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

  SESSION #6 (2021/11/24): CREATING DATABASE OBJECTS

  1. Understanding data types
  2. Creating, dropping and altering databases, schemata, tables and columns
  3. Inserting values into tables and updating values
  4. Differences between `DROP`, `TRUNCATE` and `DELETE`
 ************************************************************************

 1. As a review, we understand that the most common joins we will use are the
    following.

                  +-----------------+
                  | LEFT    +-------+---------+
                  | JOIN    | INNER |         |
                  |         | JOIN  |  RIGHT  |
                  +---------+-------+  JOIN   |
                            +-----------------+

    1.1. `INNER JOIN` calls the data shared in both tables.  The data must be
         present in both table.  All other data is ignored.

    1.2. `LEFT JOIN` calls in the left table (called first) plus any related
         data found in the right table (second table).  This means that the
         right table does not need to have corresponding data.  In other
         words, if the right table does not have related data, nothing is
         returned (NULLs at the beginning of the dataset output).

         1.2.1. As such, we can ask for all data in `AP1.Vendors` (main), not
                necessarily from `AP1.Invoices` (secondary).  In this example,
                we are interested in all `AP1.Vendors` regardless of possible
                corresponding data in `AP1.Invoices`.  In other words, some
                vendors might not have sales.
 ************************************************************************ */

SELECT *
FROM AP1.Vendors                                -- main table called first
                                                --   (left)
LEFT JOIN AP1.Invoices                          -- secondary table called
                                                --   second (right), always in
                                                --   groups of two (2) tables
  ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;


/* ************************************************************************
    1.3. `RIGHT JOIN` calls in the right table (called second) plus any related
         data found in the left table (first table).  This means that the left
```

```
53          table does not need to have corresponding data.  In other words, if
54          the left table does not have related data, nothing is returned (NULLs
55          at the end of the dataset output).
56
57          1.3.1. As such, we can ask for all data in `AP1.Invoices` (main), not
58                 necessarily from `AP1.Vendors` (secondary).  In this example,
59                 we are interested in all `AP1.Invoices` regardless of possible
60                 corresponding data in `AP1.Vendors`.  In other words, some
61                 invoices might not have vendor data.
62  ********************************************************************** */
63
64  SELECT *
65  FROM AP1.Vendors                                  -- secondary table called first
66                                                    --   (left)
67  RIGHT JOIN AP1.Invoices                            -- main table called second
68                                                    --   (right), always in groups
69                                                    --   of two (2) tables
70    ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;
71
72
73  /* ********************************************************************************
74      1.4. On a personal note, `RIGHT JOIN` is a disorganized way to write code.
75           The example above could easily be called using `LEFT JOIN` ordering
76           the tables more appropriately.  Note that the order of `VendorID`
77           coming from `AP1.Invoices` and `AP1.Vendors.VendorID` makes no
78           difference.
79   ********************************************************************** */
80
81  SELECT *
82  FROM AP1.Invoices                                 -- main table called first
83                                                    --    (left)
84  LEFT JOIN AP1.Vendors                             -- secondary table called
85                                                     --   second (right), always in
86                                                     --   groups of two (2) tables
87    ON AP1.Invoices.VendorID = AP1.Vendors.VendorID;
88
89
90  /* ********************************************************************************
91   2. Before we start creating and altering data objects, we have to understand
92      data types (how data is stored).  These are the most often used data types.
93      Refer to https://msdn.microsoft.com/en-us/library/ms187752.aspx for more
94      information on data types in SQL Server.
95
96      2.1. INT        -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)
97                      https://technet.microsoft.com/en-us/library/ms187745.aspx
98
99      2.2. DECIMAL    fixed precision and scale numbers...
100                     10^38+1 through 10^38-1
101                     https://msdn.microsoft.com/en-us/library/ms187746.aspx
102
103                     instead of DOUBLE or FLOAT, indicating the whole value
104                     followed by the number of decimals where pi(1,10) can hold
```

```
                    3.1415926536 but not 3.14159265359 for its eleven (11)
                    decimal spaces

    2.3. VARCHAR(n) 2^31-1 bytes (2 GB);  variable-length, non-Unicode string
                    data, ASCII only
                    https://technet.microsoft.com/en-us/library/ms176089.aspx

                    not to be confused with NVARCHAR(n) -- variable-length,
                    2^31-1 bytes (2 GB), Unicode string data, not part of most
                    relational database management systems (RDBMS)
                    https://technet.microsoft.com/en-us/library/ms186939.aspx

