```
/* ************************************************************************
          INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
                    WS24SQL10001, 2024/03/11 - 2024/04/10
                  https://folvera.commons.gc.cuny.edu/?cat=34
 ************************************************************************

  SESSION #9 (2024/04/08):  CREATING DATABASE OBJECTS

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

 1. LAB #10 (QUICK REVIEW)
    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 (                   -- 01. rule of thumb: table
                                                --     names in plural
  student_id INT NULL,                          -- 02. declared as INT;  can
                                                --     accept NULL (can have no
                                                --     value)
  student_fname VARCHAR(50) NULL,               -- 03. declared as VARCHAR(50);
                                                --     can accept NULL (can
                                                --     have no value)
  student_lname VARCHAR(50) NULL,               -- 04. declared as VARCHAR(50);
                                                --     can accept NULL (can
                                                --     have no value)
  student_phone VARCHAR(15) NULL,               -- 05. declared as VARCHAR(50);
                                                --     can accept NULL (can
                                                --     have no value)
  student_dob DATE NULL,                        -- 06. declared as DATE
                                                --
                                                --     DATETIME 4/08/2024 21:54
                                                --     DATE     4/08/2024
                                                --     TIME               21:54
                                                --
                                                --     can accept NULL (can
                                                --     have no value)
  record_date DATE NULL                         -- 07. 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()                                     -- 01. built-in function to
```

```
                                                --     retrieve system DATETIME
  ),
  (
  2,
  'Mary',
  'Jones',
  '212-555-1000',
  '1983/05/16',
  GETDATE()
  ),
  (
  3,
  'Peter',
  'Johnson',
  NULL,                                         -- 02. inserting empty strings
                                                --     (``) or NULL since we
                                                --     have no values for
                                                --     fields in order to
                                                --     insert same number of
                                                --     values as columns
  '06/01/1980',
  GETDATE()
  );


/* ************************************************************************
        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).
 ************************************************************************ */

INSERT INTO lab10.students (
  student_id,                                   -- 01. inserting values to only
  student_fname,                                --     four (4) columns;
  student_lname,                                --     indicating which four
  record_date                                   --     (4) columns
  )
VALUES (
  4,                                            -- 02. values to be inserted in
  'Smith',                                      --     columns `student_id`,
  'Tom',                                        --     `student_fname`,
  GETDATE()                                     --     `student_lname` and
  );                                            --     `record_date` receiving
                                                --     value from `GETDATE()`


/* ************************************************************************
        In the example below, we insert row 6 before 5.

        The values in `student_id` (the row identifier) are unique, but they
        do not need to be in order.

        If we need to insert values in `student_id` automatically in
        incremental order, we would need to use `IDENTITY(1,1)` as part of
        the table structure.  The first integer indicates that the first
        value as 1.  The second integer indicates that the value is
        incremented by 1.  Refer to
        https://www.w3schools.com/sql/sql_autoincrement.asp for more
        information.

                CREATE TABLE lab10.students (
                  student_id INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
                  student_fname VARCHAR(50) NULL,
                  student_lname VARCHAR(50) NULL,
                  student_phone VARCHAR(15) NULL,
                  student_dob DATE NULL,
                  record_date DATE NULL
                  );
 ************************************************************************ */
```

```
139
140 INSERT INTO lab10.students
141 VALUES (
142   6,
143   'John',
144   'Scott',
145   '',                                          -- 01. inserting empty strings
146   '',                                          --     (``) or NULL since we
147                                                --     have no values for
148                                                --     fields in order to
149                                                --     insert same number of
150                                                --     values as columns
151
152   GETDATE()                                    -- 02. built-in function to
153                                                --    retrieve system DATETIME
154   ),
155   (
156   5,
157   'Mary Ann',
158   'Saunders',
159   '',                                          -- 03. inserting empty strings
160   '',                                          --     (``) or NULL since we
161                                                --     have no values for
162                                                --     fields in order to
163                                                --     insert same number of
164                                                --     values as columns
165
166   GETDATE()                                    -- 04. built-in function to
167                                                --    retrieve system DATETIME
168   );
169
170
171 /* ************************************************************************
172           We can also delete/destroy data objects.
173
174           For the time being, we will work with tables
175           (https://techonthenet.com/sql_server/tables/drop_table.php).
