```sql
/* *****************************************************************************
                     DATABASE ADMINISTRATION FUNDAMENTALS:
                   INTRODUCTION TO STRUCTURED QUERY LANGUAGE
                     SF21SQL1001, 2021/11/02 - 2021/12/09
                  https://folvera.commons.gc.cuny.edu/?cat=29
 *******************************************************************************

  SESSION #6 (2021/11/18): 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
```

```sql
         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).

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


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


/* ****************************************************************************
 2. Before we start creating and altering data objects, we have to understand
    data types (how data is stored).  These are the most often used data types.
    Refer to https://msdn.microsoft.com/en-us/library/ms187752.aspx for more
    information on data types in SQL Server.

    2.1. INT        -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)
                    https://technet.microsoft.com/en-us/library/ms187745.aspx

    2.2. DECIMAL    fixed precision and scale numbers...
                    10^38+1 through 10^38-1
                    https://msdn.microsoft.com/en-us/library/ms187746.aspx

                    instead of DOUBLE or FLOAT, indicating the whole value
                    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
                |   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      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`.
*************************************************************************** */

```

```sql
CREATE DATABASE sql_class;


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

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

CREATE SCHEMA ace;


/* *****************************************************************************
    4.3. 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 10/12/2019 13:51
                                                --    DATE     10/12/2019
                                                --    TIME     13:51
                                                --
                                                --    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)
  );


/* *****************************************************************************
    4.4. After creating table `students` in schema `ace`, we insert values for
         each column in the same order as the structure that we indicated in
         #4.3.

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

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


/* *****************************************************************************
    4.5. In the example below, we insert only three (3) values.

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

         4.5.2. We do not need to call columns in order as long order as long
```

```
                as values are pushed in the same order (value 1 in field 1,
                value 2 in field 2, value 3 in field 3 and value 7 in field 7).
 *************************************************************************** */

INSERT INTO ace.students (
  student_id,                                   -- 1. inserting values to only
  student_fname,                                --    four (4) columns;
  student_lname,                                --    indicating which four (4)
  record_date                                   --    columns
  )
VALUES (
  4,                                            -- 2. values to be inserted in
  'Smith',                                      --    columns `student_id`,
  'Tom',                                        --    `student_fname`,
  GETDATE()                                     --    `student_lname` and
  );                                            --    `record_date` receiving
                                                --    value from `GETDATE()`


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

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

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

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

INSERT INTO ace.students
VALUES (
  6,
  'John',
  'Scott',
  '',                                           -- 1. inserting empty strings
  '',                                           --    (``) or NULL since we
                                                --    have no values for fields
                                                --    to insert same number of
```

```sql
                                                --    values as columns
  GETDATE()                                     -- 2. built-in function to
                                                --    retrieve system DATETIME
  ),
  (
  5,
  'Mary Ann',
  'Saunders',
  '',                                           -- 3. inserting empty strings
  '',                                           --    (``) or NULL since we
                                                --    have no values for fields
                                                --    to insert same number of
                                                --    values as columns
  GETDATE()                                     -- 4. built-in function to
                                                --    retrieve system DATETIME
  );


/* *****************************************************************************
 5. We can also delete/destroy data objects.

    5.1. For the time being, we will work with tables
         (https://techonthenet.com/sql_server/tables/drop_table.php).

    5.2. Once an object is deleted, there is no way to rescue the data
         (ROLLBACK) unless first creating a SAVEPOINT
         (https://technet.microsoft.com/en-us/library/ms178157.aspx).

    5.3. In the example below, we destroy (`DROP`) table `ace.students`
         understanding that, once we do, we cannot recover the structure or the
         data.
 **************************************************************************** */

DROP TABLE ace.students;


/* *****************************************************************************
    5.4. 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.

    6.1. ADD        to add a column to a table

    6.2. DROP       to delete a column to a table

    6.3. 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

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.

