```
/* ************************************************************************
          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 #6 (2023/04/19): 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. LAB #4
    Write a query without duplicate rows (`SELECT DISTINCT`)
    1.01. to get all fields from `AP1.Invoices` and `AP1.InvoiceLineItems` to
          retrieve shared data (`INNER JOIN`) removing all duplicate columns
          (`AP1.Invoices.InvoiceID` and `AP1.InvoiceLineItems.InvoiceID`),
    1.02. to format dates as `MMM d, yyyy` (first three letters of the month,
          the day without leading zeros and the full year)
    1.03. and to format money (`c`) as `en-us` (`$`).
 ************************************************************************ */

SELECT DISTINCT
  AP1.Invoices.InvoiceID,
  AP1.Invoices.InvoiceNumber,
  FORMAT(AP1.Invoices.InvoiceDate,              -- 1. formatting column as
    'MM/dd/yyyy', 'en-us')                      --    `MM/dd/yyyy` (date) with
                                                --    culture `en-us` as
  AS InvoiceDate,                               --    `InvoiceDate`
  FORMAT(AP1.Invoices.InvoiceTotal,             -- 2. formatting column as
    'MM/dd/yyyy', 'en-us')                      --    `MM/dd/yyyy` (date) with
                                                --    culture `en-us` as
  AS InvoiceTotal,                              --    `InvoiceTotal`
  (
    SELECT                                      -- 3. embedded query calling
      FORMAT(AVG(AP1.Invoices.InvoiceTotal),    --    `AVG(InvoiceTotal)`
        'c', 'en-us')                           --    formatted as `c`
                                                --    (currency) with culture
                                                --    `en-us`
    FROM AP1.Invoices                           --    from all values in table
                                                --    `AP1.Invoices` as
  ) AS AvgInvoiceTotal,                         --    `AvgInvoiceTotal`
  FORMAT(AP1.Invoices.PaymentTotal,             -- 4. formatting column as `c`
    'c', 'en-us')                               --    (currency) with culture
  AS PaymentTotal,                              --    `en-us` as `PaymentTotal`
  FORMAT(AP1.Invoices.CreditTotal,              -- 5. formatting column as `c`
    'c', 'en-us')                               --    (currency) with culture
  AS CreditTotal,                               --    `en-us` as `CreditTotal`
  FORMAT(AP1.Invoices.InvoiceDueDate,           -- 6. formatting column as
    'MM/dd/yyyy', 'en-us')                      --    `MM/dd/yyyy` (date) with
```

```
53                                                  --    culture `en-us` as
54    AS InvoiceDueDate,                            --    `InvoiceDueDate`
55    FORMAT(AP1.Invoices.PaymentDate,              -- 7. formatting column as
56      'MM/dd/yyyy', 'en-us')                      --    `MM/dd/yyyy` (date) with
57                                                  --    culture `en-us` as
58    AS PaymentDate,                               --    `PaymentDate`
59    AP1.InvoiceLineItems.InvoiceSequence,
60    AP1.InvoiceLineItems.AccountNo,
61    FORMAT(AP1.InvoiceLineItems.InvoiceLineItemAmount,
62      'c', 'en-us')                               -- 8. formatting column as `c`
63                                                  --    (currency) with culture
64                                                  --    `en-us` as
65    AS InvoiceLineItemAmount,                     --    `InvoiceLineItemAmount`
66    AP1.InvoiceLineItems.InvoiceLineItemDescription
67  FROM AP1.Invoices                               -- 9. from `AP1.Invoices` using
68  INNER JOIN AP1.InvoiceLineItems                 --    `INNER JOIN` to connect
69                                                  --    to `AP1.InvoiceLineItems`
70                                                  --    to get all shared values
71    ON AP1.Invoices.InvoiceID = AP1.InvoiceLineItems.InvoiceID
72                                                  --    in `AP1.InvoiceLineItems`
73                                                  --    and `AP1.Invoices`
74
75
76  /* ****************************************************************************
77   2. As a review, we understand that the most common joins we will use are the
78      following.
