```
/* ************************************************************************
          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 #9 (2023/05/01):  CREATING DATABASE OBJECTS

  1. Parameters, user-defined functions and stored procedures
 ************************************************************************

 1. LAB #10
    1.01. In schema `lab10` in database `labs`, create table `students`
          (referenced as `labs.lab10.students`) with the following structure.

                    student_id INT NULL
                    student_fname VARCHAR(50) NULL
                    student_lname VARCHAR(50) NULL
                    student_phone VARCHAR(15) NULL
                    student_dob DATE NULL
                    record_date DATE NULL

 ************************************************************************ */

CREATE SCHEMA lab10;

CREATE TABLE lab10.students (                   -- 1. rule of thumb: table
                                                --    names in plural
  student_id INT NULL,                          -- 2. declared as INT;  can
                                                --    accept NULL (can have no
                                                --    value)
  student_fname VARCHAR(50) NULL,               -- 3. declared as VARCHAR(50);
                                                --    can accept NULL (can have
                                                --    no value)
  student_lname VARCHAR(50) NULL,               -- 4. declared as VARCHAR(50);
                                                --    can accept NULL (can have
                                                --    no value)
  student_phone VARCHAR(15) NULL,               -- 5. declared as VARCHAR(50);
                                                --    can accept NULL (can have
                                                --    no value)
  student_dob DATE NULL,                        -- 6. declared as DATE
                                                --
                                                --    DATETIME  9/20/2021 21:54
                                                --    DATE      9/20/2021
                                                --    TIME                21:54
                                                --
                                                --    can accept NULL (can have
                                                --    no value)
  record_date DATE NULL                         -- 5. declared as DATE;  when
                                                --    record was created;  can
                                                --    accept NULL (can have no
                                                --    value)
```

```
  );


/* ************************************************************************
    1.02. Populate the table with some data of your choice.

          If we do not have a value for a specific field, we can push an empty
          string or NULL.
 ************************************************************************ */

INSERT INTO lab10.students
VALUES (
  1,
  'Joe',
  'Smith',
  '555-123-4567',
  '1980/05/01',
  GETDATE()                                    -- 1. built-in function to
                                               --    retrieve system DATETIME
  ),
  (
  2,
  'Mary',
  'Jones',
  '212-555-1000',
  '1983/05/16',
  GETDATE()
  ),
  (
  3,
  'Peter',
  'Johnson',
  NULL,                                        -- 2. inserting empty strings
                                               --    (``) or NULL since we
                                               --    have no values for fields
                                               --    to insert same number of
                                               --    values as columns
  '06/01/1980',
  GETDATE()
  );


/* ************************************************************************
    1.03. Insert values.

          We call the the three (3) corresponding columns to indicate which
          value goes where.

          We do not need to call columns in order as long order as long as
          values are pushed in the same order (value 1 in field 1, value 2 in
          field 2, value 3 in field 3 and value 7 in field 7).
 ************************************************************************ */
```

```
105
106 INSERT INTO lab10.students (
107   student_id,                                -- 1. inserting values to only
108   student_fname,                             --    four (4) columns;
109   student_lname,                             --    indicating which four (4)
110   record_date                                --    columns
111   )
112 VALUES (
113   4,                                         -- 2. values to be inserted in
114   'Smith',                                   --    columns `student_id`,
115   'Tom',                                     --    `student_fname`,
116   GETDATE()                                  --    `student_lname` and
117   );                                         --    `record_date` receiving
118                                              --    value from `GETDATE()`
119
120
121 /* ************************************************************************
122          In the example below, we insert row 6 before 5.
123
124          The values in `student_id` (the row identifier) are unique, but they
125          do not need to be in order.
126
127          If we need to insert values in `student_id` automatically in
128          incremental order, we would need to use `IDENTITY(1,1)` as part of
129          the table structure.  The first integer indicates that the first
130          value as 1.  The second integer indicates that the value is
131          incremented by 1.  Refer to
132          https://www.w3schools.com/sql/sql_autoincrement.asp for more
133          information.