176
177           Once an object is deleted, there is no way to rescue the data
178           (`ROLLBACK`) unless first creating a `SAVEPOINT`
179           (https://technet.microsoft.com/en-us/library/ms178157.aspx).
180
181           In the example below, we destroy (`DROP`) table `lab10.students`
182           understanding that, once we do, we cannot recover the structure or
183           the data.
184  ************************************************************************ */
185
186 DROP TABLE lab10.students;
187
188
189 /* ************************************************************************
190           In the case of tables, we can destroy (`TRUNCATE`) the data in the
191           table without affecting the structure of the table understanding
192           that, once we do, we cannot recover the data.
193  ************************************************************************ */
194
195 TRUNCATE TABLE lab10.students;
196
197
198 /* ************************************************************************
199           We can also modify (`ALTER`) data objects
200           (https://techonthenet.com/sql_server/tables/alter_table.php).
201
202           ADD       to add a column to a table
203
204           DROP      to delete a column to a table
205
206           ALTER     to change the data type or size of a column
207
```

```
208
209       1.03. Change the structure of table `lab10.students`.
210
211             * Add `email` as VARCHAR(100).
212             * Drop `Email` because we want a different name.  Remember that there
213               is no means to rename any object.
214             * Add `student_email` as VARCHAR(100).
215             * Alter the size of `student_email` to 50.
216             * Alter `student_id` to `NOT NULL`.
217             * Alter `record_date`, `student_fname` and `student_lname` to
218               `NOT NULL`.
219             * Alter `student_id` to `VARCHAR(5)`.
220             * Try to change `student_fname` to FLOAT.  Note you will get an error
221               since a string cannot be made into a number.
222     ********************************************************************** */
223
224   ALTER TABLE lab10.students                          -- 01. adding new column
225   ADD email VARCHAR(100);                             --     `Email`;  no need to
226                                                       --     specify that we are
227                                                       --     adding a column
228
229   ALTER TABLE lab10.students                          -- 02. dropping (deleting)
230   DROP COLUMN email;                                  --     column `Email` as there
231                                                       --     is no SQL statement to
232                                                       --     rename data objects;
233                                                       --     must specify that we are
234                                                       --     dropping a column
235
236   ALTER TABLE lab10.students                          -- 03. adding new (replacement)
237   ADD student_email VARCHAR(100);                     --     column `student_email`;
238                                                       --     no need to specify that
239                                                       --     we are adding a column
240
241   ALTER TABLE lab10.students                          -- 04. altering column with new
242   ALTER COLUMN student_email VARCHAR(50) NULL;        --     data type VARCHAR(50)
243                                                       --     from VARCHAR(100) and
244                                                       --     `NOT NULL`;  must
245                                                       --     specify that we are
246                                                       --     altering a column
247
248   ALTER TABLE lab10.students                          -- 05. altering column as
249   ALTER COLUMN student_id INT NOT NULL;               --     `NOT NULL`;  must
250                                                       --     specify that we are
251                                                       --     altering a column
252
253   ALTER TABLE lab10.students                          -- 06. altering column with new
254   ALTER COLUMN record_date DATETIME NOT NULL;         --     data type DATETIME from
255                                                       --     DATE and `NOT NULL`;
256                                                       --     must specify that we are
257                                                       --     altering a column
258
259   ALTER TABLE lab10.students                          -- 07. altering column with new
260   ALTER COLUMN student_fname VARCHAR(25) NOT NULL;    --     data type VARCHAR(25)
261                                                       --     from VARCHAR(50) and
262                                                       --     `NOT NULL`;  must
263                                                       --     specify that we are
264                                                       --     altering a column
