```
/* ************************************************************************
          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 #10 (2023/05/03): 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
118  UPDATE AP3.Employees
119  SET FName = CONCAT (
120      UPPER(LEFT(LTRIM(RTRIM(FName)), 1)),
121      LOWER(SUBSTRING(LTRIM(RTRIM(FName)), 2, LEN(LTRIM(RTRIM(FName))) - 1))
122      );
123
124  UPDATE AP3.Employees
125  SET LName = CONCAT (
126      UPPER(LEFT(LTRIM(RTRIM(LName)), 1)),
127      LOWER(SUBSTRING(LTRIM(RTRIM(LName)), 2, LEN(LTRIM(RTRIM(LName))) - 1))
128      );
129
130
131  /* ********************************************************************
132                 The latter can be combined setting values to multiple fields in
133                 one `SET` clause.
134   ********************************************************************** */
135
136  UPDATE AP3.Employees
137  SET FName = CONCAT (
138      UPPER(LEFT(LTRIM(RTRIM(FName)), 1)),
139      LOWER(SUBSTRING(LTRIM(RTRIM(FName)), 2, LEN(LTRIM(RTRIM(FName))) - 1))
140      ),                                          -- 1. if not do so in line #179
141    MName = UPPER(LTRIM(RTRIM(MName))),           -- 2. if not do so in line #180
142    LName = CONCAT (
143      UPPER(LEFT(LTRIM(RTRIM(LName)), 1)),
144      LOWER(SUBSTRING(LTRIM(RTRIM(LName)), 2, LEN(LTRIM(RTRIM(LName))) - 1))
145      );                                          -- 3. if not do so in line #180
146
147
148  /* ********************************************************************
149          2.2.3. Note that we cannot make `AP3.Employees.StartDate` `NOT NULL`
150                 till we push values into the column if any field is NULL.
151   ********************************************************************** */
152
153  UPDATE AP3.Employees                            -- 1. passing values of to
154  SET StartDate = GETDATE()                       --    columns where `StartDate`
155  WHERE StartDate IS NULL                         --    is NULL or has no value,
156    OR StartDate = '';                            --    in this case passing the
```

```
157                                                 --    value retrieved from
158                                                 --    `GETDATE()`
159
160 ALTER TABLE AP3.Employees                        -- 2. making column `NOT NULL`
161 ALTER COLUMN StartDate DATE NOT NULL;            --    now that `StartDate` has
162                                                 --    values from `GETDATE()`
163
164
165 /* ************************************************************************
166     2.3. The updates above (#2.1 & #2.2) to `AP3.Employees` (same table) can
167          also be done in one (1) `UPDATE` statement with only one (1) `SET` and
168          commas separating each column.
169
170          2.3.1. Note that functions are called in the same order (innermost to
171                 outermost) as specified in each section (#2.1 & #2.2).
172
173          2.3.2. Since `MName` is only a character long, there is no need for
174                 `LEFT()` or `SUBSTRING()`.
175 ************************************************************************ */
176
177 UPDATE AP3.Employees
178 SET FName = RTRIM(LTRIM(FName)),                -- 1. if not done so in lines
179                                                 --    138 to 140
180   LName = RTRIM(LTRIM(LName)),                  -- 2. if not done so in line
181                                                 --    141
182   MName = UPPER(RTRIM(LTRIM(MName))),           -- 3. if not done so in lines
183                                                 --    142 to 144
184   City = RTRIM(LTRIM(City)),
185   State = UPPER(RTRIM(LTRIM(State)));
186
187
188 /* ************************************************************************
189          2.3.3. Since the only value in `City` is `new york` in lower case, we
190                 can simply replace the original value for `New York` with the
191                 correct case.
192  ************************************************************************ */
193
194 UPDATE AP3.Employees
195 SET City = 'New York'
196 WHERE City = 'new york';
197
198
199 /* ************************************************************************
200     2.3. Capitalize States.
201  ************************************************************************ */
202
203 UPDATE AP3.Employees
204 SET State = UPPER(RTRIM(LTRIM(State)));
205
206
207 /* ************************************************************************
208     2.4. Change data type of `JobSalary` from FLOAT to MONEY.
```

```
209  ********************************************************************** */
210
211  ALTER TABLE AP3.Employees
212    ALTER COLUMN JobSalary MONEY;
213
214
215  /* *********************************************************************
216   3. Add column `LastUpdateDate` to table `AP3.Employees` with data type DATE
217      and NOT NULL with today's DATE.
218
219      3.1. When you UPDATE a table, you cannot assign `NOT NULL`.  There is
220           always a way to do things in SQL and other programming.  You might
221           have to first create a new table and then populate it with the data
222           from the original table.