79
80                      +-----------------+
81                      | LEFT    +-------+---------+
82                      | JOIN    | INNER |         |
83                      |         | JOIN  |  RIGHT  |
84                      +---------+-------+  JOIN   |
85                                +-----------------+
86
87      2.01. `INNER JOIN` calls the data shared in both tables.  The data must be
88            present in both table.  All other data is ignored.
89
90      2.02. `LEFT JOIN` calls in the left table (called first) plus any related
91            data found in the right table (second table).  This means that the
92            right table does not need to have corresponding data.  In other
93            words, if the right table does not have related data, nothing is
94            returned (NULLs at the beginning of the dataset output).
95
96            As such, we can ask for all data in `AP1.Vendors` (main), not
97            necessarily from `AP1.Invoices` (secondary).  In this example, we are
98            interested in all `AP1.Vendors` regardless of possible corresponding
99            data in `AP1.Invoices`.  In other words, some vendors might not have
100           sales.
101   **************************************************************************** */
102
103 SELECT *
104 FROM AP1.Vendors                                -- 1. main table called first
```

```
105                                                 --    (left)
106  LEFT JOIN AP1.Invoices                         -- 2. secondary table called
107                                                 --    second (right), always in
108                                                 --    groups of two (2) tables
109    ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;
110
111
112  /* ***************************************************************************
113      2.03. `RIGHT JOIN` calls in the right table (called second) plus any
114            related data found in the left table (first table).  This means that
115            the left table does not need to have corresponding data.  In other
116            words, if the left table does not have related data, nothing is
117            returned (NULLs at the end of the dataset output).
118
119            As such, we can ask for all data in `AP1.Invoices` (main), not
120            necessarily from `AP1.Vendors` (secondary).  In this example, we are
121            interested in all `AP1.Invoices` regardless of possible corresponding
122            data in `AP1.Vendors`.  In other words, some invoices might not have
123            vendor data.
124   *************************************************************************** */
125
126  SELECT *
127  FROM AP1.Vendors                               -- 1. secondary table called
128                                                 --    first (left)
129  RIGHT JOIN AP1.Invoices                        -- 2. main table called second
130                                                 --    (right), always in groups
131                                                 --    of two (2) tables
132    ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;
133
134
135  /* ***************************************************************************
136      2.04. On a personal note, `RIGHT JOIN` is a disorganized way to write code.
137            The example above could easily be called using `LEFT JOIN` ordering
138            the tables more appropriately.  Note that the order of `VendorID`
139            coming from `AP1.Invoices` and `AP1.Vendors.VendorID` makes no
140            difference.
141   *************************************************************************** */
142
143  SELECT *
144  FROM AP1.Invoices                              -- 1. main table called first
145                                                 --    (left)
146  LEFT JOIN AP1.Vendors                          -- 2. secondary table called
147                                                 --    second (right), always in
148                                                 --    groups of two (2) tables
149    ON AP1.Invoices.VendorID = AP1.Vendors.VendorID;
150
151
152  /* ***************************************************************************
153   3. Now that we understand most common data types, we can start creating data
154      objects (DATABASE, TABLE, etc.) and populating tables with data.
155
156      3.01. Note that no two objects of the same hierarchy can share the same
```

```
157           name, for example a TABLE and a VIEW.
158
159     3.02. The following is a quick view of database hierarchy.
160
161                     SERVER:  ``A server is a computer program that provides a
162                       |   service to another computer programs (and its user).
163                       |   In a data center, the physical computer that a server
164                       |   program runs in is also frequently referred to as a
165                       |   server.  That machine may be a dedicated server or
166                       |   it may be used for other purposes as well.``
167                       |   https://whatis.techtarget.com/definition/server
168                       |
169                       +- DATABASE:  ``A database is a collection of information
170                            |   that is organized so that it can be easily
171                            |   accessed, managed and updated.
172                            |   Data is organized into rows, columns and tables,
173                            |   and it is indexed to make it easier to find
174                            |   relevant information.  Data gets updated,
175                            |   expanded and deleted as new information is added.
