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

  SESSION #7 (2021/11/29): CREATING DATABASE OBJECTS

  1. Understanding functions `CONVERT()`, `CAST()`, `DAY()`, `MONTH()`,
     `YEAR()` and `GETDATE()`
  2. Creating, dropping and altering views
 ******************************************************************************

 1. As a quick review, SQL is the language to interact with a relational
    database.

    1.1. to request data (`SELECT`) from database objects like databases,
         schemas, tables and views
    1.2. to create (`CREATE`) where to store data, database objects like
         databases, schemas, tables including columns, etc.`
    1.3. to modify (`ALTER`) database objects
    1.4. to delete (`DROP`) database objects, automatic `COMMIT` in SQL Server
         hence no `ROLLBACK` (no way to rescue the data or objects)
    1.5. to manipulate data either affecting the data or not (showing data)

                    CREATE obj_type object_name
                    [other_code]

                    DROP obj_type object_name
                    [other_code]

                    ALTER obj_type object_name
                    ALTER|ADD|DROP obj_type obj_name data_type [other_code]

                    DELETE FROM table_name
                    [other_code]

                    INSERT INTO table_name
                    VALUES
                      (
                        field1 datatype1,
                        field2 datatype2
                        ...
                      )

                    TRUNCATE TABLE table_name

                    UPDATE table_name
                    SET field = new_value

    1.6. We use SQL to return data to any person or program that needs data.

```

```sql
         1.6.1. Your boss or end user requests data as a report (`RPT`),
                graphic (`GIF`, `JPEG`, `PNG`, `BMP`, etc.), an office file
                (`XLS`, `DOC`, etc.) or other data types as `PDF`.  The end
                user does not need to know where the data comes from or how to
                get it.

         1.6.2. You (the middle person handling errands) get the data from the
                database using SQL.  Normally you would not take care of
                visualization, analysis and/or interpretation.  You also do not
                need to understand the data, but you need to know your data
                (row ID of tables, keys or other constraints, etc.) and make
                sure the data is clean (no garbage data).

         1.6.3. The database holds the data that the end user needs and all SQL
                requests (`SELECT`, `DROP`, `ALTER`, etc.) if written currently
                (no syntax errors) return the data to you or directly to the
                end user.  The database does not have AI and hence only returns
                what you ask.  As such if you make the wrong request, the
                database would return the wrong data (your bad logic, not a
                syntax error).
 ******************************************************************************

 2. In the example below, write a query without duplicate rows (`SELECT
    DISTINCT`)
    2.1. to call all shared values from tables `AP1.Invoices` and `AP1.Terms`,
    2.2. to format all dates as `yyyy-MM-dd` and currency as pounds sterling
         (culture `en-gb`)
    2.3. where `AP1.Invoices.PaymentTotal` is greater than the average value of
         `AP1.Invoices.InvoiceTotal` (sub-query within the `WHERE` clause) and
         `AP1.Invoices.PaymentDate` is not null.
 *************************************************************************** */

SELECT AP1.Invoices.InvoiceID,
  AP1.Invoices.VendorID,
  AP1.Invoices.InvoiceNumber,
  FORMAT(AP1.Invoices.InvoiceDate,              -- 1. formatting column
                                                --    `InvoiceDate` as
    'yyyy-MM-dd', 'en-gb')                      --    `yyyy-MM-dd` date with
                                                --    culture `en-gb` using
     AS InvoiceDate,                            --    alias `InvoiceDate`
  FORMAT(AP1.Invoices.InvoiceTotal,             -- 2. formatting column
                                                --    `InvoiceTotal` as
    'c', 'en-gb')                               --    `c` (currency) with
                                                --    culture `en-gb` using
    AS InvoiceTotal,                            --    alias `InvoiceTotal`
  FORMAT(AP1.Invoices.PaymentTotal,             -- 3. formatting column
    'c', 'en-gb')                               --    as `c` (currency) with
                                                --    culture `en-gb` using
    AS PaymentTotal,                            --    alias `PaymentTotal`
  FORMAT(AP1.Invoices.CreditTotal,              -- 4. formatting column
    'c', 'en-gb')                               --    as `c` (currency) with
                                                --    culture `en-gb` using
    AS CreditTotal,                             --    alias `CreditTotal`
  AP1.Invoices.TermsID,
  FORMAT(AP1.Invoices.InvoiceDueDate,           -- 5. formatting column
    'yyyy-MM-dd', 'en-gb')                      --    as `yyyy-MM-dd` date with
                                                --    culture `en-gb` using
    AS InvoiceDueDate,                          --    alias `InvoiceDueDate`
  FORMAT(AP1.Invoices.PaymentDate,              -- 6. formatting column
    'yyyy-MM-dd', 'en-gb')                      --    as `yyyy-MM-dd` date with
                                                --    culture `en-gb` using
    AS PaymentDate,                             --    alias `PaymentDate`
  AP1.Terms.TermsDescription,
  AP1.Terms.TermsDueDays
FROM AP1.Invoices                               -- 7. from table `AP1.Invoices`
INNER JOIN AP1.Terms                            --    using `INNER JOIN` to
                                                --    retrieve all shared data
  ON AP1.Invoices.TermsID = AP1.Terms.TermsID   --    connecting both tables on
                                                --    shared field `TermsID`
WHERE (                                         -- 8. where the value of
    AP1.Invoices.PaymentTotal > (               --    `PaymentTotal` is larger
                                                --    than (`>`) the single
      SELECT AVG(PaymentTotal)                  --    value of sub-query
      FROM AP1.Invoices                         --        `(SELECT
                                                --         AVG(PaymentTotal)
                                                --         FROM AP1.Invoices)`
                                                --    that returns 1879.7413
      )                                         --    8.1. sub-query always in
    )                                           --         parenthesis, just
                                                --         like in algebra
                                                --    8.2. no need for
                                                --         `ORDER BY` since
                                                --         aggregate function
                                                --         `AVG()` affects only
                                                --         one column and it
                                                --         does not affect the
                                                --         main query
  AND AP1.Invoices.PaymentDate IS NOT NULL;     -- 9. and [where] value of
                                                --    `PaymentDate` is not null
                                                --    (must have a value)