223
224      3.2. One alternative is using DATE function `GETDATE()`.
225
226      3.3. After adding column `LastUpdateDate` to table `AP3.Employees`, we pass
227           the value of the result of built-in function `GETDATE()`.
228
229      3.4. `GETDATE()` must be followed by an opening and closing parenthesis
230           (without parameters) to tell the system that `GETDATE()` is a function
231           and not confuse the system into believing that `GETDATE()` is a column
232           in a table).
233   ********************************************************************** */
234
235  ALTER TABLE AP3.Employees
236    ADD LastUpdateDate DATE;
237
238  UPDATE AP3.Employees
239  SET LastUpdateDate = GETDATE();
240
241
242  /* *********************************************************************
243   4. Create a view named `AP3.EmployeesVW` with all employee information with
244      the following fields without extra spaces.
245
246             +----------------------+----------------------------------------+
247             | FIELD(S)             | FORMATTING                             |
248             +----------------------+----------------------------------------+
249             | EmpID                | first seven (7) characters masked with |
250             |                      | `XXX-XX-`                              |
251             +----------------------+----------------------------------------+
252             | LName, FName MName   | returned in one field with a single    |
253             |                      | space between them, avoid NULLs if     |
254             |                      | `MName` is NULL;  no multiple spaces;  |
255             |                      | trimmed if needed;  capitalized with a |
256             |                      | comma between `LName` and `FName`      |
257             |                      | replacing empty spaces                 |
258             +----------------------+----------------------------------------+
259             | Gender               | capitalized                            |
260             +----------------------+----------------------------------------+
```

Address1 Address2	returned in one field with a single space between them; trimmed if needed
City	first letter capitalized; trimmed if needed
State	capitalized; trimmed if needed
ZipCode	with missing Plus 4 as `-0001`
PhoneNumber	formatted as `(XXX) XXX-XXXX`
JobDept	using corresponding value from table `JobDept`, not `JobDeptID`
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

```
313                 only call `Address1` if we have no value in `Address2`.
314
315                 4.1.5.1. We can use a `CASE` clause for added logic.
316
317                 4.1.5.2. The latter can be done using a user-defined function.
318
319           4.1.6. Once again we have to concatenate strings.  In this case, we
320                 add dummy Plus 4 `-0001` (non-existent string in the database,
321                 hard-coded) before field `ZipCode`.
322
323                 4.1.6.1. The latter can be done using a user-defined function.
324
325           4.1.7. We have to separate the parts of the phone number logically and
326                 present the phone number as `(XXX) XXX-XXXX`.
327
328                 4.1.7.1. We add `(` to surround the area code part of
329                          `PhoneNumber`.
330
331                 4.1.7.2. We call the first three (3) characters of
332                          `PhoneNumber` using `LEFT()` calling three (3) spaces.
333
334                 4.1.7.3. Then we add `) ` to close the first parenthesis and
335                          complete the way area codes are commonly displayed.
336
337                 4.1.7.4. Now we have to call the inside of `PhoneNumber` using
338                          `SUBSTRING()` calling first where we want to start
339                          (fourth characters) and how many characters from the
340                          first value we want to call -- next three (3)
341                          characters.
342
343                 4.1.7.5. We add a hyphen (`-`) to continue the way how phone
344                          numbers are normally presented.
345
346                 4.1.7.6. We finish by calling the last four (4) characters
347                          using `RIGHT()` and calling four (4) places.
348
349                 4.1.7.7. The latter can be done using a user-defined function.
350
351           4.1.8. Format all monetary values (`c`) with culture `en-us`
352                (https://msdn.microsoft.com/en-us/library/hh213525.aspx).
353
354                 4.1.8.1. Note that formatting any numeric value returns a
355                          string.
356
357                 4.1.8.2. The latter can be done using a user-defined function.
358
359           4.1.9. Format all dates (`d`) with culture `en-us`
360                 (https://msdn.microsoft.com/en-us/library/hh213525.aspx).
361
362                 4.1.9.1. Note that formatting any numeric value returns a
363                          string.
364
```

```
365                 4.1.9.2. The latter can be done using a user-defined function.
366
367           4.1.10. Retrieve values for `AP3.Employees.StartDate` greater or equal
368                   to (`>=`) to `12/30/1999` and less than or equal to (`<=`) to
369                   `6/1/2015` (range of values).
370
371                           WHERE AP3.Employees.StartDate >= `12/30/1999`
372                             AND AP3.Employees.StartDate <= `6/1/2015`
373
374                   The better option is using `BETWEEN`, which is cleaner and
375                   easier to read.