176                            |   Databases process workloads to create and update
177                            |   themselves, querying the data they contain and
178                            |   running applications against it.``
179                            |   https://searchsqlserver.techtarget.com/definition/ ↵
                        database
180                            |
181                            +- SCHEMA:  ``1) In computer programming, a schema
182                                 |   (pronounced SKEE-mah) is the organization or
183                                 |   structure for a database.  The activity of
184                                 |   data modeling leads to a schema.  (The plural
185                                 |   form is schemata.  The term is from a Greek
186                                 |   word for ``form`` or ``figure.``  Another
187                                 |   word from the same source is ``schematic.``)
188                                 |   The term is used in discussing both
189                                 |   relational databases and object-oriented
190                                 |   databases.  The term sometimes seems to refer
191                                 |   to a visualization of a structure and
192                                 |   sometimes to a formal text-oriented
193                                 |   description.
194                                 |   Two common types of database schemata are the
195                                 |   star schema and the snowflake schema.
196                                 |   2) In another usage derived from mathematics,
197                                 |   a schema is a formal expression of an
198                                 |   inference rule for artificial intelligence
199                                 |   (AI) computing.  The expression is a
200                                 |   generalized axiom in which specific values or
201                                 |   cases are substituted for each symbol in the
202                                 |   axiom to derive a specific inference.``
203                                 |   https://searchsqlserver.techtarget.com/      ↵
                        definition/schema
204                                 |
205                                 +- TABLES:  ``In computer programming, a table is
206                                 |   |   a data structure used to organize
```

```
207                                   |   |   information, just as it is on paper.``
208                                   |   |   https://whatis.techtarget.com/definition/ ↵
                          table
209                                   |   |
210                                   |   +- COLUMNS (FIELDS):  ``A field is an area in
211                                   |   |   a fixed or known location in a unit of
212                                   |   |   data such as a record, message header, or
213                                   |   |   computer instruction that has a purpose
214                                   |   |   and usually a fixed size.  In some
215                                   |   |   contexts, a field can be subdivided  into
216                                   |   |   smaller fields.``
217                                   |   |   https://searchoracle.techtarget.com/       ↵
                          definition/field
218                                   |   |
219                                   |   +- PRIMARY KEY (PRIMARY KEYWORD):  ``A primary
220                                   |   |   key, also called a primary keyword, is a
221                                   |   |   key in a relational database that is
222                                   |   |   unique for each record.  It is a unique
223                                   |   |   identifier, such as a driver license
224                                   |   |   number, telephone number (including area
225                                   |   |   code), or vehicle identification number
226                                   |   |   (VIN).  A relational database must always
227                                   |   |   have one and only one primary key.
228                                   |   |   Primary keys typically appear as columns
229                                   |   |   in relational database tables.``
230                                   |   |   https://searchsqlserver.techtarget.com/    ↵
                          definition/primary-key
231                                   |   |
232                                   |   +- FOREIGN KEY:  ``A foreign key is a column
233                                   |       or columns of data in one table that
234                                   |       connects to the primary key data in the
235                                   |       original table.  To ensure the links
236                                   |       between foreign key and primary key
237                                   |       tables aren't broken, foreign key
238                                   |       constraints can be created to prevent
239                                   |       actions that would damage the links
240                                   |       between tables and prevent erroneous data
241                                   |       from being added to the foreign key
242                                   |       column.``
243                                   |       https://searchoracle.techtarget.com/       ↵
                          definition/foreign-key
244                                   |
245                                   +- VIEWS:  ``In a database management system, a
246                                   |     view is a way of portraying information in
247                                   |     the database.``
248                                   |     https://whatis.techtarget.com/search/query
249                                   |
250                                   +- STRUCTURED (MODULAR) PROGRAMMING:
251                                       |   ``Structured programming (sometimes known
252                                       |   as modular programming) is a subset of
253                                       |   procedural programming that enforces a
254                                       |   logical structure on the program being
```

```
255                                   |   written to make it more efficient and
256                                   |   easier to understand and modify.  Certain
257                                   |   languages such as Ada, Pascal, and dBASE
258                                   |   are designed with features that encourage
259                                   |   or enforce a logical program structure.``
260                                   |   https://
                        searchsoftwarequality.techtarget.com/definition/structured-
                        programming-modular-programming
261                                   |
262                                   +- FUNCTIONS:  ``In information technology,
263                                   |   the term function (pronounced FUHNK-shun)
264                                   |   has a number of  meanings.  It's taken
265                                   |   from the Latin ``functio`` -- to perform.