    2.4. DATE       date
                    https://technet.microsoft.com/en-us/library/bb630352.aspx

    2.5. TIME       time
                    https://technet.microsoft.com/en-us/library/bb677243.aspx

    2.6. DATETIME   defines a date that is combined with a time of day with
                    fractional seconds that is based on a 24-hour clock
                    https://technet.microsoft.com/en-us/library/ms187819.aspx

    2.7. MONEY      money, not part of most relational database management
                    systems (RDBMS)
                    https://technet.microsoft.com/en-us/library/ms179882.aspx

    2.8. Conversion may only take place between data similar types.

                    +----------------------+---------------------------------+
                    | CONVERSION INPUT     | CONVERSION OUTPUT               |
                    +----------------------+---------------------------------+
                    | INT       to  DECIMAL | no loss, decimal spaces added   |
                    +----------------------+---------------------------------+
                    | DECIMAL   to  INT    | possible loss of decimal        |
                    |                      | spaces;  truncated, value not   |
                    |                      | rounded                         |
                    +----------------------+---------------------------------+
                    | DECIMAL   to  MONEY  | truncated/rounded to four       |
                    |                      | decimal spaces;  two decimal    |
                    |                      | spaces shown                    |
                    +----------------------+---------------------------------+
                    | DATETIME  to  DATE   | date only;  time dropped        |
                    +----------------------+---------------------------------+
                    | DATETIME  to  TIME   | time only;  date dropped        |
                    +----------------------+---------------------------------+
                    | INT                  | numeric data type loss;         |
                    | DECIMAL              | converted to text;  no longer   |
                    | DATETIME  to  VARCHAR| can be used in mathematical     |
                    | DATE          NVARCHAR| equations as it is no longer a  |
                    | TIME                 | numeric value                   |
                    +----------------------+---------------------------------+
                    |               INT    | straight conversion to proper   |
```

```
                    |               DECIMAL | data type as long as the        |
                    | VARCHAR   to  DATETIME| VARCHAR() field only has numbers|
                    | NVARCHAR      DATE    | and structure is correct (for   |
                    |               TIME    | example, text with value of     |
                    |                       | `2018/09/10` to DATE);  no      |
                    |                       | conversion if letters or special|
                    |                       | characters are present          |
                    +-----------------------+---------------------------------+
```

2.9. Refer to https://technet.microsoft.com/en-us/library/ms187912.aspx for information on approximate numeric data types -- FLOAT and REAL. If you are considering taking the certification, you should know the concept below and why Microsoft recommends not using them. Note that FLOAT is commonly used in other relational database management systems (RDBMS) like Oracle (http://oracle.com/) and in most programming languages including those distributed by Microsoft.

> ``The float and real data types are known as approximate data types. The behavior of float and real follows the IEEE 754 specification on approximate numeric data types.
> Approximate numeric data types do not store the exact values specified for many numbers; they store an extremely close approximation of the value. For many applications, the tiny difference between the specified value and the stored approximation is not noticeable. At times, though, the difference becomes noticeable. Because of the approximate nature of the float and real data types, do not use these data types when exact numeric behavior is required, such as in financial applications, in operations involving rounding, or in equality checks. Instead, use the integer, decimal, money, or smallmoney data types.
> Avoid using float or real columns in WHERE clause search conditions, especially the = and <> operators. It is best to limit float and real columns to > or < comparisons.
> The IEEE 754 specification provides four rounding modes: round to nearest, round up, round down, and round to zero. Microsoft SQL Server uses round up. All are accurate to the guaranteed precision but can result in slightly different floating-point values. Because the binary representation of a floating-point number may use one of many legal rounding schemes, it is impossible to reliably quantify a floating-point value.``
> https://technet.microsoft.com/en-us/library/ms187912.aspx

3. Now that we understand most common data types, we can start creating data objects (DATABASE, TABLE, etc.) and populating tables with data.

3.1. Note that no two objects of the same hierarchy can share the same name, for example a TABLE and a VIEW.

3.2. The following is a quick view of database hierarchy.