134
135                    CREATE TABLE lab10.students (
136                      student_id INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
137                      student_fname VARCHAR(50) NULL,
138                      student_lname VARCHAR(50) NULL,
139                      student_phone VARCHAR(15) NULL,
140                      student_dob DATE NULL,
141                      record_date DATE NULL
142                      );
143  ************************************************************************ */
144
145 INSERT INTO lab10.students
146 VALUES (
147   6,
148   'John',
149   'Scott',
150   '',                                        -- 1. inserting empty strings
151   '',                                        --    (``) or NULL since we
152                                              --    have no values for fields
153                                              --    to insert same number of
154                                              --    values as columns
155   GETDATE()                                  -- 2. built-in function to
156                                              --    retrieve system DATETIME
```

```
157    ),
158    (
159    5,
160    'Mary Ann',
161    'Saunders',
162    '',                                          -- 3. inserting empty strings
163    '',                                          --    (``) or NULL since we
164                                                --    have no values for fields
165                                                --    to insert same number of
166                                                --    values as columns
167    GETDATE()                                    -- 4. built-in function to
168                                                --    retrieve system DATETIME
169    );
170
171
172  /* ************************************************************************
173            We can also delete/destroy data objects.
174
175            For the time being, we will work with tables
176            (https://techonthenet.com/sql_server/tables/drop_table.php).
177
178            Once an object is deleted, there is no way to rescue the data
179            (`ROLLBACK`) unless first creating a `SAVEPOINT`
180            (https://technet.microsoft.com/en-us/library/ms178157.aspx).
181
182            In the example below, we destroy (`DROP`) table `lab10.students`
183            understanding that, once we do, we cannot recover the structure or
184            the data.
185   ************************************************************************ */
186
187  DROP TABLE lab10.students;
188
189
190  /* ************************************************************************
191            In the case of tables, we can destroy (`TRUNCATE`) the data in the
192            table without affecting the structure of the table understanding
193            that, once we do, we cannot recover the data.
194   ************************************************************************ */
195
196  TRUNCATE TABLE lab10.students;
197
198
199  /* ************************************************************************
200            We can also modify (`ALTER`) data objects
201            (https://techonthenet.com/sql_server/tables/alter_table.php).
202
203            ADD       to add a column to a table
204
205            DROP      to delete a column to a table
206
207            ALTER     to change the data type or size of a column
208
```

```
209
210     1.04. Change the structure of table `lab10.students`.
211
212           * Add `email` as VARCHAR(100).
213           * Drop `Email` because we want a different name.  Remember that there
214             is no means to rename any object.
215           * Add `student_email` as VARCHAR(100).
216           * Alter the size of `student_email` to 50.
217           * Alter `student_id` to `NOT NULL`.
218           * Alter `record_date`, `student_fname` and `student_lname` to
219             `NOT NULL`.
220           * Alter `student_id` to `VARCHAR(5)`.
221           * Try to change `student_fname` to FLOAT.  Note you will get an error
222             since a string cannot be made into a number.