265
266   ALTER TABLE lab10.students                          -- 08. altering column with new
267   ALTER COLUMN student_lname VARCHAR(25) NOT NULL;    --     data type VARCHAR(25)
268                                                       --     from VARCHAR(50) and
269                                                       --     `NOT NULL`;  must
270                                                       --     specify that we are
271                                                       --     altering a column
272
273   ALTER TABLE lab10.students                          -- 09. altering column with new
274   ALTER COLUMN student_id VARCHAR(5);                 --     data type VARCHAR(5)
275                                                       --     from INT;  no error
276                                                       --     during conversion;  must
```

```
                                                --     specify that we are
                                                --     altering a column

ALTER TABLE lab10.students                      -- 10. altering column back to
ALTER COLUMN student_id INT NOT NULL;           --     data type INT from
                                                --     VARCHAR(5);  no error
                                                --     during conversion;  must
                                                --     specify that we are
                                                --     altering a column

ALTER TABLE lab10.students                      -- 11. trying to alter column
ALTER COLUMN student_fname FLOAT;               --     to data type FLOAT from
                                                --     VARCHAR(25);  conversion
                                                --     failure due to format
                                                --     incompatibility (letters
                                                --     to numbers)


/* ***************************************************************************
    1.04. Update (overwrite) the value of column `student_phone` passing value
          `No Number` where there is no value (`IS NULL`) or there is an empty
          space (` `) or empty string (``).
 *************************************************************************** */

UPDATE lab10.students
SET student_phone = 'No Number'
WHERE student_phone IS NULL                     -- 01. checking for NULL
  OR student_phone = ' '                        -- 02. checking for a space
  OR student_phone = '';                        -- 03. checking for an empty
                                                --     string


/* ***************************************************************************
    1.05. Update (overwrite) the value of column `student_email` passing the
          value of the concatenation of `student_fname` and `student_lname`
          with a period (`.`) between the two columns -- for example,
          `john.smith@foobar.foo` for `student_fname` with value of `John` and
          `student_lname` with value of `Smith`.
 *************************************************************************** */

UPDATE lab10.students
SET student_email = LOWER(CONCAT (
      student_fname,
      '.',
      student_lname,
      '@foobar.foo'
      ));


/* ***************************************************************************
    1.06. In the example below, we UPDATE column `record_date` where the field
          is NULL or has an empty space (``) with value from `GETDATE()`.
 *************************************************************************** */

UPDATE lab10.students
SET record_date = GETDATE()
WHERE record_date IS NULL
  OR record_date = '';


/* ***************************************************************************
          In the example below, we can UPDATE `student_dob` to `1980/01/23`
          where `student_id` is `1`.
 *************************************************************************** */

UPDATE lab10.students
SET student_dob = '1980/01/23'
WHERE student_id = 1;

```

```
/* ***************************************************************************
 2. LAB #11 (CREATING OBJECTS)
    2.01. In schema `lab11` in database `labs`, create table `grades`
          (referenced as `labs.lab11.grades`) with the following structure.

                  grade_id      INT           NOT NULL    UNIQUE
                  student_id    INT           NOT NULL
                  student_grade FLOAT         NOT NULL
                  grade_comment VARCHAR(255)  NULL
 *************************************************************************** */

CREATE SCHEMA lab11;

CREATE TABLE lab11.grades (
  grade_id INT NOT NULL UNIQUE,
  student_id INT NOT NULL,
  student_grade FLOAT NOT NULL,
  grade_comment VARCHAR(255) NULL
  );


/* ***************************************************************************
    2.02. Then populate the table with some data of your choice.
 *************************************************************************** */

INSERT INTO lab11.grades
VALUES (
  1,
  1,
  80,
  'He missed the midterm.'
  ),
  (
  2,
  3,
  65,
  'He slept in class.'
