WHEN SF21JOB2.AP3.Employees.Address2 IS NOT NULL
491           OR SF21JOB2.AP3.Employees.Address2 <> ''
492           THEN CONCAT (
493               SF21JOB2.AP3.Employees.Address1,
494               ' ',
495               SF21JOB2.AP3.Employees.Address2
496               )
497         ELSE SF21JOB2.AP3.Employees.Address1
498         END AS Address,
499       SF21JOB2.AP3.Employees.City,
500       CASE
501         WHEN SF21JOB2.AP3.Employees.ZipCode IS NULL
502           OR SF21JOB2.AP3.Employees.ZipCode = ''
503           OR LEN(SF21JOB2.AP3.Employees.ZipCode) <> 5
504           THEN ''
505         WHEN LEN(SF21JOB2.AP3.Employees.ZipCode) = 5
506           THEN CONCAT (
507               SF21JOB2.AP3.Employees.ZipCode,
508               '-0001'
509               )
510         ELSE SF21JOB2.AP3.Employees.ZipCode
511         END AS ZipCode,
512       CASE
513         WHEN PhoneNumber IS NOT NULL            -- 1. checking for NULL
514           OR PhoneNumber <> ''                  -- 2. checking that it is not
515                                                 --    an empty string
516           OR PhoneNumber <> 10                  -- 3. checking for length
517           OR PhoneNumber NOT LIKE ('(%)%-%')    -- 4. checking for format
518           THEN CONCAT (
519               '(',
520               LEFT(PhoneNumber, 3),
```

```
521                ') ',
522                SUBSTRING(PhoneNumber, 4, 3),
523                '-',
524                RIGHT(PhoneNumber, 4)
525                )
526          ELSE PhoneNumber
527          END AS PhoneNumber,
528        SF21JOB2.AP3.JobDept.JobDept,
529        SF21JOB2.AP3.JobTitle.JobTitle,
530        FORMAT(SF21JOB2.AP3.Employees.JobSalary,
531          'c', 'en-us') AS JobSalary,
532        FORMAT(SF21JOB2.AP3.Employees.StartDate,
533          'd', 'en-us') AS StartDate,
534        FORMAT(SF21JOB2.AP3.Employees.LastUpdateDate,
535          'd', 'en-us') AS LastUpdateDate,
536        SF21JOB2.AP3.JobDept.Comments
537      FROM SF21JOB2.AP3.Employees
538      LEFT JOIN SF21JOB2.AP3.JobDept
539        ON SF21JOB2.AP3.Employees.JobDeptID =
540          SF21JOB2.AP3.JobDept.JobDeptID
541      LEFT JOIN SF21JOB2.AP3.JobTitle
542        ON SF21JOB2.AP3.Employees.JobTitleID =
543          SF21JOB2.AP3.JobTitle.JobTitleID
544      WHERE SF21JOB2.AP3.Employees.StartDate BETWEEN '12/30/1999'
545          AND '6/1/2015'
546      -- ORDER BY Name,                          -- 3. no `ORDER BY` in views
547        -- EmpID                                 --    3.1. end of query (#4.1)
548      );                                        --    3.2. end of view (#4.2)
549
550
551  /* ***************************************************************************
552      4.3. To make sure that the view displays the correct data, we can call all
553           the values referred in view `final.EmployeesVW`.
554   *************************************************************************** */
555
556  SELECT *
557  FROM final.EmployeesVW;
558
559
560  /* ***************************************************************************
561   5. As a bonus, you can use functions to format, concatenate and other tasks
562      when writing the query that will be part of your view.
563
564      5.1. First we write the function to add `XXX-XX-` to the last four
565           characters in `EmpID` (#4.1.3).
566   *************************************************************************** */
567
568  CREATE FUNCTION final.EmpID_udf (@in_empid VARCHAR(15))
569  RETURNS VARCHAR(15)                           -- function returns...
570  AS
571    BEGIN
572      -- declaring output and passing value of `REPLACE` in-line
```

```
573     DECLARE @out_empid VARCHAR(15) = REPLACE(@in_empid,
574                                       LEFT(@in_empid, 7),
575                                       'xxx-xx-')
576     RETURN @out_empid
577   END;
578
579
580 /* ************************************************************************
581     5.2. Then we write the function to concatenate `LName`, `FName` and `MName`
582          with spaces in between if we have a value for `MName` or `LName` and
583          `FName` with a space in between if we have no value for `MName`
584          (#4.1.4).