    7.1. 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'
      ));


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

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


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

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


/* ****************************************************************************
 8. In the example below, we use `TRUNCATE` to delete all data from table
    `ace.students` without dropping (destroying) the table.
 *************************************************************************** */

TRUNCATE TABLE ace.students;


/* ****************************************************************************
 9. Since there is no copy statements in SQL, we are limited to the vendor
    extensions (vendor-specific SQL).

    9.1. When working with some vendors like Oracle, we can CREATE a new table
         from a query on another table.

                    CREATE TABLE new_table
                    AS
                      (
                        SELECT field1, field2 ...
                        FROM old_table
                      )

     9.2. In SQL Server, we use `INTO`.

                    SELECT field1, field2 ...
                      INTO new_table
                    FROM old_table

     9.3. In the example below, we push the output of the query to retrieve all
          values from table `ace.students` into `ace.students2`.

                    SELECT field1, field2 ...
                      INTO new_table
                    FROM old_table1
                    INNER|LEFT|RIGHT JOIN old_table2
                      ON old_table1.common_field1 = old_table2.common_field1...

          9.3.1. A view (http://searchsqlserver.techtarget.com/definition/view)
                 is a better option, which we will cover on the next class.
 *************************************************************************** */

SELECT *                                        -- 1. selecting all values
                                                --    from `ace.students`
INTO ace.students2                              -- 2. creating the new table
                                                --    `ace.students2`
FROM ace.students;                              -- 3. from table `ace.students`


/* *****************************************************************************
 10. LAB #5
     Write a query
     10.1. to call all columns and values shared by tables `AP1.ContactUpdates`
           and `AP1.Vendors` (`INNER JOIN`),
     10.2. retrieving only rows with `AP1.Vendors.VendorState` with values of
           `NY`, `NJ` and `CA`
     10.3. using `CASE` to replace `NY` to `New York`, `NJ` to `New Jersey`,
           `CA` to `California` and any other value to `Other`
     10.4. ordered first by `AP1.Vendors.VendorState` and then by
           `AP1.Vendors.VendorID`.
 *************************************************************************** */

SELECT AP1.ContactUpdates.VendorID,
  AP1.ContactUpdates.LastName,
  AP1.ContactUpdates.FirstName,
  -- AP1.Vendors.VendorID AS Expr1,             -- 1. duplicate column name
                                                --    commented out
  AP1.Vendors.VendorName,
  AP1.Vendors.VendorAddress1,
  AP1.Vendors.VendorAddress2,
  AP1.Vendors.VendorCity,
  CASE                                          -- 2. beginning of logic
    WHEN AP1.Vendors.VendorState = 'NY'         --    2.1. checking for value
      THEN 'New York'                           --         `NY` and return
                                                --         value `New York`
    WHEN AP1.Vendors.VendorState = 'NJ'         --    2.2. checking for value
      THEN 'New Jersey'                         --         `NY` and return
                                                --         value `New Jersey`
    WHEN AP1.Vendors.VendorState = 'CA'         --    2.3. checking for value
      THEN 'California'                         --         `NY` and return
                                                --         value `California`
    ELSE 'Other'                                --    2.4. checking for other
                                                --         values and return
                                                --         value `Other`
    END AS VendorState,
  AP1.Vendors.VendorZipCode,
  AP1.Vendors.VendorPhone,
  AP1.Vendors.VendorContactLName,
  AP1.Vendors.VendorContactFName,
  AP1.Vendors.DefaultTermsID,
  AP1.Vendors.DefaultAccountNo
FROM AP1.ContactUpdates
INNER JOIN AP1.Vendors
  ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
WHERE AP1.Vendors.VendorState IN (              -- 3. indicating what values we
    'NY',                                       --    query to return
    'NJ',
    'CA'
    );

/* ****************************************************************************
 https://folvera.commons.gc.cuny.edu/?p=1021
 *************************************************************************** */
```