223  ********************************************************************** */
224
225 ALTER TABLE lab10.students                      -- 01. adding new column
226 ADD email VARCHAR(100);                         --     `Email`;  no need to
227                                                 --     specify that we are
228                                                 --     adding a column
229
230 ALTER TABLE lab10.students                      -- 02. dropping (deleting)
231 DROP COLUMN email;                              --     column `Email` as there
232                                                 --     is no SQL statement to
233                                                 --     rename data objects;
234                                                 --     must specify that we are
235                                                 --     dropping a column
236
237 ALTER TABLE lab10.students                      -- 03. adding new (replacement)
238 ADD student_email VARCHAR(100);                 --     column `student_email`;
239                                                 --     no need to specify that
240                                                 --     we are adding a column
241
242 ALTER TABLE lab10.students                      -- 04. altering column with new
243 ALTER COLUMN student_email VARCHAR(50) NULL;    --     data type VARCHAR(50)
244                                                 --     from VARCHAR(100) and
245                                                 --     `NOT NULL`;  must
246                                                 --     specify that we are
247                                                 --     altering a column
248
249 ALTER TABLE lab10.students                      -- 05. altering column as
250 ALTER COLUMN student_id INT NOT NULL;           --     `NOT NULL`;  must
251                                                 --     specify that we are
252                                                 --     altering a column
253
254 ALTER TABLE lab10.students                      -- 06. altering column with new
255 ALTER COLUMN record_date DATETIME NOT NULL;     --     data type DATETIME from
256                                                 --     DATE and `NOT NULL`;
257                                                 --     must specify that we are
258                                                 --     altering a column
259
260 ALTER TABLE lab10.students                      -- 07. altering column with new
```

```
261 ALTER COLUMN student_fname VARCHAR(25) NOT NULL;--     data type VARCHAR(25)
262                                                 --     from VARCHAR(50) and
263                                                 --     `NOT NULL`;  must
264                                                 --     specify that we are
265                                                 --     altering a column
266
267 ALTER TABLE lab10.students                       -- 08. altering column with new
268 ALTER COLUMN student_lname VARCHAR(25) NOT NULL;--     data type VARCHAR(25)
269                                                 --     from VARCHAR(50) and
270                                                 --     `NOT NULL`;  must
271                                                 --     specify that we are
272                                                 --     altering a column
273
274 ALTER TABLE lab10.students                       -- 09. altering column with new
275 ALTER COLUMN student_id VARCHAR(5);              --     data type VARCHAR(5)
276                                                 --     from INT;  no error
277                                                 --     during conversion;  must
278                                                 --     specify that we are
279                                                 --     altering a column
280
281 ALTER TABLE lab10.students                       -- 10. altering column back to
282 ALTER COLUMN student_id INT NOT NULL;            --     data type INT from
283                                                 --     VARCHAR(5);  no error
284                                                 --     during conversion;  must
285                                                 --     specify that we are
286                                                 --     altering a column
287
288 ALTER TABLE lab10.students                       -- 11. trying to alter column
289 ALTER COLUMN student_fname FLOAT;                --     to data type FLOAT from
290                                                 --     VARCHAR(25);  conversion
291                                                 --     failure due to format
292                                                 --     incompatibility (letters
293                                                 --     to numbers)
294
295
296 /* ********************************************************************************
297     1.05. Update the value of column `student_phone` passing value `No Number`
298           where there is no value (`IS NULL`) or there is an empty space (` `)
299           or empty string (``).
300  ******************************************************************************** */
301
302 UPDATE lab10.students
303 SET student_phone = 'No Number'
304 WHERE student_phone IS NULL                      -- 1. checking for NULL
305   OR student_phone = ' '                         -- 2. checking for a space
306   OR student_phone = '';                         -- 3. checking for an empty
307                                                 --    string
308
309
310 /* ********************************************************************************
311     1.06. Update the value of column `student_email` passing the value of the
312           concatenation of `student_fname` and `student_lname` with a period
```

```
313           (`.`) between the two columns -- for example, `john.smith@foobar.foo`
314           for `student_fname` with value of `John` and `student_lname` with
315           value of `Smith`.
316   ********************************************************************** */
317
318  UPDATE lab10.students
319  SET student_email = LOWER(CONCAT (
320        student_fname,
321        '.',
322        student_lname,
323        '@foobar.foo'
324        ));
325
326
327  /* ***************************************************************************
328      1.07. In the example below, we UPDATE column `record_date` where the field
329           is NULL or has an empty space (``) with value from `GETDATE()`.
330   ********************************************************************** */
331
332  UPDATE lab10.students
333  SET record_date = GETDATE()
334  WHERE record_date IS NULL
335    OR record_date = '';
336
337
338  /* ***************************************************************************
339           In the example below, we can UPDATE `student_dob` to `1980/01/23`
340           where `student_id` is `1`.