```
        SERVER:  ``A server is a computer program that provides a service
          |   to another computer programs (and its user).  In a data
```

```
209                |   center, the physical computer that a server program runs in
210                |   is also frequently referred to as a server.  That machine may
211                |   be a dedicated server or it may be used for other purposes as
212                |   well.``
213                |   https://whatis.techtarget.com/definition/server
214                |
215                +- DATABASE:  `A database is a collection of information that is
216                    |   organized so that it can be easily accessed, managed and
217                    |   updated.
218                    |   Data is organized into rows, columns and tables, and it
219                    |   is indexed to make it easier to find relevant
220                    |   information.  Data gets updated, expanded and deleted as
221                    |   new information is added.  Databases process workloads to
222                    |   create and update themselves, querying the data they
223                    |   contain and running applications against it.``
224                    |   https://searchsqlserver.techtarget.com/definition/database
225                    |
226                    +- SCHEMA:  ``1) In computer programming, a schema
227                        |   (pronounced SKEE-mah) is the organization or
228                        |   structure for a database.  The activity of data
229                        |   modeling leads to a schema.  (The plural form is
230                        |   schemata.  The term is from a Greek word for ``form``
231                        |   or ``figure.``  Another word from the same source is
232                        |   ``schematic.``)  The term is used in discussing both
233                        |   relational databases and object-oriented databases.
234                        |   The term sometimes seems to refer to a visualization
235                        |   of a structure and sometimes to a formal
236                        |   text-oriented description.
237                        |   Two common types of database schemata are the star
238                        |   schema and the snowflake schema.
239                        |   2) In another usage derived from mathematics, a
240                        |   schema is a formal expression of an inference rule
241                        |   for artificial intelligence (AI) computing.  The
242                        |   expression is a generalized axiom in which specific
243                        |   values or cases are substituted for each symbol in
244                        |   the axiom to derive a specific inference.``
245                        |   https://searchsqlserver.techtarget.com/definition/ ↵
                       schema
246                        |
247                        +- TABLES:  ``In computer programming, a table is a data
248                        |   |   structure used to organize information, just as
249                        |   |   it is on paper.``
250                        |   |   https://whatis.techtarget.com/definition/table
251                        |   |
252                        |   +- COLUMNS (FIELDS):  ``A field is an area in a fixed
253                        |   |   or known location in a unit of data such as a
254                        |   |   record, message header, or computer instruction
255                        |   |   that has a purpose and usually a fixed size.  In
256                        |   |   some contexts, a field can be subdivided  into
257                        |   |   smaller fields.``
258                        |   |   https://searchoracle.techtarget.com/definition/ ↵
                       field
```

```
259                       |   |
260                       |   +- PRIMARY KEY (PRIMARY KEYWORD):  ``A primary key,
261                       |   |   also called a primary keyword, is a key in a
262                       |   |   relational database that is unique for each
263                       |   |   record.  It is a unique identifier, such as a
264                       |   |   driver license number, telephone number
265                       |   |   (including area code), or vehicle identification
266                       |   |   number (VIN).  A relational database must always
267                       |   |   have one and only one primary key.  Primary keys
268                       |   |   typically appear as columns in relational
269                       |   |   database tables.``
270                       |   |   https://searchsqlserver.techtarget.com/definition/ ↵
                      primary-key
271                       |   |
272                       |   +- FOREIGN KEY:  ``A foreign key is a column or
273                       |       columns of data in one table that connects to the
274                       |       primary key data in the original table.
275                       |       To ensure the links between foreign key and
276                       |       primary key tables aren't broken, foreign key
277                       |       constraints can be created to prevent actions
278                       |       that would damage the links between tables and
279                       |       prevent erroneous data from being added to the
280                       |       foreign key column.``
281                       |       https://searchoracle.techtarget.com/definition/ ↵
                      foreign-key
282                       |
283                       +- VIEWS:  ``In a database management system, a view is a
284                       |     way of portraying information in the database.``
285                       |     https://whatis.techtarget.com/search/query
286                       |
287                       +- STRUCTURED (MODULAR) PROGRAMMING:  ``Structured
288                            |   programming (sometimes known as modular
289                            |   programming) is a subset of procedural
290                            |   programming that enforces a logical structure on
291                            |   the program being written to make it more
292                            |   efficient and easier to understand and modify.
293                            |   Certain languages such as Ada, Pascal, and dBASE
294                            |   are designed with features that encourage or
295                            |   enforce a logical program structure.``
296                            |   https://searchsoftwarequality.techtarget.com/ ↵
                      definition/structured-programming-modular-programming
297                            |
298                            +- FUNCTIONS:  ``In information technology, the term
299                            |   function (pronounced FUHNK-shun) has a number of
300                            |   meanings.  It's taken from the Latin ``functio``
301                            |   -- to perform.
302                            |   1) In its most general use, a function is what a
303                            |   given entity does in being what it is.
304                            |   2) In C language and other programming, a
305                            |   function is a named procedure that performs a
306                            |   distinct service.  The language statement that
307                            |   requests the function is called a function call.
```