  ),
  (
  3,
  2,
  98,
  ''
  );


/* ***************************************************************************
    2.03. Since we have shared (`student_id`) data between `labs.lab11.grades`
          and `labs.lab10.students` from the previous lab, we can retrieve all
          the data from `labs.lab10.students` (main) and any related data from
          `labs.lab11.grades` (secondary) without duplicate rows (`SELECT
          DISTINCT`).
 *************************************************************************** */

SELECT DISTINCT lab10.students.student_id,
  lab10.students.student_fname,
  lab10.students.student_lname,
  lab10.students.student_phone,
  lab10.students.student_dob,
  lab10.students.record_date,
  lab11.grades.grade_id,
  -- lab11.grades.student_id AS Expr1,
  lab11.grades.student_grade,
  lab11.grades.grade_comment
FROM lab10.students
LEFT OUTER JOIN lab11.grades
  ON lab10.students.student_id = lab11.grades.student_id
ORDER BY student_lname;
```

```
415
416
417  /* ****************************************************************************
418           As an alternative, we can use an alias (`AS`) for each table to make
419           our code tidier and to avoid repeating the database and schema before
420           each table (`<database>.<schema>.<table>` or `<schema>.<table>`)
421           throughout the query.
422
423                    `s` for `lab10.students`
424                    `g` 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 (`AS`) for each table to make
478           our code tidier and to avoid repeating the database and schema before
479           each table (`<database>.<schema>.<table>` or `<schema>.<table>`)
480           throughout the query.
481
482                    `s` for `lab10.students`
483                    `g` for `lab11.grades`
```

```
 ********************************************************************************* */

ALTER VIEW lab11.students_grades_vw
AS
SELECT DISTINCT s.student_id,
  s.student_fname,
  s.student_lname,
  s.student_phone,
  s.student_dob,
  s.record_date,
  g.grade_id,
  -- g.student_id AS Expr1,
  g.student_grade,
  g.grade_comment
FROM lab10.students AS s
LEFT OUTER JOIN lab11.grades AS g
  ON s.student_id = g.student_id
-- ORDER BY student_lname;


/* ********************************************************************************
          Although we can UPDATE a record when we change any existing value,
          there are situations where we need to keep track every transaction
          historically -- for example, to keep track of bank transactions.  In
          such scenario, we should INSERT a new record for each transaction
          with a separate column to record the time stamp.

          First we would need to add a column for the time stamp.

          Then we would push the value of `GETDATE()` into the new column.  Of
          course, for this to work all records should have a value in new
          column.

          To retrieve the latest record for student, we would need to call the
          `MAX()` value of all fields in the query and group the results by an
          identifier -- for example, `student_id` in the example below.
 ********************************************************************************* */

ALTER TABLE lab11.grades                        -- 01. adding `grade_timestamp`
ADD grade_timestamp DATETIME;                   --     to table `lab11.grades`

UPDATE lab11.grades                             -- 02. inserting values into
SET grade_timestamp = GETDATE();                --     `grade_timestamp`

INSERT INTO lab11.grades                        -- 03. inserting two new
VALUES (                                        --     records at the same time
  1,                                            --     hence writing the same
  1,                                            --     value of `GETDATE()` to
  90,                                           --     both records
  'teacher''s pet'
  ),
  (
  5,
  2,
  85,
  '',
  GETDATE()
  );

INSERT INTO lab11.grades                        -- 04. inserting a new record
VALUES (                                        --     for `student_id` 1
  1,
  8,
  95,
  'grade change',
  GETDATE()
  );

SELECT DISTINCT MAX(lab10.students.student_id) AS student_id,
```

```
  MAX(lab10.students.student_fname) AS student_fname,
  MAX(lab10.students.student_lname) AS student_lname,
  MAX(lab10.students.student_phone) AS student_phone,
  MAX(lab10.students.student_dob) AS student_dob,
  MAX(lab10.students.record_date) AS record_date,
  MAX(lab11.grades.grade_id) AS grade_id,
  MAX(lab11.grades.student_grade) AS student_grade,
  MAX(lab11.grades.grade_comment) AS grade_comment,
  MAX(lab11.grades.grade_timestamp) AS grade_timestamp
                                                -- 05. calling the maximum
                                                --     value of
                                                --     `grade_timestamp` for
                                                --     latest transaction
FROM lab11.grades
INNER JOIN lab10.students
  ON lab11.grades.student_id = lab10.students.student_id
GROUP BY lab11.grades.student_id;


/* ***************************************************************************
          As an alternative, we can use an alias (`AS`) for each table to make
          our code tidier and to avoid repeating the database and schema before
          each table (`<database>.<schema>.<table>` or `<schema>.<table>`)
          throughout the query.

