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

  SESSION #6 (2024/03/27): 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
                                                --    culture `en-us` as
  AS InvoiceDueDate,                            --    `InvoiceDueDate`
  FORMAT(AP1.Invoices.PaymentDate,              -- 7. formatting column as
    'MM/dd/yyyy', 'en-us')                      --    `MM/dd/yyyy` (date) with
                                                --    culture `en-us` as
  AS PaymentDate,                               --    `PaymentDate`
  AP1.InvoiceLineItems.InvoiceSequence,
  AP1.InvoiceLineItems.AccountNo,
  FORMAT(AP1.InvoiceLineItems.InvoiceLineItemAmount,
    'c', 'en-us')                               -- 8. formatting column as `c`
                                                --    (currency) with culture
                                                --    `en-us` as
  AS InvoiceLineItemAmount,                     --    `InvoiceLineItemAmount`
```

```
  AP1.InvoiceLineItems.InvoiceLineItemDescription
FROM AP1.Invoices                                   -- 9. from `AP1.Invoices` using
INNER JOIN AP1.InvoiceLineItems                     --    `INNER JOIN` to connect
                                                    --    to `AP1.InvoiceLineItems`
                                                    --    to get all shared values
  ON AP1.Invoices.InvoiceID = AP1.InvoiceLineItems.InvoiceID
                                                    --    in `AP1.InvoiceLineItems`
                                                    --    and `AP1.Invoices`


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

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

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

    2.02. `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).

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


/* ************************************************************************
    2.03. `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 table does not need to have corresponding data.  In other
          words, if the left table does not have related data, nothing is
          returned (NULLs at the end of the dataset output).

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

SELECT *
FROM AP1.Vendors                                    -- 1. secondary table called
                                                    --    first (left)
RIGHT JOIN AP1.Invoices                             -- 2. main table called second
                                                    --    (right), always in groups
```

```
                                                --    of two (2) tables
  ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;


/* ************************************************************************
    2.04. On a personal note, `RIGHT JOIN` is a disorganized way to write code.
          The example above could easily be called using `LEFT JOIN` ordering
          the tables more appropriately.  Note that the order of `VendorID`
          coming from `AP1.Invoices` and `AP1.Vendors.VendorID` makes no
          difference.
 ************************************************************************ */

SELECT *
FROM AP1.Invoices                               -- 1. main table called first
                                                --    (left)
LEFT JOIN AP1.Vendors                           -- 2. secondary table called
                                                --    second (right), always in
                                                --    groups of two (2) tables
  ON AP1.Invoices.VendorID = AP1.Vendors.VendorID;


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

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

    3.02. 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 center, the physical computer that a server
                      |   program runs in is also frequently referred to as a
                      |   server.  That machine may be a dedicated server or
                      |   it may be used for other purposes as well.``
                      |   https://whatis.techtarget.com/definition/server
                      |
                      +- DATABASE:  ``A database is a collection of information
                          |   that is organized so that it can be easily
                          |   accessed, managed and updated.
                          |   Data is organized into rows, columns and tables,
                          |   and it is indexed to make it easier to find
                          |   relevant information.  Data gets updated,
                          |   expanded and deleted as new information is added.
                          |   Databases process workloads to create and update
                          |   themselves, querying the data they contain and
                          |   running applications against it.``
                          |   https://searchsqlserver.techtarget.com/definition/database
                          |
                          +- SCHEMA:  ``1) In computer programming, a schema
                              |   (pronounced SKEE-mah) is the organization or
                              |   structure for a database.  The activity of
                              |   data modeling leads to a schema.  (The plural
                              |   form is schemata.  The term is from a Greek
                              |   word for ``form`` or ``figure.``  Another
                              |   word from the same source is ``schematic.``)
                              |   The term is used in discussing both
                              |   relational databases and object-oriented
                              |   databases.  The term sometimes seems to refer
                              |   to a visualization of a structure and
                              |   sometimes to a formal text-oriented
                              |   description.
                              |   Two common types of database schemata are the
                              |   star schema and the snowflake schema.
```