376
377                           WHERE AP3.Employees.StartDate BETWEEN `12/30/1999`
378                             AND `6/1/2015`
379
380                   It also avoids the possibility of errors when using operators
381                   `>=` and `<=`.
382    ********************************************************************** */
383
384  SELECT DISTINCT                                  -- start of query (#4.1)
385    REPLACE(AP3.Employees.EmpID, LEFT(AP3.Employees.EmpID, 7), 'xxx-xx-')
386      AS EmpID,
387    CASE                                           -- 1. concatenating last, first
388      WHEN AP3.Employees.MName IS NOT NULL         --    and middle names; if not
389        OR AP3.Employees.MName <> ''               --    concatenating only first
390        THEN CONCAT (                              --    and last names
391            AP3.Employees.LName,
392            ', ',
393            AP3.Employees.FName,
394            ' ',
395            AP3.Employees.MName
396            )
397      ELSE CONCAT (
398          AP3.Employees.LName,
399          ', ',
400          AP3.Employees.FName
401          )
402      END AS Name,
403    AP3.Employees.Gender,
404    CASE
405      WHEN AP3.Employees.Address2 IS NOT NULL      -- 2. concatenating `Address1`
406        OR AP3.Employees.Address2 <> ''            --    and `Address2` when
407        THEN CONCAT (                              --    `Address2` is NOT NULL or
408            AP3.Employees.Address1,                --    an empty string; if not
409            ' ',                                   --    showing only the original
410            AP3.Employees.Address2                 --    value of `Address1`
411            )
412      ELSE AP3.Employees.Address1
413      END AS Address,
414    AP3.Employees.City,
415    CASE                                           -- 3. concatenating `ZipCode`
416      WHEN AP3.Employees.ZipCode IS NULL           --    and dummy Plus 4 `-0001`
```

```
417       OR AP3.Employees.ZipCode = ''              --    is NOT NULL, an empty
418       OR LEN(AP3.Employees.ZipCode) <> 5         --    string or longer than 5;
419       THEN ''                                    --    otherwise showing only
420     WHEN LEN(AP3.Employees.ZipCode) = 5          --    the original value of
421       THEN CONCAT (                              --    `ZipCode`
422           AP3.Employees.ZipCode,
423           '-0001'
424           )
425     ELSE AP3.Employees.ZipCode
426     END AS ZipCode,
427   CASE                                           -- 4. concatenating an opening
428     WHEN PhoneNumber <> ''                       --    parenthesis (`(`), three
429       OR PhoneNumber <> 10                       --    characters from the left
430       THEN CONCAT (                              --    of `PhoneNumber`, the
431           '(',                                   --    substring starting from
432           LEFT(PhoneNumber, 3),                  --    the 4th character taking
433           ') ',                                  --    3 characters (4th to
434           SUBSTRING(PhoneNumber, 4, 3),          --    6th), a closing
435           '-',                                   --    parenthesis and space
436           RIGHT(PhoneNumber, 4)                  --    (`( `), a hyphen and 4
437           )                                      --    characters from the
438     ELSE PhoneNumber,                            --    right; if not showing
439     END AS PhoneNumber,                          --    only the original value
440                                                  --    of `PhoneNumber`
441   AP3.JobDept.JobDept,
442   AP3.JobTitle.JobTitle,
443   FORMAT(AP3.Employees.JobSalary, 'c', 'en-us') -- 5. formatting dollar amounts
444     AS JobSalary,                                --    as currency (`c`) with
445                                                  --    culture `en-us`
446   FORMAT(AP3.Employees.StartDate, 'd', 'en-us') -- 6. formatting dates as short
447     AS StartDate,                                --    date (`d`) with culture
448                                                  --    `en-us`
449   FORMAT(AP3.Employees.LastUpdateDate,           -- 7. formatting dates as short
450     'd', 'en-us') AS LastUpdateDate,             --    date (`d`) with culture
451                                                  --    `en-us`
452   AP3.JobDept.Comments
453 FROM AP3.Employees
454 LEFT JOIN AP3.JobDept
455   ON AP3.Employees.JobDeptID = AP3.JobDept.JobDeptID
456 LEFT JOIN AP3.JobTitle
457   ON AP3.Employees.JobTitleID = AP3.JobTitle.JobTitleID
458 WHERE AP3.Employees.StartDate BETWEEN '12/30/1999'
459     AND '6/1/2015'
460 ORDER BY Name,
461   EmpID;                                         -- end of query (#4.1)
462
463
464 /* ***************************************************************************
465     4.2. Now we can create view `final.EmployeesVW` in database `labs` using
466          the query above.
467
468          4.2.1. Do not forget to create schema `final` in in database `labs`
```

```
469                 and to add database `WS23SQL1001` before schema `AP3` and the
470                 name of the tables.