                    `g` for `lab11.grades`
                    `s` for `lab10.students`
 *************************************************************************** */

SELECT DISTINCT MAX(s.student_id) AS student_id,
  MAX(s.student_fname) AS student_fname,
  MAX(s.student_lname) AS student_lname,
  MAX(s.student_phone) AS student_phone,
  MAX(s.student_dob) AS student_dob,
  MAX(s.record_date) AS record_date,
  MAX(g.grade_id) AS grade_id,
  MAX(g.student_grade) AS student_grade,
  MAX(g.grade_comment) AS grade_comment,
  MAX(g.grade_timestamp) AS grade_timestamp
FROM lab11.grades AS g
INNER JOIN lab10.students AS s
  ON g.student_id = s.student_id
GROUP BY g.student_id;


/* ***************************************************************************
 3. LAB #12 (PROCEDURES)
    3.01. Understanding that the following is the structure for a procedure
          (https://techonthenet.com/sql_server/procedures.php)

                    CREATE PROCEDURE procedure_name [@input_param data_type]
                    AS
                    BEGIN
                      [DECLARE @output_param data_type
                      SET @output_param = some_value]
                      executable_code
                    END;

          that we EXECUTE (EXEC) in order for it to run,

                    EXEC procedure_name [@input_param]

          write a procedure `strings2_sp` in schema `lab12` in database `labs`
          to concatenate two (2) strings with an empty space (` `) between the
          two strings.

          HINT: two (2) input parameters to produce one (1) output parameter
                with the minimal size of the sum of the sizes of the first
                input parameter and the second input parameter
```

```
622
623       3.02. To test that procedure `lab12.strings2_sp` works, execute the
624             procedure passing first name and last name.
625
626             HINT: EXEC procedure_name(@in_param1, @in_param2)
627    ********************************************************************** */
628
629   CREATE SCHEMA lab12;
630
631   CREATE PROCEDURE lab12.string2_sp                -- 01. start of procedure
632     @in_string1 VARCHAR(50),                       --     accepting two (2) input
633     @in_string2 VARCHAR(50)                        --     parameters
634   AS
635   BEGIN
636     DECLARE @out_string VARCHAR(101)               -- 02. accepting VARCHAR(50)
637                                                    --     for `@in_string1`,
638                                                    --     VARCHAR(1) for a space
639                                                    --     and VARCHAR(50) for
640                                                    --     `@in_string2`
641     SET @out_string = CONCAT (                     -- 03. push value returned from
642         @in_string1,                               --     the concatenation of the
643         ' ',                                       --     values of the three (3)
644         @in_string2                                --     input parameters into
645         )                                          --     output parameter
646                                                    --     `@out_string`
647     PRINT @out_string                              -- 04. print value of
648                                                    --     `@out_string` to console
649   END;
650
651
652   /* ***********************************************************************
653       3.03. Then we execute procedure `lab12.string2_sp` passing two (2) values.
654             Passing more or fewer values will return an error.
655
656                       Msg 201, Level 16, State 4, Procedure lab12.string2_sp,
657                       Line 0 [Batch Start Line 53]
658                       Procedure or function 'string2_sp' expects parameter
659                       '@in_string2', which was not supplied.