266                                   |   1) In its most general use, a function is
267                                   |   what a given entity does in being what it
268                                   |   is.
269                                   |   2) In C language and other programming, a
270                                   |   function is a named procedure that
271                                   |   performs a distinct service.  The
272                                   |   language statement that requests the
273                                   |   function is called a function call.
274                                   |   Programming languages usually come with a
275                                   |   compiler and a set of ``canned``
276                                   |   functions that a programmer can specify
277                                   |   by writing language statements.  These
278                                   |   provided functions are sometimes referred
279                                   |   to as library routines.  Some functions
280                                   |   are self-sufficient and can return
281                                   |   results to the requesting program without
282                                   |   help.  Other functions need to make
283                                   |   requests of the operating system in order
284                                   |   to perform their work.``
285                                   |   https://whatis.techtarget.com/definition/
                        function
286                                   |
287                                   +- PROCEDURES:  ``A stored procedure is a set
288                                       of Structured Query Language (SQL)
289                                       statements with an assigned name, which
290                                       are stored in a relational database
291                                       management system as a group, so it can
292                                       be reused and shared by multiple
293                                       programs.``
294                                       https://searchoracle.techtarget.com/
                        definition/stored-procedure
295
296   4. Now that you have a better understanding of data types, we can start
297      creating objects.
298
299                       CREATE  obj_type   obj_name [some_code]
300
301                       CREATE  DATABASE   db_name;
302
```

```
303                     CREATE  SCHEMA      schema_name;
304
305                     CREATE  TABLE       schema_name.table_name
306                       (
307                         field_1 datatype_1 [attributes],
308                         field_2 datatype_2 [attributes],
309                         field_3 datatype_3 [attributes],
310                         ...
311                       );
312
313                     CREATE  VIEW        schema_name.view_table
314                     AS
315                       (
316                         SELECT fields...
317                         FROM table(s)
318                       );
319
320     As you can see, the syntax to create objects is similar regardless of the
321     object type.
322
323     4.01. In the example below, we create database `labs`.
324  ************************************************************************ */
325
326  CREATE DATABASE labs;
327
328
329  /* ************************************************************************
330     4.02. We then create schema `ace`, which must be called to be used when
331           creating tables or other objects.
332
333           There is no need to call the name of the schema when using the SQL
334           Server default schema `dbo` (database owner) -- not used in this
335           example.
336  ************************************************************************ */
337
338  CREATE SCHEMA ace;
339
340
341  /* ************************************************************************
342     4.03. After creating the database (and the schema if needed), we can create
343           the table.
344
345                     CREATE TABLE table_name
346                     (
347                       field1 data type [null|not null] [unique] [primary key],
348                       field2 data type [null|not null],
349                       ...
350                     )
351  ************************************************************************ */
352
353  CREATE TABLE ace.students (                    -- 1. rule of thumb: table
354                                                 --    names in plural
```

```
355    student_id INT NULL,                          -- 2. declared as INT;  can
356                                                  --    accept NULL (can have no
357                                                  --    value)
358    student_fname VARCHAR(50) NULL,               -- 3. declared as VARCHAR(50);
359                                                  --    can accept NULL (can have
360                                                  --    no value)
361    student_lname VARCHAR(50) NULL,               -- 4. declared as VARCHAR(50);
362                                                  --    can accept NULL (can have
363                                                  --    no value)
364    student_phone VARCHAR(15) NULL,               -- 5. declared as VARCHAR(50);
365                                                  --    can accept NULL (can have
366                                                  --    no value)
367    student_dob DATE NULL,                        -- 6. declared as DATE
368                                                  --
369                                                  --    DATETIME 04/20/2023 20:51
370                                                  --    DATE     04/20/2023
371                                                  --    TIME     20:51
372                                                  --
373                                                  --    can accept NULL (can have
374                                                  --    no value)
375    record_date DATE NULL                         -- 5. declared as DATE;  when
376                                                  --    record was created;  can
377                                                  --    accept NULL (can have no
378                                                  --    value)
379    );
380
381
382  /* ***************************************************************************
383      4.04. After creating table `students` in schema `ace`, we insert values for
384            each column in the same order as the structure that we indicated in
385            #4.03.