```
|   2) In another usage derived from mathematics,
|   a schema is a formal expression of an
|   inference rule for artificial intelligence
|   (AI) computing.  The expression is a
|   generalized axiom in which specific values or
|   cases are substituted for each symbol in the
|   axiom to derive a specific inference.``
|   https://searchsqlserver.techtarget.com/definition/schema
|
+- TABLES:  ``In computer programming, a table is
|   |   a data structure used to organize
|   |   information, just as it is on paper.``
|   |   https://whatis.techtarget.com/definition/table
|   |
|   +- COLUMNS (FIELDS):  ``A field is an area in
|   |   a fixed or known location in a unit of
|   |   data such as a record, message header, or
|   |   computer instruction that has a purpose
|   |   and usually a fixed size.  In some
|   |   contexts, a field can be subdivided  into
|   |   smaller fields.``
|   |   https://searchoracle.techtarget.com/definition/field
|   |
|   +- PRIMARY KEY (PRIMARY KEYWORD):  ``A primary
|   |   key, also called a primary keyword, is a
|   |   key in a relational database that is
|   |   unique for each record.  It is a unique
|   |   identifier, such as a driver license
|   |   number, telephone number (including area
|   |   code), or vehicle identification number
|   |   (VIN).  A relational database must always
|   |   have one and only one primary key.
|   |   Primary keys typically appear as columns
|   |   in relational database tables.``
|   |
https://searchsqlserver.techtarget.com/definition/primary-key
|   |
|   +- FOREIGN KEY:  ``A foreign key is a column
|       or columns of data in one table that
|       connects to the primary key data in the
|       original table.  To ensure the links
|       between foreign key and primary key
|       tables aren't broken, foreign key
|       constraints can be created to prevent
|       actions that would damage the links
|       between tables and prevent erroneous data
|       from being added to the foreign key
|       column.``
|       https://searchoracle.techtarget.com/definition/foreign-key
|
+- VIEWS:  ``In a database management system, a
|     view is a way of portraying information in
|     the database.``
|     https://whatis.techtarget.com/search/query
|
+- STRUCTURED (MODULAR) PROGRAMMING:
    |   ``Structured programming (sometimes known
    |   as modular programming) is a subset of
    |   procedural programming that enforces a
    |   logical structure on the program being
    |   written to make it more efficient and
    |   easier to understand and modify.  Certain
    |   languages such as Ada, Pascal, and dBASE
    |   are designed with features that encourage
    |   or enforce a logical program structure.``
```

```
                                |
                                https://searchsoftwarequality.techtarget.com/definition/structured-programming-modular-programming
                                |
                                +- FUNCTIONS:  ``In information technology,
                                |   the term function (pronounced FUHNK-shun)
                                |   has a number of  meanings.  It's taken
                                |   from the Latin ``functio`` -- to perform.
                                |   1) In its most general use, a function is
                                |   what a given entity does in being what it
                                |   is.
                                |   2) In C language and other programming, a
                                |   function is a named procedure that
                                |   performs a distinct service.  The
                                |   language statement that 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      schema_name.table_name
                      (
                        field_1 datatype_1 [attributes],
                        field_2 datatype_2 [attributes],
                        field_3 datatype_3 [attributes],
                        ...
                      );