471
472           4.4.2. Note that views in T-SQL cannot be created using `ORDER BY` so
473                 we have to remove the clause.
474
475           4.2.3. Therefore we are forced to remove the `ORDER BY` clause from
476                 our query.
477  ********************************************************************** */
478
479 CREATE SCHEMA final;                              -- creating schema `final` in
480                                                   -- database `labs` where we are
481                                                   -- creating view `EmployeesVW`
482
483 CREATE VIEW final.EmployeesVW                     -- 1. start of view
484 AS                                                --    `final.EmployeesVW`
485 (
486     SELECT DISTINCT                               -- 2. start of query (#4.1)
487       REPLACE(WS23SQL1001.AP3.Employees.EmpID,
488         LEFT(WS23SQL1001.AP3.Employees.EmpID, 7), 'xxx-xx-') AS EmpID,
489       CASE
490         WHEN WS23SQL1001.AP3.Employees.MName IS NOT NULL
491           OR WS23SQL1001.AP3.Employees.MName <> ''
492           THEN CONCAT (
493               WS23SQL1001.AP3.Employees.LName,
494               ', ',
495               WS23SQL1001.AP3.Employees.FName,
496               ' ',
497               WS23SQL1001.AP3.Employees.MName
498               )
499         ELSE CONCAT (
500             WS23SQL1001.AP3.Employees.LName,
501             ', ',
502             WS23SQL1001.AP3.Employees.FName
503             )
504         END AS Name,
505       WS23SQL1001.AP3.Employees.Gender,
506       CASE
507         WHEN WS23SQL1001.AP3.Employees.Address2 IS NOT NULL
508           OR WS23SQL1001.AP3.Employees.Address2 <> ''
509           THEN CONCAT (
510               WS23SQL1001.AP3.Employees.Address1,
511               ' ',
512               WS23SQL1001.AP3.Employees.Address2
513               )
514         ELSE WS23SQL1001.AP3.Employees.Address1
515         END AS Address,
516       WS23SQL1001.AP3.Employees.City,
517       CASE
518         WHEN WS23SQL1001.AP3.Employees.ZipCode IS NULL
519           OR WS23SQL1001.AP3.Employees.ZipCode = ''
520           OR LEN(WS23SQL1001.AP3.Employees.ZipCode) <> 5
```

```
521           THEN ''
522         WHEN LEN(WS23SQL1001.AP3.Employees.ZipCode) = 5
523           THEN CONCAT (
524               WS23SQL1001.AP3.Employees.ZipCode,
525               '-0001'
526               )
527         ELSE WS23SQL1001.AP3.Employees.ZipCode
528         END AS ZipCode,
529       CASE
530         WHEN PhoneNumber IS NOT NULL            -- 1. checking for NULL
531           OR PhoneNumber <> ''                  -- 2. checking that it is not
532                                                 --    an empty string
533           OR PhoneNumber <> 10                  -- 3. checking for length
534           OR PhoneNumber NOT LIKE ('(%)%-%')    -- 4. checking for format
535           THEN CONCAT (
536               '(',
537               LEFT(PhoneNumber, 3),
538               ') ',
539               SUBSTRING(PhoneNumber, 4, 3),
540               '-',
541               RIGHT(PhoneNumber, 4)
542               )
543         ELSE PhoneNumber
544         END AS PhoneNumber,
545       WS23SQL1001.AP3.JobDept.JobDept,
546       WS23SQL1001.AP3.JobTitle.JobTitle,
547       FORMAT(WS23SQL1001.AP3.Employees.JobSalary,
548         'c', 'en-us') AS JobSalary,
549       FORMAT(WS23SQL1001.AP3.Employees.StartDate,
550         'd', 'en-us') AS StartDate,
551       FORMAT(WS23SQL1001.AP3.Employees.LastUpdateDate,
552         'd', 'en-us') AS LastUpdateDate,
553       WS23SQL1001.AP3.JobDept.Comments
554     FROM WS23SQL1001.AP3.Employees
555     LEFT JOIN WS23SQL1001.AP3.JobDept
556       ON WS23SQL1001.AP3.Employees.JobDeptID =
557         WS23SQL1001.AP3.JobDept.JobDeptID
558     LEFT JOIN WS23SQL1001.AP3.JobTitle
559       ON WS23SQL1001.AP3.Employees.JobTitleID =
560         WS23SQL1001.AP3.JobTitle.JobTitleID
561     WHERE WS23SQL1001.AP3.Employees.StartDate BETWEEN '12/30/1999'
562         AND '6/1/2015'
563     -- ORDER BY Name,                           -- 3. no `ORDER BY` in views
564       -- EmpID                                 --    3.1. end of query (#4.1)
565     );                                          --    3.2. end of view (#4.2)
566
567
568 /* ***************************************************************************
569          As an alternative, we could use aliases (`AS`) for the tables in the
570          previous query inside the view.