341   ********************************************************************** */
342
343  UPDATE lab10.students
344  SET student_dob = '1980/01/23'
345  WHERE student_id = 1;
346
347
348  /* ***************************************************************************
349   2. LAB #11 (CREATING OBJECTS)
350      2.01. In schema `lab11` in database `labs`, create table `grades`
351           (referenced as `labs.lab11.grades`) with the following structure.
352
353                     grade_id      INT           NOT NULL    UNIQUE
354                     student_id    INT           NOT NULL
355                     student_grade FLOAT         NOT NULL
356                     grade_comment VARCHAR(255)  NULL
357   ********************************************************************** */
358
359  CREATE SCHEMA lab11;
360
361  CREATE TABLE lab11.grades (
362    grade_id INT NOT NULL UNIQUE,
363    student_id INT NOT NULL,
364    student_grade FLOAT NOT NULL,
```

```
365    grade_comment VARCHAR(255) NULL
366    );
367
368
369  /* ********************************************************************
370      2.02. Then populate the table with some data of your choice.
371   ******************************************************************** */
372
373  INSERT INTO lab11.grades
374  VALUES (
375    1,
376    1,
377    80,
378    'He missed the midterm.'
379    ),
380    (
381    2,
382    3,
383    65,
384    'He slept in class.'
385    ),
386    (
387    3,
388    2,
389    98,
390    ''
391    );
392
393
394  /* ********************************************************************
395      2.03. Since we have shared (`student_id`) data between `labs.lab11.grades`
396            and `labs.lab10.students` from the previous lab, we can retrieve all
397            the data from `labs.lab10.students` (main) and any related data from
398            `labs.lab11.grades` (secondary) without duplicate rows (`SELECT
399            DISTINCT`).
400   ******************************************************************** */
401
402  SELECT DISTINCT lab10.students.student_id,
403    lab10.students.student_fname,
404    lab10.students.student_lname,
405    lab10.students.student_phone,
406    lab10.students.student_dob,
407    lab10.students.record_date,
408    lab11.grades.grade_id,
409    -- lab11.grades.student_id AS Expr1,
410    lab11.grades.student_grade,
411    lab11.grades.grade_comment
412  FROM lab10.students
413  LEFT OUTER JOIN lab11.grades
414    ON lab10.students.student_id = lab11.grades.student_id
415  ORDER BY student_lname;
416
```

```
417
418 /* ************************************************************************
419           As an alternative, we can use an alias for the table to avoid
420           repeating the long name of `SF21SQL1001.AP1.Vendors` throughout the
421           query.
422
423                     `s` for `lab10.students`
424                     `s` for `lab11.grades`
425  ************************************************************************ */
426
427  SELECT DISTINCT s.student_id,
428   s.student_fname,
429   s.student_lname,
430   s.student_phone,
431   s.student_dob,
432   s.record_date,
433   g.grade_id,
434   -- g.student_id AS Expr1,
435   g.student_grade,
436   g.grade_comment
437 FROM lab10.students AS s
438 LEFT OUTER JOIN lab11.grades AS g
439   ON s.student_id = g.student_id
440 ORDER BY student_lname;
441
442
443 /* ************************************************************************
444     2.04. Since we can query `labs.lab10.students` (main table) and
445           `labs.lab11.grades` (secondary table), we can also CREATE VIEW
446           `labs.lab11.students_grades_vw` from it.
447
448           Since a VIEW calls a `SELECT` statement and is of the same hierarchy
449           as a TABLE, we can query the VIEW as if it were a TABLE.