```
                          |   Programming languages usually come with a
                          |   compiler and a set of ``canned`` functions that a
                          |   programmer can specify by writing language
                          |   statements.  These provided functions are
                          |   sometimes referred to as library routines.  Some
                          |   functions are self-sufficient and can return
                          |   results to the requesting program without help.
                          |   Other functions need to make requests of the
                          |   operating system in order to perform their
                          |   work.``
                          |   https://whatis.techtarget.com/definition/function
                          |
                          +- PROCEDURES:  ``A stored procedure is a set of
                              Structured Query Language (SQL) statements with
                              an assigned name, which are stored in a
                              relational database management system as a group,
                              so it can be reused and shared by multiple
                              programs.``
                              https://searchoracle.techtarget.com/definition/
                     stored-procedure

   4. Now that you have a better understanding of data types, we can start
      creating objects.

                      CREATE  obj_type   obj_name [some_code]

                      CREATE  DATABASE   db_name;

                      CREATE  SCHEMA     schema_name;

                      CREATE  TABLE      table_name
                        (
                          field_1 datatype_1 [attributes],
                          field_2 datatype_2 [attributes],
                          field_3 datatype_3 [attributes],
                          ...
                        );

                      CREATE  VIEW       view_table
                      AS
                        (
                          SELECT fields...
                          FROM table(s)
                        );

      As you can see, the syntax to create objects is similar regardless of the
      object type.

      4.1. In the example below, we create database `sql_class`.
    ********************************************************************** */

CREATE DATABASE sql_class;
```

```
359
360
361 /* ************************************************************************
362     4.2. We then create schema `ace`, which must be called to be used when
363          creating tables or other objects.
364
365          There is no need to call the name of the schema when using the SQL
366          Server default schema `dbo` (database owner) -- not used in this
367          example.
368  ************************************************************************ */
369
370 CREATE SCHEMA ace;
371
372
373 /* ************************************************************************
374     4.3. After creating the database (and the schema if needed), we can create
375          the table.
376
377                     CREATE TABLE table_name
378                     (
379                       field1 data type [null|not null] [unique] [primary key],
380                       field2 data type [null|not null],
381                       ...
382                     )
383  ************************************************************************ */
384
385 CREATE TABLE ace.students (                    -- 1. rule of thumb: table
386                                                --    names in plural
387   student_id INT NULL,                         -- 2. declared as INT;  can
388                                                --    accept NULL (can have no
389                                                --    value)
390   student_fname VARCHAR(50) NULL,              -- 3. declared as VARCHAR(50);
391                                                --    can accept NULL (can have
392                                                --    no value)
393   student_lname VARCHAR(50) NULL,              -- 4. declared as VARCHAR(50);
394                                                --    can accept NULL (can have
395                                                --    no value)
396   student_phone VARCHAR(15) NULL,              -- 5. declared as VARCHAR(50);
397                                                --    can accept NULL (can have
398                                                --    no value)
399   student_dob DATE NULL,                       -- 6. declared as DATE
400                                                --
401                                                --    DATETIME 10/12/2019 13:51
402                                                --    DATE     10/12/2019
403                                                --    TIME     13:51
404                                                --
405                                                --    can accept NULL (can have
406                                                --    no value)
407   record_date DATE NULL                        -- 5. declared as DATE;  when
408                                                --    record was created;  can
409                                                --    accept NULL (can have no
410                                                --    value)
```