571
572                     `emp` for `WS23SQL1001.AP3.Employees`
```

```
573                     `jd`  for `WS23SQL1001.AP3.JobDept`
574                     `jt`  for `WS23SQL1001.AP3.JobTitle`
575  ********************************************************************** */
576
577 CREATE VIEW final.EmployeesVW
578 AS
579 (
580     SELECT DISTINCT REPLACE(emp.EmpID, LEFT(emp.EmpID, 7), 'xxx-xx-') AS EmpID,
581       CASE
582         WHEN emp.MName IS NOT NULL
583           OR emp.MName <> ''
584           THEN CONCAT (
585               emp.LName,
586               ', ',
587               emp.FName,
588               ' ',
589               emp.MName
590               )
591         ELSE CONCAT (
592             emp.LName,
593             ', ',
594             emp.FName
595             )
596         END AS Name,
597       emp.Gender,
598       CASE
599         WHEN emp.Address2 IS NOT NULL
600           OR emp.Address2 <> ''
601           THEN CONCAT (
602               emp.Address1,
603               ' ',
604               emp.Address2
605               )
606         ELSE emp.Address1
607         END AS Address,
608       emp.City,
609       CASE
610         WHEN emp.ZipCode IS NULL
611           OR emp.ZipCode = ''
612           OR LEN(emp.ZipCode) <> 5
613           THEN ''
614         WHEN LEN(emp.ZipCode) = 5
615           THEN CONCAT (
616               emp.ZipCode,
617               '-0001'
618               )
619         ELSE emp.ZipCode
620         END AS ZipCode,
621       CASE
622         WHEN PhoneNumber IS NOT NULL
623           OR PhoneNumber <> ''
624           OR PhoneNumber <> 10
```

```
625             OR PhoneNumber NOT LIKE ('(%)%-%')
626             THEN CONCAT (
627                 '(',
628                 LEFT(PhoneNumber, 3),
629                 ') ',
630                 SUBSTRING(PhoneNumber, 4, 3),
631                 '-',
632                 RIGHT(PhoneNumber, 4)
633                 )
634           ELSE PhoneNumber
635           END AS PhoneNumber,
636         jd.JobDept,
637         jt.JobTitle,
638         FORMAT(emp.JobSalary, 'c', 'en-us') AS JobSalary,
639         FORMAT(emp.StartDate, 'd', 'en-us') AS StartDate,
640         FORMAT(emp.LastUpdateDate, 'd', 'en-us') AS LastUpdateDate,
641         jd.Comments
642       FROM WS23SQL1001.AP3.Employees AS emp
643       LEFT JOIN WS23SQL1001.AP3.JobDept AS jd
644         ON emp.JobDeptID = jd.JobDeptID
645       LEFT JOIN WS23SQL1001.AP3.JobTitle AS jt
646         ON emp.JobTitleID = jt.JobTitleID
647       WHERE emp.StartDate BETWEEN '12/30/1999'
648           AND '6/1/2015'
649       );
650
651
652   /* ***************************************************************************
653       4.3. To make sure that the view displays the correct data, we can call all
654            the values referred in view `final.EmployeesVW`.
655    *************************************************************************** */
656
657   SELECT *
658   FROM final.EmployeesVW;
659
660
661   /* ***************************************************************************
662    5. As a bonus, you can use functions to format, concatenate and other tasks
663       when writing the query that will be part of your view.
664
665       5.1. First we write the function to add `XXX-XX-` to the last four
666            characters in `EmpID` (#4.1.3).
667    *************************************************************************** */
668
669   CREATE FUNCTION final.EmpID_udf (@in_empid VARCHAR(15))
670   RETURNS VARCHAR(15)                              -- function returns...
671   AS
672     BEGIN
673       -- declaring output and passing value of `REPLACE` in-line
674       DECLARE @out_empid VARCHAR(15) = REPLACE(@in_empid,
675                                           LEFT(@in_empid, 7),
676                                           'xxx-xx-')
```

```
677     RETURN @out_empid
678   END;
679
680
681 /* ************************************************************************
682     5.2. Then we write the function to concatenate `LName`, `FName` and `MName`
683          with spaces in between if we have a value for `MName` or `LName` and
684          `FName` with a space in between if we have no value for `MName`
685          (#4.1.4).