450
451                     CREATE VIEW schema.view_name
452                     AS
453                       (
454                         SELECT ...
455                       )
456  ************************************************************************ */
457
458 CREATE VIEW lab11.students_grades_vw
459 AS
460 SELECT DISTINCT lab10.students.student_id,
461   lab10.students.student_fname,
462   lab10.students.student_lname,
463   lab10.students.student_phone,
464   lab10.students.student_dob,
465   lab10.students.record_date,
466   lab11.grades.grade_id,
467   -- lab11.grades.student_id AS Expr1,
468   lab11.grades.student_grade,
```

```
469     lab11.grades.grade_comment
470   FROM lab10.students
471   LEFT JOIN lab11.grades
472     ON lab10.students.student_id = lab11.grades.student_id
473   -- ORDER BY student_lname
474
475
476   /* ************************************************************************
477             As an alternative, we can use an alias for the table to avoid
478             repeating the long name of `SF21SQL1001.AP1.Vendors` throughout the
479             query.
480
481                       `s` for `lab10.students`
482                       `s` for `lab11.grades`
483    ************************************************************************ */
484
485   ALTER VIEW lab11.students_grades_vw
486   AS
487   SELECT DISTINCT s.student_id,
488     s.student_fname,
489     s.student_lname,
490     s.student_phone,
491     s.student_dob,
492     s.record_date,
493     g.grade_id,
494     -- g.student_id AS Expr1,
495     g.student_grade,
496     g.grade_comment
497   FROM lab10.students AS s
498   LEFT OUTER JOIN lab11.grades AS g
499     ON s.student_id = g.student_id
500   -- ORDER BY student_lname;
501
502
503   /* ************************************************************************
504             Although we can UPDATE a record when we change any existing value,
505             there are situations where we need to keep track every transaction
506             historically -- for example, to keep track of bank transactions.  In
507             such scenario, we should INSERT a new record for each transaction
508             with a separate column to record the time stamp.
509
510             First we would need to add a column for the time stamp.
511
512             Then we would push the value of `GETDATE()` into the new column.  Of
513             course, for this to work all records should have a value in new
514             column.
515
516             To retrieve the latest record for student, we would need to call the
517             `MAX()` value of all fields in the query and group the results by an
518             identifier -- for example, `student_id` in the example below.
519    ************************************************************************ */
520
```

```
521 ALTER TABLE lab11.grades                          -- adding `grade_timestamp` to
522 ADD grade_timestamp DATETIME;                    -- table `lab11.grades`
523
524 UPDATE lab11.grades                              -- inserting values into
525 SET grade_timestamp = GETDATE();                 -- `grade_timestamp`
526
527 INSERT INTO lab11.grades                         -- inserting two new records at
528 VALUES (                                         -- the same time hence writing
529   1,                                             -- the same value of
530   1,                                             -- `GETDATE()` to both records
531   90,
532   'teacher''s pet'
533   ),
534   (
535   5,
536   2,
537   85,
538   '',
539   GETDATE()
540   );
541
542 INSERT INTO lab11.grades                         -- inserting a new record for
543 VALUES (                                         -- for `student_id` 1
544   1,
545   8,
546   95,
547   'grade change',
548   GETDATE()
549   );
550
551 SELECT DISTINCT MAX(lab10.students.student_id) AS student_id,
552   MAX(lab10.students.student_fname) AS student_fname,
553   MAX(lab10.students.student_lname) AS student_lname,
554   MAX(lab10.students.student_phone) AS student_phone,
555   MAX(lab10.students.student_dob) AS student_dob,
556   MAX(lab10.students.record_date) AS record_date,
557   MAX(lab11.grades.grade_id) AS grade_id,
558   MAX(lab11.grades.student_grade) AS student_grade,
559   MAX(lab11.grades.grade_comment) AS grade_comment,
560   MAX(lab11.grades.grade_timestamp) AS grade_timestamp
561                                                  -- calling the maximum value of
562                                                  -- `grade_timestamp` for latest
563                                                  -- transaction of each
564                                                  -- `lab11.grades.student_id`