/* ****************************************************************************
  3. Although using a custom format like `yyyy-MM-dd` overrides the culture
     (`en-us`) and there is no longer need to include this culture, it is a
     good idea to include it as good practice.
 *************************************************************************** */

SELECT FORMAT(InvoiceTotal, 'yyyy-MM-dd')       -- no culture (`en-us`) needed
FROM AP1.Invoices;                              -- because of the custom format

SELECT FORMAT(InvoiceTotal,                     -- good practice to include the
  'yyyy-MM-dd', 'en-us')                        -- culture (`en-us`) even when
FROM AP1.Invoices;                              -- overridden by custom format
```

```sql


/* ****************************************************************************
  4. As mentioned several times, `FORMAT()` changes numeric values to strings.
     We can also use `CONVERT()` to change ``an expression from a data type to
     another data type`` -- in other words, numeric values to strings or vice
     versa (https://techonthenet.com/sql_server/functions/convert.php).

                    CONVERT(new_data_type, column)

     `CONVERT()` does not change the currency sign or adds commas to divide
     thousands or millions as `FORMAT()` does.

     4.1. In the example below, we change the data type of `InvoiceTotal` to
          VARCHAR(50) -- an allocation in RAM to hold a variable character
          value with a maximum size of fifty (50) characters.
 *************************************************************************** */

SELECT CONVERT(VARCHAR(50), InvoiceTotal)       -- changing data type of column
  AS InvoiceTotal                               -- `InvoiceTotal` (`FLOAT`) to
FROM AP1.Invoices;                              -- `VARCHAR(50)`


/* ****************************************************************************
     4.2. In the example below, we use `CONVERT()` to return the value of
          `AP1.Invoices.InvoiceTotal` as a dollar amount concatenating the
          dollar sign (`$`) at the beginning.
 *************************************************************************** */

SELECT CONCAT (
        '$',                                    -- concatenating `$` to the
        CONVERT(VARCHAR(50), InvoiceTotal)      -- output of
        ) AS InvoiceTotal                       --    `CONVERT(VARCHAR(50),
FROM AP1.Invoices;                              --    `InvoiceTotal)`


/* ****************************************************************************
          4.2.1. We could also use `CONVERT()` to return the value of
                 `AP1.Invoices.InvoiceTotal` as a dollar amount with `USD `
                 rather than the dollar sign (`$`).
 *************************************************************************** */

SELECT CONCAT (
        'USD ',                                 -- concatenating `USD ` to the
        CONVERT(VARCHAR(50), InvoiceTotal)      -- output of
        ) AS InvoiceTotal                       --    `CONVERT(VARCHAR(50),
FROM AP1.Invoices;                              --    `InvoiceTotal)`


/* ****************************************************************************
          4.2.2. Of course, if you are ``dressing up`` a numeric value like
                 `AP1.Invoices.InvoiceTotal` as currency, it is better to just
                 use `FORMAT()` to keep your code simple.
 *************************************************************************** */

SELECT FORMAT(InvoiceTotal, 'c', 'en-us') AS InvoiceTotal
FROM AP1.Invoices;


/* ****************************************************************************
     4.3. In the example below, we use `CONVERT()` to change the data type of
          `AP1.Invoices.InvoiceID` and `AP1.Invoices.VendorID` from FLOAT to
          `VARCHAR(50)` before concatenating these values to a string.
 *************************************************************************** */