686  ************************************************************************ */
687
688 CREATE FUNCTION final.fullname_udf (
689   @in_fname VARCHAR(50),
690   @in_mname VARCHAR(1),
691   @in_lname VARCHAR(50)
692   )                                          -- accepting three (3) values
693 RETURNS VARCHAR(101)                         -- function returns...
694 AS
695   BEGIN
696     DECLARE @out_fullname VARCHAR(101) = CASE
697         WHEN @in_mname IS NOT NULL
698           OR @in_mname <> ''
699           THEN CONCAT (
700               @in_lname,
701               ', ',
702               @in_fname,
703               ' ',
704               @in_mname
705               )
706         ELSE CONCAT (
707             @in_lname,
708             ', ',
709             @in_fname
710             )
711         END
712     RETURN @out_fullname
713   END;
714
715
716 /* ************************************************************************
717     5.3. Then we write the function to Concatenate with space between
718          `Address1` and `Address2` if we have a value in `Address2` or only
719          call `Address1` if we have no value in `Address2` (#4.1.5).
720  ************************************************************************ */
721
722 CREATE FUNCTION final.address2_udf (
723   @in_address1 VARCHAR(100),
724   @in_address2 VARCHAR(100)
725   )                                          -- accepting two (2) values
726 RETURNS VARCHAR(201)                         -- function returns...
727 AS
728   BEGIN
```

```
729     DECLARE @out_address VARCHAR(201) = CASE
730         WHEN @in_address2 IS NOT NULL
731           OR @in_address2 <> ''
732           THEN CONCAT (
733               @in_address1,
734               ' ',
735               @in_address2
736               )
737         ELSE @in_address1
738         END                                     -- closing `CASE`
739     RETURN @out_address
740   END;                                          -- closing `BEGIN` (function)
741
742
743 /* ************************************************************************
744     5.4. Then we write the function to add dummy Plus 4 `-0001` (non-existent
745          string in the database, hard-coded) before field `ZipCode` (#4.1.6).
746  ************************************************************************ */
747
748 CREATE FUNCTION final.zipcode_udf (@in_zipcode VARCHAR(10))
749 RETURNS VARCHAR(10)                             -- function returns...
750 AS
751   BEGIN
752     DECLARE @out_zipcode VARCHAR(10) = CASE
753         WHEN @in_zipcode IS NULL
754           OR @in_zipcode = ''
755           OR LEN(@in_zipcode) <> 5
756           THEN ''
757         WHEN LEN(@in_zipcode) = 5
758           THEN CONCAT (
759               @in_zipcode,
760               '-0001'
761               )
762         ELSE @in_zipcode
763         END                                     -- closing `CASE`
764     RETURN @out_zipcode
765   END;                                          -- closing `BEGIN` (function)
766
767
768 /* ************************************************************************
769     5.4. Then we write the function to to separate the parts of the phone
770          number logically and present the phone number as `(XXX) XXX-XXXX`
771          (#4.1.7).
772  ************************************************************************ */
773
774 CREATE FUNCTION final.phones_udf (@in_phone VARCHAR(15))
775 RETURNS VARCHAR(15)                             -- function returns...
776 AS
777   BEGIN
778     DECLARE @out_phone VARCHAR(15) = CASE
779         WHEN @in_phone IS NOT NULL
780           OR @in_phone <> ''
```

```
781             OR @in_phone <> 10
782             OR @in_phone NOT LIKE ('(%)%-%')
783             THEN CONCAT (
784                 '(',
785                 LEFT(@in_phone, 3),
786                 ') ',
787                 SUBSTRING(@in_phone, 4, 3),
788                 '-',
789                 RIGHT(@in_phone, 4)
790                 )
791           ELSE @in_phone
792           END                                       -- closes `CASE`
793       RETURN @out_phone
794     END;                                            -- closes function
795
796
797   /* ***************************************************************************
798       5.5. Now we can re-write the view.