660    ********************************************************************** */
661
662   EXEC lab12.string2_sp 'John', 'Smith';
663
664
665   /* ***********************************************************************
666    4. LAB #13 (FUNCTIONS)
667       4.01. Understanding that the following is the structure for a function
668             (https://techonthenet.com/sql_server/functions.php)
669
670                       CREATE FUNCTION function_name (@input_param data_type)
671                       RETURNS data_type
672                       AS
673                       BEGIN
674                         DECLARE @output_param data_type
675                         SET @output_param = some_value
676                         executable_code
677                         RETURN output_param
678                       END;
679
680             that affects a field or other value,
681
682                       function_name(field)
683
684             write function `strings2_udf()` in schema `lab13` in database `labs`
685             to concatenate two (2) strings with an empty space (` `) between the
686             two strings.
687
688             HINT: two (2) input parameters to produce one (1) output parameter
689                   with the minimal size of the sum of the sizes of the first
690                   input parameter and the second input parameter
```

```
691
692       4.02. To test that function `lab13.strings2_udf()` works, write a query
693             calling all values from `AP1.ContactUpdates` using function
694             `lab13.string2_udf()` on `first_name` and `last_name`.
695
696             HINT: SELECT function_name(@in_param1, @in_param2)
697    ********************************************************************** */
698
699   CREATE SCHEMA lab13;
700
701   CREATE FUNCTION lab13.string2_udf (                -- 01. start of function
702     @in_string1 VARCHAR(50),                         --     accepting two (2) input
703     @in_string2 VARCHAR(50)                          --     parameters
704     )
705   RETURNS VARCHAR(101)                               -- 03. returning same datatype
706                                                      --     and size as output
707                                                      --     parameter `@out_string`,
708                                                      --     in this case
709   AS
710   BEGIN
711     DECLARE @out_string VARCHAR(101)                 -- 04. accepting VARCHAR(50)
712                                                      --     for `@in_string1`,
713                                                      --     VARCHAR(1) for a space
714                                                      --     and VARCHAR(50) for
715                                                      --     `@in_string2`
716     SET @out_string = CONCAT (                       -- 05. push value returned from
717         @in_string1,                                 --     the concatenation of the
718         ' ',                                         --     values of the three (3)
719         @in_string2                                  --     input parameters into
720         )                                            --     output parameter
721                                                      --     `@out_string`
722   END;
723
724
725   /* ********************************************************************
726       4.03. Then we use function `lab13.string2_udf` passing two (2) values.
727             Note that passing more or fewer values will return an error.
728
729                     Msg 313, Level 16, State 2, Line 101
730                     An insufficient number of arguments were supplied for the
731                     procedure or function lab13.string2_udf.
732    ********************************************************************** */
733
734   SELECT lab13.string2_udf('John', 'Smith');
735
736
737   /* ********************************************************************
738    5. LAB #14 (FUNCTIONS)
739       5.01. Make a function to dress up phone numbers as `(xxx) xxx-xxxx` in
740             schema `lab14`.
741    ********************************************************************** */
742
743   CREATE SCHEMA lab14;
744
745   CREATE FUNCTION lab14.phones_udf (@in_phone VARCHAR(15))
746   RETURNS VARCHAR(15)                                -- 01. need to remember that a
747                                                      --     function RETURNS a value
748   AS
749   BEGIN
750     DECLARE @out_phone VARCHAR(15)
751     SET @out_phone = CASE                            -- 02. `CASE` clause to check
752         WHEN @in_phone IS NOT NULL                   --     if `CONCAT` needs to be
753           OR @in_phone <> ''                         --     run
754           OR @in_phone NOT LIKE ('(%)%-%')           -- 03. checking if phone is
755                                                      --     already formatted
756           THEN CONCAT (
757               '(',
758               LEFT(@in_phone, 3),
759               ') ',
```

```
760             SUBSTRING(@in_phone, 4, 3),
761             '-',
762             RIGHT(@in_phone, 4)
763             )
764       ELSE @in_phone
765       END                                           -- 04. ending/closing `CASE`
766                                                     --     clause
767   RETURN @out_phone                                 -- 05. returning value of
768                                                     --     function
769   END;                                              -- 06. ending/closing function
770
771
772   /* ***************************************************************************
773       5.02. Use function `lab13.string2_udf` from the previous lab passing two
774             (2) values when querying `WS24SQL10001.AP1.Vendors`.