386
387            If we do not have a value for a specific field, we can push an empty
388            string or NULL.
389   *************************************************************************** */
390
391  INSERT INTO ace.students
392  VALUES (
393    1,
394    'Joe',
395    'Smith',
396    '555-123-4567',
397    '1980/05/01',
398    GETDATE()                                     -- 1. built-in function to
399                                                  --    retrieve system DATETIME
400    ),
401    (
402    2,
403    'Mary',
404    'Jones',
405    '212-555-1000',
406    '1983/05/16',
```

```
407    GETDATE()
408    ),
409    (
410    3,
411    'Peter',
412    'Johnson',
413    NULL,                                    -- 2. inserting empty strings
414                                             --    (``) or NULL since we
415                                             --    have no values for fields
416                                             --    to insert same number of
417                                             --    values as columns
418    '06/01/1980',
419    GETDATE()
420    );
421
422
423  /* ********************************************************************
424      4.05. In the example below, we insert only three (3) values.
425
426            We call the the three (3) corresponding columns to indicate which
427            value goes where.
428
429            We do not need to call columns in order as long order as long as
430            values are pushed in the same order (value 1 in field 1, value 2 in
431            field 2, value 3 in field 3 and value 7 in field 7).
432   ******************************************************************** */
433
434  INSERT INTO ace.students (
435    student_id,                              -- 1. inserting values to only
436    student_fname,                           --    four (4) columns;
437    student_lname,                           --    indicating which four (4)
438    record_date                              --    columns
439    )
440  VALUES (
441    4,                                       -- 2. values to be inserted in
442    'Smith',                                 --    columns `student_id`,
443    'Tom',                                   --    `student_fname`,
444    GETDATE()                                --    `student_lname` and
445    );                                       --    `record_date` receiving
446                                             --    value from `GETDATE()`
447
448
449  /* ********************************************************************
450      4.06. In the example below, we insert row 6 before 5.
451
452            The values in `student_id` (the row identifier) are unique, but they
453            do not need to be in order.
454
455             If you need to insert values in `student_id` automatically in
456             incremental order, you would need to use `IDENTITY(1,1)` as part of
457             the table structure.  The first integer indicates that the first
458             value as one.  The second integer indicates that the value is
```

```
459             incremented by one.  Refer to
460             https://www.w3schools.com/sql/sql_autoincrement.asp for more
461             information.
462
463
464                     CREATE TABLE ace.students (
465                       student_id INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
466                       student_fname VARCHAR(50) NULL,
467                       student_lname VARCHAR(50) NULL,
468                       student_phone VARCHAR(15) NULL,
469                       student_dob DATE NULL,
470                       record_date DATE NULL
471                       );
472   ********************************************************************** */
473
474  INSERT INTO ace.students
475  VALUES (
476    6,
477    'John',
478    'Scott',
479    '',                                          -- 1. inserting empty strings
480    '',                                          --    (``) or NULL since we
481                                                 --    have no values for fields
482                                                 --    to insert same number of
483                                                 --    values as columns
484    GETDATE()                                    -- 2. built-in function to
485                                                 --    retrieve system DATETIME
486    ),
487    (
488    5,
489    'Mary Ann',
490    'Saunders',
491    '',                                          -- 3. inserting empty strings
492    '',                                          --    (``) or NULL since we
493                                                 --    have no values for fields
494                                                 --    to insert same number of
495                                                 --    values as columns
496    GETDATE()                                    -- 4. built-in function to
497                                                 --    retrieve system DATETIME
498    );
499
500
501  /* ************************************************************************
502   5. We can also delete/destroy data objects.