```
411   );
412
413
414  /* ********************************************************************
415      4.4. After creating table `students` in schema `ace`, we insert values for
416           each column in the same order as the structure that we indicated in
417           #4.3.
418
419           If we do not have a value for a specific field, we can push an empty
420           string or NULL.
421   ******************************************************************** */
422
423  INSERT INTO ace.students
424  VALUES (
425    1,
426    'Joe',
427    'Smith',
428    '555-123-4567',
429    '1980/05/01',
430    GETDATE()                                    -- 1. built-in function to
431                                                 --    retrieve system DATETIME
432    ),
433    (
434    2,
435    'Mary',
436    'Jones',
437    '212-555-1000',
438    '1983/05/16',
439    GETDATE()
440    ),
441    (
442    3,
443    'Peter',
444    'Johnson',
445    NULL,                                        -- 2. inserting empty strings
446                                                 --    (``) or NULL since we
447                                                 --    have no values for fields
448                                                 --    to insert same number of
449                                                 --    values as columns
450    '06/01/1980',
451    GETDATE()
452    );
453
454
455  /* ********************************************************************
456      4.5. In the example below, we insert only three (3) values.
457
458           We call the the three (3) corresponding columns to indicate which
459           value goes where.
460
461           We do not need to call columns in order as long order as long as
462           values are pushed in the same order (value 1 in field 1, value 2 in
```

```
463           field 2, value 3 in field 3 and value 7 in field 7).
464   ********************************************************************** */
465
466   INSERT INTO ace.students (
467     student_id,                                 -- 1. inserting values to only
468     student_fname,                              --    four (4) columns;
469     student_lname,                              --    indicating which four (4)
470     record_date                                 --    columns
471     )
472   VALUES (
473     4,                                          -- 2. values to be inserted in
474     'Smith',                                    --    columns `student_id`,
475     'Tom',                                      --    `student_fname`,
476     GETDATE()                                   --    `student_lname` and
477     );                                          --    `record_date` receiving
478                                                 --    value from `GETDATE()`
479
480
481   /* **********************************************************************
482       4.6. In the example below, we insert row 6 before 5.
483
484           The values in `student_id` (the row identifier) are unique, but they
485           do not need to be in order.
486
487           If you need to insert values in `student_id` automatically in
488           incremental order, you would need to use `IDENTITY(1,1)` as part of
489           the table structure.  The first integer indicates that the first value
490           as one.  The second integer indicates that the value is incremented by
491           one.  Refer to https://www.w3schools.com/sql/sql_autoincrement.asp for
492           more information.
493
494
495                       CREATE TABLE ace.students (
496                         student_id INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
497                         student_fname VARCHAR(50) NULL,
498                         student_lname VARCHAR(50) NULL,
499                         student_phone VARCHAR(15) NULL,
500                         student_dob DATE NULL,
501                         record_date DATE NULL
502                         );
503   ********************************************************************** */
504
505   INSERT INTO ace.students
506   VALUES (
507     6,
508     'John',
509     'Scott',
510     '',                                         -- 1. inserting empty strings
511     '',                                         --    (``) or NULL since we
512                                                 --    have no values for fields
513                                                 --    to insert same number of
514                                                 --    values as columns
```

```
515    GETDATE()                                    -- 2. built-in function to
516                                                 --    retrieve system DATETIME
517    ),
518    (
519    5,
520    'Mary Ann',
521    'Saunders',
522    '',                                          -- 3. inserting empty strings
523    '',                                          --    (``) or NULL since we
524                                                 --    have no values for fields
525                                                 --    to insert same number of
526                                                 --    values as columns
527    GETDATE()                                    -- 4. built-in function to
528                                                 --    retrieve system DATETIME
529    );
530
531
532  /* ***************************************************************************
533   5. We can also delete/destroy data objects.
534
535      For the time being, we will work with tables
536      (https://techonthenet.com/sql_server/tables/drop_table.php).
537
538      Once an object is deleted, there is no way to rescue the data (ROLLBACK)
539      unless first creating a SAVEPOINT
540      (https://technet.microsoft.com/en-us/library/ms178157.aspx).
541
542      5.1. In the example below, we destroy (`DROP`) table `ace.students`
543           understanding that, once we do, we cannot recover the structure or the
544           data.
545   *************************************************************************** */
546
547  DROP TABLE ace.students;
548
549
550  /* ***************************************************************************
551      5.2. In the case of tables, we can destroy (`TRUNCATE`) the data in the
552           table without affecting the structure of the table understanding that,
553           once we do, we cannot recover the data.
554   *************************************************************************** */
555
556  TRUNCATE TABLE ace.students;
557
558
559  /* ***************************************************************************
560   6. We can also modify (`ALTER`) data objects.  We will start modifying tables
561      (https://techonthenet.com/sql_server/tables/alter_table.php) since you
562      might do this more often.
563
564      6.1. ADD        to add a column to a table
565
566      6.2. DROP       to delete a column to a table
```