585  ************************************************************************ */
586
587 CREATE FUNCTION final.fullname_udf (
588   @in_fname VARCHAR(50),
589   @in_mname VARCHAR(1),
590   @in_lname VARCHAR(50)
591   )                                          -- accepting three (3) values
592 RETURNS VARCHAR(101)                         -- function returns...
593 AS
594   BEGIN
595     DECLARE @out_fullname VARCHAR(101) = CASE
596         WHEN @in_mname IS NOT NULL
597           OR @in_mname <> ''
598           THEN CONCAT (
599               @in_lname,
600               ', ',
601               @in_fname,
602               ' ',
603               @in_mname
604               )
605         ELSE CONCAT (
606             @in_lname,
607             ', ',
608             @in_fname
609             )
610         END
611     RETURN @out_fullname
612   END;
613
614
615 /* ************************************************************************
616     5.3. Then we write the function to Concatenate with space between
617          `Address1` and `Address2` if we have a value in `Address2` or only
618          call `Address1` if we have no value in `Address2` (#4.1.5).
619  ************************************************************************ */
620
621 CREATE FUNCTION final.address2_udf (
622   @in_address1 VARCHAR(100),
623   @in_address2 VARCHAR(100)
624   )                                          -- accepting two (2) values
```

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

```
677     DECLARE @out_phone VARCHAR(15) = CASE
678         WHEN @in_phone IS NOT NULL
679           OR @in_phone <> ''
680           OR @in_phone <> 10
681           OR @in_phone NOT LIKE ('(%)%-%')
682           THEN CONCAT (
683               '(',
684               LEFT(@in_phone, 3),
685               ') ',
686               SUBSTRING(@in_phone, 4, 3),
687               '-',
688               RIGHT(@in_phone, 4)
689               )
690         ELSE @in_phone
691         END                                     -- closes `CASE`
692     RETURN @out_phone
693   END;                                          -- closes function
694
695
696 /* ***************************************************************************
697     5.5. Now we can re-write the view.
698  *************************************************************************** */
699
700 CREATE VIEW final.EmployeesVW
701 AS
702 (
703     SELECT DISTINCT
704       final.EmpID_udf(SF21JOB2.AP3.Employees.EmpID) AS EmpID,
705       final.fullname_udf(SF21JOB2.AP3.Employees.LName,
706               SF21JOB2.AP3.Employees.FName,
707               SF21JOB2.AP3.Employees.MName
708               ) AS Name,
709       SF21JOB2.AP3.Employees.Gender,
710       final.address2_udf(
711               SF21JOB2.AP3.Employees.Address1,
712               SF21JOB2.AP3.Employees.Address2
713               ) AS Address,
714       SF21JOB2.AP3.Employees.City,
715       final.zipcode_udf(SF21JOB2.AP3.Employees.ZipCode) AS ZipCode,
716       final.phones_udf(PhoneNumber) AS PhoneNumber,
717       SF21JOB2.AP3.JobDept.JobDept,
718       SF21JOB2.AP3.JobTitle.JobTitle,
719       FORMAT(SF21JOB2.AP3.Employees.JobSalary,
720         'c', 'en-us') AS JobSalary,
721       FORMAT(SF21JOB2.AP3.Employees.StartDate,
722         'd', 'en-us') AS StartDate,
723       FORMAT(SF21JOB2.AP3.Employees.LastUpdateDate,
724         'd', 'en-us') AS LastUpdateDate,
725       SF21JOB2.AP3.JobDept.Comments
726     FROM SF21JOB2.AP3.Employees
727     LEFT JOIN SF21JOB2.AP3.JobDept
728       ON SF21JOB2.AP3.Employees.JobDeptID =
```

```
729         SF21JOB2.AP3.JobDept.JobDeptID
730     LEFT JOIN SF21JOB2.AP3.JobTitle
731       ON SF21JOB2.AP3.Employees.JobTitleID =
732         SF21JOB2.AP3.JobTitle.JobTitleID
733     WHERE SF21JOB2.AP3.Employees.StartDate BETWEEN '12/30/1999'
734         AND '6/1/2015'
735     );
736
737
738 /* ************************************************************************
739     5.6. As a bonus, there are other objects that can be added to tables.
740          Constraints like keys restrict the possibility of losing data by
741          dropping the wrong object or even inserting the wrong data into a
742          field.
743
744            ``A primary key, also called a primary keyword, is a key in a
745            relational database that is unique for each record.  It is a unique
746            identifier, such as a driver license number, telephone number
747            (including area code), or vehicle identification number (VIN).  A
748            relational database must always have one and only one primary key.
749            Primary keys typically appear as columns in relational database
750            tables.
751            The choice of a primary key in a relational database often depends
752            on the preference of the administrator.  It is possible to change
753            the primary key for a given database when the specific needs of the
754            users changes.  For example, the people in a town might be uniquely
755            identified according to their driver license numbers in one
756            application, but in another situation it might be more convenient to
757            identify them according to their telephone numbers.``
758            https://searchsqlserver.techtarget.com/definition/primary-key
759
760          The syntax is similar to that of any structure change of a table.  In
761          this case, we add a primary key (PK) constraint.