503
504      For the time being, we will work with tables
505      (https://techonthenet.com/sql_server/tables/drop_table.php).
506
507      Once an object is deleted, there is no way to rescue the data (ROLLBACK)
508      unless first creating a SAVEPOINT
509      (https://technet.microsoft.com/en-us/library/ms178157.aspx).
510
```

```
511      5.01. In the example below, we destroy (`DROP`) table `ace.students`
512            understanding that, once we do, we cannot recover the structure or
513            the data.
514   ********************************************************************** */
515
516  DROP TABLE ace.students;
517
518
519  /* **********************************************************************
520      5.2. In the case of tables, we can destroy (`TRUNCATE`) the data in the
521           table without affecting the structure of the table understanding that,
522           once we do, we cannot recover the data.
523   ********************************************************************** */
524
525  TRUNCATE TABLE ace.students;
526
527
528  /* **********************************************************************
529   6. We can also modify (`ALTER`) data objects.  We will start modifying tables
530      (https://techonthenet.com/sql_server/tables/alter_table.php) since you
531      might do this more often.
532
533      ADD             to add a column to a table
534
535      DROP            to delete a column to a table
536
537      ALTER           to change the data type or size of a column
538   ********************************************************************** */
539
540  ALTER TABLE ace.students                        -- 1. adding new column
541  ADD Email VARCHAR(100);                         --    `Email`;  no need to
542                                                  --    specify that you are
543                                                  --    adding a column
544
545  ALTER TABLE ace.students                        -- 2. dropping (deleting)
546  DROP COLUMN Email;                              --    column `Email` as there
547                                                  --    is no SQL statement to
548                                                  --    rename data objects;
549                                                  --    must specify that you are
550                                                  --    dropping a column
551
552  ALTER TABLE ace.students                        -- 3. adding new (replacement)
553  ADD student_email VARCHAR(100);                 --    column `student_email`;
554                                                  --    no need to specify that
555                                                  --    you are adding a column
556
557  ALTER TABLE ace.students                        -- 4. altering column with new
558  ALTER COLUMN student_email VARCHAR(50) NULL;    --    data type VARCHAR(50)
559                                                  --    from VARCHAR(100) and
560                                                  --    `NOT NULL`;  must specify
561                                                  --    that you are altering a
562                                                  --    column
```

```
563
564 ALTER TABLE ace.students                          -- 5. altering column as
565 ALTER COLUMN student_id INT NOT NULL;            --    `NOT NULL`;  must specify
566                                                  --    that you are altering a
567                                                  --    column
568
569 ALTER TABLE ace.students                          -- 6. altering column with new
570 ALTER COLUMN record_date DATETIME NOT NULL;      --    data type DATETIME
571                                                  --    from DATE and `NOT NULL`;
572                                                  --    must specify that you are
573                                                  --    altering a column
574
575 ALTER TABLE ace.students                          -- 7. altering column with new
576 ALTER COLUMN student_fname VARCHAR(25) NOT NULL;--    data type VARCHAR(25)
577                                                  --    from VARCHAR(50) and
578                                                  --    `NOT NULL`;  must specify
579                                                  --    that you are altering a
580                                                  --    column
581
582 ALTER TABLE ace.students                          -- 8. altering column with new
583 ALTER COLUMN student_fname VARCHAR(25) NOT NULL;--    data type VARCHAR(25)
584                                                  --    from VARCHAR(50) and
585                                                  --    `NOT NULL`;  must specify
586                                                  --    that you are altering a
587                                                  --    column
588
589 ALTER TABLE ace.students                          -- 9. altering column with new
590 ALTER COLUMN student_id VARCHAR(5);              --    data type VARCHAR(5) from
591                                                  --    INT;  no error during
592                                                  --    conversion;  must specify
593                                                  --    that you are altering a
594                                                  --    column
595
596 ALTER TABLE ace.students                          -- 10. altering column back to
597 ALTER COLUMN student_id INT NOT NULL;            --     data type INT from
598                                                  --     VARCHAR(5);  no error
599                                                  --     during conversion;  must
600                                                  --     specify that you are
601                                                  --     altering a column
602
603 ALTER TABLE ace.students                          -- 11. trying to alter column
604 ALTER COLUMN student_fname FLOAT;                --     to data type FLOAT from
605                                                  --     VARCHAR(25);  conversion
606                                                  --     failure due to format
607                                                  --     incompatibility (letters
608                                                  --     to numbers)
609
610
611 /* ****************************************************************************
612  7. We can use `UPDATE` to write new values into an existing row.