```
567
568     6.3. ALTER      to change the data type or size of a column
569   ********************************************************************** */
570
571  ALTER TABLE ace.students                          -- 1. adding new column
572  ADD Email VARCHAR(100);                           --    `Email`;  no need to
573                                                    --    specify that you are
574                                                    --    adding a column
575
576  ALTER TABLE ace.students                          -- 2. dropping (deleting)
577  DROP COLUMN Email;                                --    column `Email` as there
578                                                    --    is no SQL statement to
579                                                    --    rename data objects;
580                                                    --    must specify that you are
581                                                    --    dropping a column
582
583  ALTER TABLE ace.students                          -- 3. adding new (replacement)
584  ADD student_email VARCHAR(100);                   --    column `student_email`;
585                                                    --    no need to specify that
586                                                    --    you are adding a column
587
588  ALTER TABLE ace.students                          -- 4. altering column with new
589  ALTER COLUMN student_email VARCHAR(50) NULL;      --    data type VARCHAR(50)
590                                                    --    from VARCHAR(100) and
591                                                    --    `NOT NULL`;  must specify
592                                                    --    that you are altering a
593                                                    --    column
594
595  ALTER TABLE ace.students                          -- 5. altering column as
596  ALTER COLUMN student_id INT NOT NULL;             --    `NOT NULL`;  must specify
597                                                    --    that you are altering a
598                                                    --    column
599
600  ALTER TABLE ace.students                          -- 6. altering column with new
601  ALTER COLUMN record_date DATETIME NOT NULL;       --    data type DATETIME
602                                                    --    from DATE and `NOT NULL`;
603                                                    --    must specify that you are
604                                                    --    altering a column
605
606  ALTER TABLE ace.students                          -- 7. altering column with new
607  ALTER COLUMN student_fname VARCHAR(25) NOT NULL;--    data type VARCHAR(25)
608                                                    --    from VARCHAR(50) and
609                                                    --    `NOT NULL`;  must specify
610                                                    --    that you are altering a
611                                                    --    column
612
613  ALTER TABLE ace.students                          -- 8. altering column with new
614  ALTER COLUMN student_fname VARCHAR(25) NOT NULL;--    data type VARCHAR(25)
615                                                    --    from VARCHAR(50) and
616                                                    --    `NOT NULL`;  must specify
617                                                    --    that you are altering a
618                                                    --    column
```

```sql

ALTER TABLE ace.students                          -- 9. altering column with new
ALTER COLUMN student_id VARCHAR(5);               --    data type VARCHAR(5) from
                                                  --    INT;  no error during
                                                  --    conversion;  must specify
                                                  --    that you are altering a
                                                  --    column

ALTER TABLE ace.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 you are
                                                  --     altering a column

ALTER TABLE ace.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)


/* ***************************************************************************
 7. We can use `UPDATE` to write new values into an existing row.

    7.1. In the example below, we UPDATE the value of column `student_phone`
         passing value `No Number` where there is no value (`IS NULL`) or there
         is an empty space (` `)
 *************************************************************************** */

UPDATE ace.students
SET student_phone = 'No Number'
WHERE student_phone IS NULL
  OR student_phone = '';


/* ***************************************************************************
    7.2. In the example below, we UPDATE 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@example.com` for `student_fname` with value of
         `John` and `student_lname` with value of `Smith`.
 *************************************************************************** */

UPDATE ace.students
SET student_email = LOWER(CONCAT (
      student_fname,
      '.',
      student_lname,
      '@example.com'
      ));
```