565 FROM lab11.grades
566 INNER JOIN lab10.students
567   ON lab11.grades.student_id = lab10.students.student_id
568 GROUP BY lab11.grades.student_id;
569
570
571 /* ********************************************************************
572           As an alternative, we can use an alias for each table.
```

```
573
574                     `g` for `lab11.grades`
575                     `s` for `lab10.students`
576   ********************************************************************** */
577
578  SELECT DISTINCT MAX(s.student_id) AS student_id,
579    MAX(s.student_fname) AS student_fname,
580    MAX(s.student_lname) AS student_lname,
581    MAX(s.student_phone) AS student_phone,
582    MAX(s.student_dob) AS student_dob,
583    MAX(s.record_date) AS record_date,
584    MAX(g.grade_id) AS grade_id,
585    MAX(g.student_grade) AS student_grade,
586    MAX(g.grade_comment) AS grade_comment,
587    MAX(g.grade_timestamp) AS grade_timestamp
588  FROM lab11.grades AS g
589  INNER JOIN lab10.students AS s
590    ON g.student_id = s.student_id
591  GROUP BY g.student_id;
592
593
594  /* ***************************************************************************
595   3. LAB #12 (Procedures)
596      3.01. Understanding that the following is the structure for a procedure
597            (https://techonthenet.com/sql_server/procedures.php)
598
599                      CREATE PROCEDURE procedure_name [@input_param data_type]
600                      AS
601                      BEGIN
602                        [DECLARE @output_param data_type
603                        SET @output_param = some_value]
604                        executable_code
605                      END;
606
607            that we EXECUTE (EXEC) in order for it to run,
608
609                      EXEC procedure_name [@input_param]
610
611            write a procedure `strings2_sp` in schema `lab12` in database `labs`
612            to concatenate two (2) strings with an empty space (` `) between the
613            two strings.
614
615            HINT: two (2) input parameters to produce one (1) output parameter
616                  with the minimal size of the sum of the sizes of the first input
617                  parameter and the second input parameter
618
619      3.02. To test that procedure `lab12.strings2_sp` works, execute the
620            procedure passing first name and last name.
621
622            HINT: EXEC procedure_name(@in_param1, @in_param2)
623   ********************************************************************** */
624
```

```
625 CREATE SCHEMA lab12;
626
627 CREATE PROCEDURE lab12.string2_sp @in_string1 VARCHAR(50),
628   @in_string2 VARCHAR(50)
629 AS
630 BEGIN
631   DECLARE @out_string VARCHAR(101)              -- to accept VARCHAR(50) for
632                                                 -- `@in_string1`, VARCHAR(1)
633                                                 -- for a space + VARCHAR(50)
634                                                 -- for `@in_string2`
635   SET @out_string = CONCAT (
636       @in_string1,
637       ' ',
638       @in_string2
639       )
640   PRINT @out_string
641 END;
642
643
644 /* ************************************************************************
645     3.03. Then we execute procedure `lab12.string2_sp` passing two (2) values.
646           Passing more or fewer values will return an error.
647
648                     Msg 201, Level 16, State 4, Procedure lab12.string2_sp,
649                     Line 0 [Batch Start Line 53]
650                     Procedure or function 'string2_sp' expects parameter
651                     '@in_string2', which was not supplied.
652  ************************************************************************ */
653
654 EXEC lab12.string2_sp 'John', 'Smith';
655
656
657 /* ************************************************************************
658  4. LAB #13 (Functions)
659     4.01. Understanding that the following is the structure for a function
660           (https://techonthenet.com/sql_server/functions.php)
661
662                     CREATE FUNCTION funtion_name (@input_param data_type)
663                     RETURNS data_type
664                     AS
665                     BEGIN
666                       DECLARE @output_param data_type
667                       SET @output_param = some_value
668                       executable_code
669                       RETURN output_param
670                     END;
671
672           that affects a field or other value,
673
674                     funtion_name(field)
675
676           write a function `strings2_udf()` in schema `lab13` in database `labs`
```

```
677           to concatenate two (2) strings with an empty space (` `) between the
678           two strings.
679
680           HINT: two (2) input parameters to produce one (1) output parameter
681                 with the minimal size of the sum of the sizes of the first
682                 input parameter and the second input parameter
683
684     4.02. To test that function `lab13.strings2_udf()` works, write a query
685           calling all values from `AP1.ContactUpdates` using function
686           `lab13.string2_udf()` on `first_name` and `last_name`.