799    *************************************************************************** */
800
801   CREATE VIEW final.EmployeesVW
802   AS
803   (
804       SELECT DISTINCT
805         final.EmpID_udf(WS23SQL1001.AP3.Employees.EmpID) AS EmpID,
806         final.fullname_udf(WS23SQL1001.AP3.Employees.LName,
807                 WS23SQL1001.AP3.Employees.FName,
808                 WS23SQL1001.AP3.Employees.MName
809                 ) AS Name,
810         WS23SQL1001.AP3.Employees.Gender,
811         final.address2_udf(
812                 WS23SQL1001.AP3.Employees.Address1,
813                 WS23SQL1001.AP3.Employees.Address2
814                 ) AS Address,
815         WS23SQL1001.AP3.Employees.City,
816         final.zipcode_udf(WS23SQL1001.AP3.Employees.ZipCode) AS ZipCode,
817         final.phones_udf(PhoneNumber) AS PhoneNumber,
818         WS23SQL1001.AP3.JobDept.JobDept,
819         WS23SQL1001.AP3.JobTitle.JobTitle,
820         FORMAT(WS23SQL1001.AP3.Employees.JobSalary,
821           'c', 'en-us') AS JobSalary,
822         FORMAT(WS23SQL1001.AP3.Employees.StartDate,
823           'd', 'en-us') AS StartDate,
824         FORMAT(WS23SQL1001.AP3.Employees.LastUpdateDate,
825           'd', 'en-us') AS LastUpdateDate,
826         WS23SQL1001.AP3.JobDept.Comments
827       FROM WS23SQL1001.AP3.Employees
828       LEFT JOIN WS23SQL1001.AP3.JobDept
829         ON WS23SQL1001.AP3.Employees.JobDeptID =
830           WS23SQL1001.AP3.JobDept.JobDeptID
831       LEFT JOIN WS23SQL1001.AP3.JobTitle
832         ON WS23SQL1001.AP3.Employees.JobTitleID =
```

```
        WS23SQL1001.AP3.JobTitle.JobTitleID
    WHERE WS23SQL1001.AP3.Employees.StartDate BETWEEN '12/30/1999'
        AND '6/1/2015'
    );


/* ************************************************************************
    5.6. As a bonus, there are other objects that can be added to tables.
         Constraints like keys restrict the possibility of losing data by
         dropping the wrong object or even inserting the wrong data into a
         field.

           ``A primary key, also called a primary keyword, is a key in a
           relational database that is unique for each record.  It is a unique
           identifier, such as a driver license number, telephone number
           (including area code), or vehicle identification number (VIN).  A
           relational database must always have one and only one primary key.
           Primary keys typically appear as columns in relational database
           tables.
           The choice of a primary key in a relational database often depends
           on the preference of the administrator.  It is possible to change
           the primary key for a given database when the specific needs of the
           users changes.  For example, the people in a town might be uniquely
           identified according to their driver license numbers in one
           application, but in another situation it might be more convenient to
           identify them according to their telephone numbers.``
           https://searchsqlserver.techtarget.com/definition/primary-key

         The syntax is similar to that of any structure change of a table.  In
         this case, we add a primary key (PK) constraint.

                    ALTER TABLE table_name
                    ADD CONSTRAINT pk_name
                      PRIMARY KEY (column_name)

           ``A foreign key is a column or columns of data in one table that
           connects to the primary key data in the original table.
           To ensure the links between foreign key and primary key tables
           aren't broken, foreign key constraints can be created to prevent
           actions that would damage the links between tables and prevent
           erroneous data from being added to the foreign key column.``
           https://searchoracle.techtarget.com/definition/foreign-key

         Just like in the case of primary keys, we need to alter the table to
         add a foreign key (FK) constraint.  The big difference is that we need
         to indicate the primary key (PK), on which the FK depends on.

                    ALTER TABLE child_table
                    ADD CONSTRAINT fk_name
                      FOREIGN KEY child_table_column
                      REFERENCES parent_table (parent_column)

```

```
885           These two constraints are different as noted above.  The main
886           difference is that a table can have only one primary key (PK) making
887           it the identifier for the row and/or multiple foreign keys (FK's')
888           referrencing other PK's in other tables in order to maintain
889           dependency to other tables.
890
891             ``Differences between primary and foreign keys
892             A primary key in the original table, or parent table, can be
893             targeted by multiple foreign keys from other `child` tables.  But a
894             primary key does not necessarily have to be the target of any
895             foreign keys.  A primary key is a column or a set of columns that
896             identify a row in a table.  A foreign key, however, is in a table
897             that is different from the table that the primary key must match.``
898             https://searchoracle.techtarget.com/definition/foreign-key
899
900           In the example below, we make a PK from `AP1.Vendors.VendorID` and a
901           FK from `AP1.ContactUpdates.ContactUpdateID`.
902    ********************************************************************** */
903
904   ALTER TABLE AP1.Vendors                          -- must make `VendorID` NOT
905     ALTER COLUMN VendorID INT NOT NULL;            -- NULL before making it a key
906
907   ALTER TABLE AP1.Vendors                          -- altering table by adding
908     ADD CONSTRAINT VendorID_PK                     -- constraint PRIMARY KEY with
909     PRIMARY KEY (VendorID);                        -- `VendorID` in parenthesis
910
911
912   ALTER TABLE AP1.ContactUpdates                   -- must make `ContactUpdateID`
913     ALTER COLUMN ContactUpdateID INT NOT NULL;     -- NOT NULL before making it a
914                                                    -- key
915
916   ALTER TABLE AP1.ContactUpdates                   -- altering table by adding
917     ADD CONSTRAINT ContactUpdateID_FK              -- constraint FOREIGN KEY with
918     FOREIGN KEY (VendorID)                         -- `VendorID` in parenthesis
919     REFERENCES AP1.Vendors (VendorID);             -- indicating the reference to
920                                                    -- the PK in parent table
921
922
923   /* *********************************************************************
924    7. What do you do now that the `Introduction to SQL` course has ended?