```
671
672
673 /* ********************************************************************************
674     7.3. In the example below, we UPDATE column `record_date` where the field
675          is NULL or has an empty space (``) with value from `GETDATE()`.
676  ******************************************************************************** */
677
678 UPDATE ace.students
679 SET record_date = GETDATE()
680 WHERE record_date IS NULL
681   OR record_date = '';
682
683
684 /* ********************************************************************************
685     7.4. In the example below, we can UPDATE `student_dob` to `1980/01/23`
686          where `student_id` is `1`.
687  ******************************************************************************** */
688
689 UPDATE ace.students
690 SET student_dob = '1980/01/23'
691 WHERE student_id = 1;
692
693
694 /* ********************************************************************************
695  8. In the example below, we use `TRUNCATE` to delete all data from table
696     `ace.students` without dropping (destroying) the table.
697  ******************************************************************************** */
698
699 TRUNCATE TABLE ace.students;
700
701
702 /* ********************************************************************************
703  9. Since there is no copy statements in SQL, we are limited to the vendor
704     extensions (vendor-specific SQL).
705
706     When working with some vendors like Oracle, we can CREATE a new table from
707     a query on another table.
708
709                     CREATE TABLE new_table
710                     AS
711                       (
712                         SELECT field1, field2 ...
713                         FROM old_table
714                       )
715
716     In SQL Server, we use `INTO`.
717
718                     SELECT field1, field2 ...
719                       INTO new_table
720                     FROM old_table
721
722     In the example below, we push the output of the query to retrieve all
```

```
723     values from table `ace.students` into `ace.students2`.
724
725                     SELECT field1, field2 ...
726                       INTO new_table
727                     FROM old_table1
728                     INNER|LEFT|RIGHT JOIN old_table2
729                       ON old_table1.common_field1 = old_table2.common_field1...
730
731     A view (http://searchsqlserver.techtarget.com/definition/view) is a better
732     option, which we will cover on the next class.
733  ************************************************************************ */
734
735  SELECT *                                        -- 1. selecting all values
736                                                  --    from `ace.students`
737  INTO ace.students2                              -- 2. creating the new table
738                                                  --    `ace.students2`
739  FROM ace.students;                              -- 3. from table `ace.students`
740
741
742  /* ************************************************************************
743   10. LAB #5
744       Write a query
745       10.1. to call all columns and values shared by tables `AP1.ContactUpdates`
746             and `AP1.Vendors` (`INNER JOIN`),
747       10.2. retrieving only rows with `AP1.Vendors.VendorState` with values of
748             `NY`, `NJ` and `CA`
749       10.3. using `CASE` to replace `NY` to `New York`, `NJ` to `New Jersey`,
750             `CA` to `California` and any other value to `Other`
751       10.4. ordered first by `AP1.Vendors.VendorState` and then by
752             `AP1.Vendors.VendorID`.
753   ************************************************************************ */
754
755  SELECT AP1.ContactUpdates.VendorID,
756    AP1.ContactUpdates.LastName,
757    AP1.ContactUpdates.FirstName,
758    -- AP1.Vendors.VendorID AS Expr1,            -- 1. duplicate column name
759                                                  --    commented out
760    AP1.Vendors.VendorName,
761    AP1.Vendors.VendorAddress1,
762    AP1.Vendors.VendorAddress2,
763    AP1.Vendors.VendorCity,
764    CASE                                          -- 2. beginning of logic
765      WHEN AP1.Vendors.VendorState = 'NY'         --    2.1. checking for value
766        THEN 'New York'                           --         `NY` and return
767                                                  --         value `New York`
768      WHEN AP1.Vendors.VendorState = 'NJ'         --    2.2. checking for value
769        THEN 'New Jersey'                         --         `NY` and return
770                                                  --         value `New Jersey`
771      WHEN AP1.Vendors.VendorState = 'CA'         --    2.3. checking for value
772        THEN 'California'                         --         `NY` and return
773                                                  --         value `California`
774      ELSE 'Other'                                --    2.4. checking for other
```

```
775                                                 --         values and return
776                                                 --         value `Other`
777       END AS VendorState,
778     AP1.Vendors.VendorZipCode,
779     AP1.Vendors.VendorPhone,
780     AP1.Vendors.VendorContactLName,
781     AP1.Vendors.VendorContactFName,
782     AP1.Vendors.DefaultTermsID,
783     AP1.Vendors.DefaultAccountNo
784  FROM AP1.ContactUpdates
785  INNER JOIN AP1.Vendors
786     ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
787  WHERE AP1.Vendors.VendorState IN (              -- 3. indicating what values we
788       'NY',                                     --    query to return
789       'NJ',
790       'CA'
791       );
792
793  /* ***************************************************************************
794   https://folvera.commons.gc.cuny.edu/?p=1034
795   ************************************************************************** */
```