687
688           HINT: SELECT function_name(@in_param1, @in_param2)
689  ************************************************************************** */
690
691  CREATE SCHEMA lab13;
692
693  CREATE FUNCTION lab13.string2_udf (
694    @in_string1 VARCHAR(50),
695    @in_string2 VARCHAR(50)
696    )
697  RETURNS VARCHAR(101)                              -- same datatype and size as
698                                                    -- `@out_string`, in this case
699  AS
700  BEGIN
701    DECLARE @out_string VARCHAR(101)                -- to accept VARCHAR(50) for
702                                                    -- `@in_string1`, VARCHAR(1)
703                                                    -- for a space + VARCHAR(50)
704                                                    -- for `@in_string2`
705    SET @out_string = CONCAT (
706        @in_string1,
707        ' ',
708        @in_string2
709        )
710    RETURN @out_string
711  END;
712
713
714  /* ************************************************************************
715      4.03. Then we use function `lab13.string2_udf` passing two (2) values.
716            Note that passing more or fewer values will return an error.
717
718                    Msg 313, Level 16, State 2, Line 101
719                    An insufficient number of arguments were supplied for the
720                    procedure or function lab13.string2_udf.
721   ************************************************************************** */
722
723  SELECT lab13.string2_udf('John', 'Smith');
724
725
726  /* ************************************************************************
727   5. LAB #14 (Functions)
728      5.01. Make a function to dress up phone numbers as `(xxx) xxx-xxxx` in
```

```
729           schema `lab14`.
730  ********************************************************************** */
731
732  CREATE SCHEMA lab14;
733
734  CREATE FUNCTION lab14.phones_udf (@in_phone VARCHAR(15))
735  RETURNS VARCHAR(15)                              -- 1. need to remember that a
736                                                   --    function RETURNS a value
737  AS
738  BEGIN
739    DECLARE @out_phone VARCHAR(15)
740    SET @out_phone = CASE                          -- 2. `CASE` clause to check if
741        WHEN @in_phone IS NOT NULL                 --    `CONCAT` needs to be run
742          OR @in_phone <> ''
743          OR @in_phone NOT LIKE ('(%)%-%')        -- 3. checking if phone is
744                                                   --    already formatted
745          THEN CONCAT (
746              '(',
747              LEFT(@in_phone, 3),
748              ') ',
749              SUBSTRING(@in_phone, 4, 3),
750              '-',
751              RIGHT(@in_phone, 4)
752              )
753        ELSE @in_phone
754        END                                       -- 4. ending/closing `CASE`
755                                                  --    clause
756    RETURN @out_phone                             -- 5. returning value of
757                                                  --    function
758  END;                                            -- 6. ending/closing function
759
760
761  /* *********************************************************************
762      5.03. Use function `lab13.string2_udf` from the previous lab passing two
763            (2) values when querying `SF21SQL1001.AP1.Vendors`.
764
765            Since accessing another objects in another database, you need to call
766            the full name the function (`labs.lab13.string2_udf`) and/or the
767            table (`SF21SQL1001.AP1.Vendors`) depending in which database you are
768            in.
769   ********************************************************************** */
770
771  SELECT SF21SQL1001.AP1.Vendors.VVendorID,
772    SF21SQL1001.AP1.Vendors.VendorName,
773    labs.lab13.string2_udf(SF21SQL1001.AP1.Vendors.VendorAddress1,
774      SF21SQL1001.AP1.Vendors.VendorAddress2)     -- 1. using function
775      AS VendorAddress,                           --    `labs.lab13.string2_udf`
776                                                  --    on `VendorAddress1` and
777                                                  --    `VendorAddress2`
778    SF21SQL1001.AP1.Vendors.VendorCity,
779    SF21SQL1001.AP1.Vendors.VendorState,
780    SF21SQL1001.AP1.Vendors.VendorZipCode,
```

```
781    labs.lab14.phones_udf(VendorPhone)            -- 2. using function
782      AS VendorPhone                              --    `labs.lab14.phones_udf`
783  FROM SF21SQL1001.AP1.Vendors;
784
785
786  /* ****************************************************************************
787            As an alternative, we can use an alias for the table to avoid
788            repeating the long name of `SF21SQL1001.AP1.Vendors` throughout the
789            query.