925
926       7.1. Go to Microsoft Virtual Academy (https://mva.microsoft.com/) and
927            continue learning including application to write reports
928            (https://docs.microsoft.com/en-us/sql/reporting-services/), visualize
929            and/or analyze data (https://powerbi.microsoft.com/).
930
931       7.2. Register for the `Intermediate to SQL` course
932            (https://www.campusce.net/bmcc/course/course.aspx?catId=33).
933
934       7.3. If you are interested in a career in data science, learn Python
935            (https://python.org/) and the multiple libraries available for data
936            analysis (https://folvera.commons.gc.cuny.edu/?s=python).
```

```
937
938      7.4. Contact me if you ever need help.
939
940   8. Before you leave, figure out what the following prints.
941
942      8.1. Change the value `your_name` with your name before running the code
943           below to create the procedure.  Then execute it.
944
945      8.2. If curious, visit http://ascii.cl/ for more information on ASCII
946           (http://whatis.techtarget.com/definition/ASCII-American-Standard-Code-
             for-Information-Interchange).
947   ********************************************************************** */
948
949   CREATE PROCEDURE final.message_sp
950   AS
951   BEGIN
952     DECLARE @yourName VARCHAR(50) = 'your_name',     -- param to hold your name
953             @CourseCd VARCHAR(15) = 'WS23SQL1001';   -- code for the SQL course
954     PRINT CONCAT ( CHAR(084), CHAR(104), CHAR(097), CHAR(110), CHAR(107),
955         CHAR(032), CHAR(121), CHAR(111), CHAR(117), CHAR(044), CHAR(032),
956         @yourName, CHAR(044), CHAR(032), CHAR(102), CHAR(111), CHAR(114),
957         CHAR(032), CHAR(116), CHAR(097), CHAR(107), CHAR(105), CHAR(110),
958         CHAR(103), CHAR(032), CHAR(099), CHAR(108), CHAR(097), CHAR(115),
959         CHAR(115), CHAR(032), @CourseCd, CHAR(046), CHAR(013), CHAR(083),
960         CHAR(101), CHAR(101), CHAR(032), CHAR(121), CHAR(111), CHAR(117),
961         CHAR(032), CHAR(105), CHAR(110), CHAR(032), CHAR(116), CHAR(104),
962         CHAR(101), CHAR(032), CHAR(105), CHAR(110), CHAR(116), CHAR(101),
963         CHAR(114), CHAR(109), CHAR(101), CHAR(100), CHAR(105), CHAR(097),
964         CHAR(116), CHAR(101), CHAR(032), CHAR(099), CHAR(108), CHAR(097),
965         CHAR(115), CHAR(115), CHAR(046), CHAR(013), CHAR(013), CHAR(070),
966         CHAR(046), CHAR(079), CHAR(108), CHAR(118), CHAR(101), CHAR(114),
967         CHAR(097), CHAR(032), CHAR(040), CHAR(102), CHAR(111), CHAR(108),
968         CHAR(118), CHAR(101), CHAR(114), CHAR(097), CHAR(064), CHAR(098),
969         CHAR(109), CHAR(099), CHAR(099), CHAR(046), CHAR(099), CHAR(117),
970         CHAR(110), CHAR(121), CHAR(046), CHAR(101), CHAR(100), CHAR(117),
971         CHAR(041), CHAR(013), CHAR(104), CHAR(116), CHAR(116), CHAR(112),
972         CHAR(058), CHAR(047), CHAR(047), CHAR(102), CHAR(111), CHAR(108),
973         CHAR(118), CHAR(101), CHAR(114), CHAR(097), CHAR(046), CHAR(099),
974         CHAR(111), CHAR(109), CHAR(109), CHAR(111), CHAR(110), CHAR(115),
975         CHAR(046), CHAR(103), CHAR(099), CHAR(046), CHAR(099), CHAR(117),
976         CHAR(110), CHAR(121), CHAR(046), CHAR(101), CHAR(100), CHAR(117),
977         CHAR(047) );                                 -- all characters in ASCII
978   END;
979
980   EXEC final.message_sp;
981
982   /* **********************************************************************
983    https://folvera.commons.gc.cuny.edu/?p=1245
984    ********************************************************************** */
```