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

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

```
    object type.

    4.01. In the example below, we create database `labs`.
 ********************************************************************** */

CREATE DATABASE labs;


/* **********************************************************************
    4.02. We then create schema `ace`, which must be called to be used when
          creating tables or other objects.

          There is no need to call the name of the schema when using the SQL
          Server default T-SQL schema `dbo` (database owner) -- not used in
          this example.
 ********************************************************************** */

CREATE SCHEMA ace;


/* **********************************************************************
    4.03. After creating the database (and the schema if needed), we can create
          the table.

                    CREATE TABLE table_name
                    (
                      field1 data type [null|not null] [unique] [primary key],
                      field2 data type [null|not null],
                      ...
                    )
 ********************************************************************** */

CREATE TABLE ace.students (                     -- 1. rule of thumb: table
                                                --    names in plural
  student_id INT NULL,                          -- 2. declared as INT;  can
                                                --    accept NULL (can have no
                                                --    value)
  student_fname VARCHAR(50) NULL,               -- 3. declared as VARCHAR(50);
                                                --    can accept NULL (can have
                                                --    no value)
  student_lname VARCHAR(50) NULL,               -- 4. declared as VARCHAR(50);
                                                --    can accept NULL (can have
                                                --    no value)
  student_phone VARCHAR(15) NULL,               -- 5. declared as VARCHAR(50);
                                                --    can accept NULL (can have
                                                --    no value)
  student_dob DATE NULL,                        -- 6. declared as DATE
                                                --
                                                --    DATETIME 04/20/2023 20:51
                                                --    DATE     04/20/2023
                                                --    TIME     20:51
                                                --
                                                --    can accept NULL (can have
                                                --    no value)
  record_date DATE NULL                         -- 5. declared as DATE;  when
                                                --    record was created;  can
                                                --    accept NULL (can have no
                                                --    value)
  );


/* **********************************************************************
    4.04. After creating table `students` in schema `ace`, we insert values for
          each column in the same order as the structure that we indicated in
          #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
```

```sql
DROP TABLE ace.students;


/* ************************************************************************
    5.2. In the case of tables, we can destroy (`TRUNCATE`) the data in the
         table without affecting the structure of the table understanding that,
         once we do, we cannot recover the data.
 ************************************************************************ */

TRUNCATE TABLE ace.students;


/* ************************************************************************
 6. We can also modify (`ALTER`) data objects.  We will start modifying tables
    (https://techonthenet.com/sql_server/tables/alter_table.php) since you
    might do this more often.

    ADD             to add a column to a table

    DROP            to delete a column to a table

    ALTER           to change the data type or size of a column
 ************************************************************************ */

ALTER TABLE ace.students                        -- 1. adding new column
ADD Email VARCHAR(100);                         --    `Email`;  no need to
                                                --    specify that you are
                                                --    adding a column

ALTER TABLE ace.students                        -- 2. dropping (deleting)
DROP COLUMN Email;                              --    column `Email` as there
                                                --    is no SQL statement to
                                                --    rename data objects;
                                                --    must specify that you are
                                                --    dropping a column

ALTER TABLE ace.students                        -- 3. adding new (replacement)
ADD student_email VARCHAR(100);                 --    column `student_email`;
                                                --    no need to specify that
                                                --    you are adding a column

ALTER TABLE ace.students                        -- 4. altering column with new
ALTER COLUMN student_email VARCHAR(50) NULL;    --    data type VARCHAR(50)
                                                --    from VARCHAR(100) and
                                                --    `NOT NULL`;  must specify
                                                --    that you are altering a
                                                --    column

ALTER TABLE ace.students                        -- 5. altering column as
ALTER COLUMN student_id INT NOT NULL;           --    `NOT NULL`;  must specify
                                                --    that you are altering a
                                                --    column

ALTER TABLE ace.students                        -- 6. altering column with new
ALTER COLUMN record_date DATETIME NOT NULL;     --    data type DATETIME
                                                --    from DATE and `NOT NULL`;
                                                --    must specify that you are
                                                --    altering a column

ALTER TABLE ace.students                        -- 7. altering column with new
ALTER COLUMN student_fname VARCHAR(25) NOT NULL;--    data type VARCHAR(25)
                                                --    from VARCHAR(50) and
                                                --    `NOT NULL`;  must specify
                                                --    that you are altering a
                                                --    column
```

```sql

ALTER TABLE ace.students                          -- 8. altering column with new
ALTER COLUMN student_fname VARCHAR(25) NOT NULL;--    data type VARCHAR(25)
                                                  --    from VARCHAR(50) and
                                                  --    `NOT NULL`;  must specify
                                                  --    that you are altering a
                                                  --    column

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.

    7.01. 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 = '';


/* ********************************************************************************
    7.02. 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.foo` 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.foo'
      ));


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

```
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.
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
```

```
/* ***************************************************************************
 https://folvera.commons.gc.cuny.edu/?p=1282
 *************************************************************************** */
```