790
791                      `v` for `SF21SQL1001.AP1.Vendors`
792   **************************************************************************** */
793
794  SELECT v.VVendorID,
795    v.VendorName,
796    labs.lab13.string2_udf(v.VendorAddress1, v.VendorAddress2) AS VendorAddress,
797    v.VendorCity,
798    v.VendorState,
799    v.VendorZipCode,
800    labs.lab14.phones_udf(VendorPhone) AS VendorPhone
801  FROM SF21SQL1001.AP1.Vendors AS v;
802
803
804  /* ****************************************************************************
805   6. This marks the end of new material.
806
807      6.01. As a developer, you should have a list of resources -- websites,
808            books or people whom you can contact for help.  The following is only
809            a list of resources -- not a recommendation of goods and/or services.
810
811            Analytics Vidhya (data science community)
812            https://analyticsvidhya.com/
813
814            Apache Spark - Unified Analytics Engine
815            https://spark.apache.org/
816
817            Apache Spark - Unified Analytics Engine - Spark SQL & DataFrames
818            https://spark.apache.org/sql/
819
820            Azure Cosmos DB
821            https://azure.microsoft.com/en-us/services/cosmos-db/
822
823            Azure SQL - Azure SQL documentation - Microsoft Docs
824            https://docs.microsoft.com/en-us/azure/azure-sql/
825
826            Cockroach Labs - CockroachDB
827            https://cockroachlabs.com/
828
829            DBeaver - Universal Database Tool
830            https://dbeaver.com/
831
832            DBeaver Community (client)
```

```
            https://dbeaver.io/

            EverSQL - SQL Query Optimizer Tool Online
            https://eversql.com/sql-syntax-check-validator/

            HeidiSQL (client)
            https://heidisql.com/

            Infrastructure as SQL (iaSQL)
            https://iasql.com/

            MariaDB (MySQL fork, not related to Oracle)
            https://mariadb.org/

            Microsoft Azure
            https://portal.azure.com/

            Microsoft Azure - Quickstart Templates
            https://azure.microsoft.com/en-us/resources/templates/

            Microsoft Power Automate
            https://flow.microsoft.com/en-us/desktop/

            Microsoft Power BI (business intelligence)
            https://powerbi.microsoft.com/

            Microsoft SQL Server
            https://microsoft.com/en-us/sql-server/

            Microsoft SQL Server - Get Started
            https://microsoft.com/en-us/sql-server/developer-get-started/

            MongoDB
            https://mongodb.com/

            MongoDB Blog
            https://mongodb.com/blog

            mycli (CLI MariaDB, MySQL & Percona)
            https://mycli.net/

            MySQL (Oracle)
            https://mysql.com/

            Oracle
            http://oracle.com/

            phpMyAdmin (administration tool for MySQL/MariaDB)
            https://phpmyadmin.net/

            Poor SQL (code formatter)
            https://poorsql.com/
```

```
885
886             PostgreSQL
887             https://postgresql.org/
888
889             Slack - codebar - sql
890             https://app.slack.com/client/T08CJBA82/CHPE04RU7
891
892             SQLite
893             https://sqlite.org/
894
895             SQLZOO
896             https://sqlzoo.net/
897
898             Tech on the Net - SQL Server
899             https://techonthenet.com/sql_server/
900
901             Vespa (big data AI, Oath/Yahoo)
902             http://vespa.ai/
903
904   ************************************************************************
905   https://folvera.commons.gc.cuny.edu/?p=1241
906   ************************************************************************ */
```