613
614     7.01. In the example below, we UPDATE the value of column `student_phone`
```

```
615           passing value `No Number` where there is no value (`IS NULL`) or
616           there is an empty space (` `)
617    ********************************************************************** */
618
619  UPDATE ace.students
620  SET student_phone = 'No Number'
621  WHERE student_phone IS NULL
622    OR student_phone = '';
623
624
625  /* **********************************************************************
626      7.02. In the example below, we UPDATE the value of column `student_email`
627            passing the value of the concatenation of `student_fname` and
628            `student_lname` with a period (`.`) between the two columns -- for
629            example, `john.smith@example.foo` for `student_fname` with value of
630            `John` and `student_lname` with value of `Smith`.
631    ********************************************************************** */
632
633  UPDATE ace.students
634  SET student_email = LOWER(CONCAT (
635        student_fname,
636        '.',
637        student_lname,
638        '@example.foo'
639        ));
640
641
642  /* **********************************************************************
643      7.03. In the example below, we UPDATE column `record_date` where the field
644            is NULL or has an empty space (``) with value from `GETDATE()`.
645    ********************************************************************** */
646
647  UPDATE ace.students
648  SET record_date = GETDATE()
649  WHERE record_date IS NULL
650    OR record_date = '';
651
652
653  /* **********************************************************************
654      7.04. In the example below, we can UPDATE `student_dob` to `1980/01/23`
655            where `student_id` is `1`.
656    ********************************************************************** */
657
658  UPDATE ace.students
659  SET student_dob = '1980/01/23'
660  WHERE student_id = 1;
661
662
663  /* **********************************************************************
664    8. In the example below, we use `TRUNCATE` to delete all data from table
665       `ace.students` without dropping (destroying) the table.
666    ********************************************************************** */
```

```
667
668 TRUNCATE TABLE ace.students;
669
670
671 /* ************************************************************************
672  9. Since there is no copy statements in SQL, we are limited to the vendor
673     extensions (vendor-specific SQL).
674
675     When working with some vendors like Oracle, we can CREATE a new table from
676     a query on another table.
677
678                     CREATE TABLE new_table
679                     AS
680                       (
681                         SELECT field1, field2 ...
682                         FROM old_table
683                       )
684
685     In SQL Server, we use `INTO`.
686
687                     SELECT field1, field2 ...
688                       INTO new_table
689                     FROM old_table
690
691     In the example below, we push the output of the query to retrieve all
692     values from table `ace.students` into `ace.students2`.
693
694                     SELECT field1, field2 ...
695                       INTO new_table
696                     FROM old_table1
697                     INNER|LEFT|RIGHT JOIN old_table2
698                       ON old_table1.common_field1 = old_table2.common_field1...
699
700     A view (http://searchsqlserver.techtarget.com/definition/view) is a better
701     option, which we will cover on the next class.
702  ************************************************************************ */
703
704 SELECT *                                        -- 1. selecting all values
705                                                 --    from `ace.students`
706 INTO ace.students2                              -- 2. creating the new table
707                                                 --    `ace.students2`
708 FROM ace.students;                              -- 3. from table `ace.students`
709
710
711 /* ************************************************************************
712  https://folvera.commons.gc.cuny.edu/?p=1227
713  ************************************************************************ */
```