762
763                      ALTER TABLE table_name
764                      ADD CONSTRAINT pk_name
765                        PRIMARY KEY (column_name)
766
767            ``A foreign key is a column or columns of data in one table that
768            connects to the primary key data in the original table.
769            To ensure the links between foreign key and primary key tables
770            aren't broken, foreign key constraints can be created to prevent
771            actions that would damage the links between tables and prevent
772            erroneous data from being added to the foreign key column.``
773            https://searchoracle.techtarget.com/definition/foreign-key
774
775          Just like in the case of primary keys, we need to alter the table to
776          add a foreign key (FK) constraint.  The big difference is that we need
777          to indicate the primary key (PK), on which the FK depends on.
778
779                      ALTER TABLE child_table
780                      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      7.3. If you are interested in a career in data science, learn Python
834           (https://python.org/) and the multiple libraries available for data
835           analysis (https://folvera.commons.gc.cuny.edu/?s=python).
836
837      7.4. Contact me if you ever need help.
838
839   8. Before you leave, figure out what the following prints.
840
841      8.1. Change the value `your_name` with your name before running the code
842           below to create the procedure.  Then execute it.
843
844      8.2. If curious, visit http://ascii.cl/ for more information on ASCII
845           (http://whatis.techtarget.com/definition/ASCII-American-Standard-Code-
              for-Information-Interchange).
846   ********************************************************************** */
847
848   CREATE PROCEDURE final.message_sp
849   AS
850   BEGIN
851     DECLARE @yourName VARCHAR(50) = 'your_name',     -- param to hold your name
852             @CourseCd VARCHAR(15) = 'SF21JOB2';      -- code for the SQL course
853     PRINT CONCAT ( CHAR(084), CHAR(104), CHAR(097), CHAR(110), CHAR(107),
854         CHAR(032), CHAR(121), CHAR(111), CHAR(117), CHAR(044), CHAR(032),
855         @yourName, CHAR(044), CHAR(032), CHAR(102), CHAR(111), CHAR(114),
856         CHAR(032), CHAR(116), CHAR(097), CHAR(107), CHAR(105), CHAR(110),
857         CHAR(103), CHAR(032), CHAR(099), CHAR(108), CHAR(097), CHAR(115),
858         CHAR(115), CHAR(032), @CourseCd, CHAR(046), CHAR(013), CHAR(083),
859         CHAR(101), CHAR(101), CHAR(032), CHAR(121), CHAR(111), CHAR(117),
860         CHAR(032), CHAR(105), CHAR(110), CHAR(032), CHAR(116), CHAR(104),
861         CHAR(101), CHAR(032), CHAR(105), CHAR(110), CHAR(116), CHAR(101),
862         CHAR(114), CHAR(109), CHAR(101), CHAR(100), CHAR(105), CHAR(097),
863         CHAR(116), CHAR(101), CHAR(032), CHAR(099), CHAR(108), CHAR(097),
864         CHAR(115), CHAR(115), CHAR(046), CHAR(013), CHAR(013), CHAR(070),
865         CHAR(046), CHAR(079), CHAR(108), CHAR(118), CHAR(101), CHAR(114),
866         CHAR(097), CHAR(032), CHAR(040), CHAR(102), CHAR(111), CHAR(108),
867         CHAR(118), CHAR(101), CHAR(114), CHAR(097), CHAR(064), CHAR(098),
868         CHAR(109), CHAR(099), CHAR(099), CHAR(046), CHAR(099), CHAR(117),
869         CHAR(110), CHAR(121), CHAR(046), CHAR(101), CHAR(100), CHAR(117),
870         CHAR(041), CHAR(013), CHAR(104), CHAR(116), CHAR(116), CHAR(112),
871         CHAR(058), CHAR(047), CHAR(047), CHAR(102), CHAR(111), CHAR(108),
872         CHAR(118), CHAR(101), CHAR(114), CHAR(097), CHAR(046), CHAR(099),
873         CHAR(111), CHAR(109), CHAR(109), CHAR(111), CHAR(110), CHAR(115),
874         CHAR(046), CHAR(103), CHAR(099), CHAR(046), CHAR(099), CHAR(117),
875         CHAR(110), CHAR(121), CHAR(046), CHAR(101), CHAR(100), CHAR(117),
876         CHAR(047) );                                 -- all characters in ASCII
877   END;
878
879   EXEC final.message_sp;
880
881   /* **********************************************************************
882    https://folvera.commons.gc.cuny.edu/?p=1068
883    ********************************************************************** */
```