775
776             Since accessing another objects in another database, you need to call
777             the full name the function (`labs.lab13.string2_udf`) and/or the
778             table (`WS24SQL10001.AP1.Vendors`) depending in which database you
779             are in.
780    *************************************************************************** */
781
782   SELECT WS24SQL10001.AP1.Vendors.VVendorID,
783     WS24SQL10001.AP1.Vendors.VendorName,
784     labs.lab13.string2_udf(WS24SQL10001.AP1.Vendors.VendorAddress1,
785       WS24SQL10001.AP1.Vendors.VendorAddress2)      -- 01. using function
786       AS VendorAddress,                             --     `labs.lab13.string2_udf`
787                                                     --     on `VendorAddress1` and
788                                                     --     `VendorAddress2`
789     WS24SQL10001.AP1.Vendors.VendorCity,
790     WS24SQL10001.AP1.Vendors.VendorState,
791     WS24SQL10001.AP1.Vendors.VendorZipCode,
792     labs.lab14.phones_udf(WS24SQL10001.AP1.Vendors.VendorPhone)
793                                                     -- 02. using function
794       AS VendorPhone                                --     `labs.lab14.phones_udf`
795   FROM WS24SQL10001.AP1.Vendors;
796
797
798   /* ***************************************************************************
799             As an alternative, we can use an alias (`AS`) for each table to make
800             our code tidier and to avoid repeating the database and schema before
801             each table (`<database>.<schema>.<table>` or `<schema>.<table>`)
802             throughout the query.
803
804                       `v` for `WS24SQL10001.AP1.Vendors`
805    *************************************************************************** */
806
807   SELECT v.VVendorID,
808     v.VendorName,
809     labs.lab13.string2_udf(v.VendorAddress1, v.VendorAddress2) AS VendorAddress,
810     v.VendorCity,
811     v.VendorState,
812     v.VendorZipCode,
813     labs.lab14.phones_udf(VendorPhone) AS VendorPhone
814   FROM WS24SQL10001.AP1.Vendors AS v;
815
816
817   /* ***************************************************************************
818    6. This marks the end of new material.
819
820       6.01. As a developer, you should have a list of resources -- websites,
821             books or people whom you can contact for help.  The following is only
822             a list of resources -- not a recommendation of goods and/or services.
823
824             Analytics Vidhya (data science community)
825             https://analyticsvidhya.com/
826
827             Apache Spark - Unified Analytics Engine
828             https://spark.apache.org/
```

Apache Spark - Unified Analytics Engine - Spark SQL & DataFrames
https://spark.apache.org/sql/

Azure Cosmos DB
https://azure.microsoft.com/en-us/services/cosmos-db/

Azure SQL - Azure SQL documentation - Microsoft Docs
https://docs.microsoft.com/en-us/azure/azure-sql/

Cockroach Labs - CockroachDB
https://cockroachlabs.com/

DBeaver - Universal Database Tool
https://dbeaver.com/

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/

```
898
899             PostgreSQL
900             https://postgresql.org/
901
902             Slack - codebar - sql
903             https://app.slack.com/client/T08CJBA82/CHPE04RU7
904
905             SQLite
906             https://sqlite.org/
907
908             SQLZOO
909             https://sqlzoo.net/
910
911             Tech on the Net - SQL Server
912             https://techonthenet.com/sql_server/
913
914             Vespa (big data AI, Oath/Yahoo)
915             http://vespa.ai/
916
917     ********************************************************************************
918     https://folvera.commons.gc.cuny.edu/?p=1295
919     ******************************************************************************** */
```