SELECT CONCAT (                                 -- 1. concatenating string
                                                --    values of
    'Invoice ',                                 --    1.1. `Invoice `
                                                --         (hard-coded),
    CONVERT(VARCHAR(3), InvoiceID),             --    1.2. the conversion of
                                                --         `InvoiceID` to
                                                --         `VARCHAR(3)`,
    ' from vendor ',                            --    1.3. ` from vendor `
                                                --         (hard-coded) and
    CONVERT(VARCHAR(3), VendorID)               --    1.4. the conversion of
                                                --         `VendorID` to
                                                --         `VARCHAR(3)`
    ) AS InvoiceVendor                          --    1.5. using alias
                                                --         `InvoiceVendor`
FROM AP1.Invoices;


/* ****************************************************************************
 5. We use the `WHERE` (https://techonthenet.com/sql_server/where.php)
    clause to ``filter the results from a SELECT, INSERT, UPDATE, or DELETE
    statement.``

                    SELECT table1.field1, table1.field2 ...
                      table2.field1, table2.field2 ...
                    FROM table1
                      INNER|LEFT|RIGHT JOIN table2
                      ON table1.shared_field1 = table2.shared_field1
                        AND table1.shared_field2 = table2.shared_field2
                      ...
                    WHERE condition1
                      AND|OR condition2
                      ...

    5.1. We use conditions in order to filter data.

         5.1.1. AND       to test for two or more conditions
                          https://techonthenet.com/sql_server/and.php

         5.1.2. OR        to test multiple conditions where records are
```

```
                          returned when any one of the conditions are met
                          https://techonthenet.com/sql_server/or.php

    5.2. We use operators to compare values.

         5.2.1. =         equal to
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.2. <>        not equal to
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.3. !=        not equal to
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.4. <         less than
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.5. >         greater than
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.6. <=        less than or equal to
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.7. >=        greater than or equal to
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.8. !>        not greater than (same as <=)
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.9. !<        not less than (same as >=)
                          https://techonthenet.com/sql_server/comparison_operators.php

         5.2.10. LIKE     allows wild cards to be used in the WHERE clause of a
                          SELECT, INSERT, UPDATE, or DELETE statement
                          [allowing] you to perform pattern matching
                          https://techonthenet.com/sql_server/like.php

         5.2.11. IN       to help reduce the need to use multiple OR conditions
                          in a SELECT, INSERT, UPDATE, or DELETE statement
                          https://techonthenet.com/sql_server/in.php

         5.2.12. BETWEEN  used to retrieve values within a range in a SELECT,
                          INSERT, UPDATE, or DELETE statement
                          https://techonthenet.com/sql_server/between.php

         5.2.13. IS NULL  condition... used to test for a NULL no value
                          https://techonthenet.com/sql_server/is_null.php

         5.2.14. NOT      to negate a condition in a SELECT, INSERT, UPDATE, or
                          DELETE statement
                          https://techonthenet.com/sql_server/not.php

                          * NOT LIKE
                          * NOT IN
                          * NOT BETWEEN
                          * IS NOT NULL
                            https://techonthenet.com/sql_server/is_not_null.php

    5.3. In the example, below, we retrieve all values from table `AP1.Vendors`
         where `VendorState` is equal to `CA` and `VendorCity` could either be
         `Fresno` or `Sacramento`.

         Use parenthesis for SQL (regardless of vendor/distribution) to process
         the inner condition first

                    (
                      VendorCity = 'Fresno'
                      OR VendorCity = 'Sacramento'
                    )

         before the outer condition.
 *************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorState = 'CA'                        -- 1. inner criterion that must
                                                --    be true (satisfied)
  AND (                                         -- 2. outer criterion that must
                                                --    be true composed of two
                                                --    sections where either
                                                --    could be true (satisfied)
    VendorCity = 'Fresno'                       --    3.1. first criteria that
                                                --         could be met
    OR VendorCity = 'Sacramento'                --    3.2. second criteria that
                                                --         could be met
    );


/* ****************************************************************************
    5.4. In the example below, we retrieve all values from table `AP1.Vendors`
         where `VendorState` is not (`<>` or `!=`) `NY`.
 *************************************************************************** */

SELECT *
FROM AP1.Vendors                                -- can also be written as
```

```
WHERE VendorState <> 'NY';                      -- `VendorState != `NY``


/* *****************************************************************************
    5.5. In the example below, we retrieve all values from table `AP1.Vendors`
         where `VendorState` is either `DC` or `IA`.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorState = 'DC'                        -- checking if either criterion
  OR VendorState = 'IA';                        -- is true


/* *****************************************************************************
    5.6. In the example below, we retrieve all values from table `AP1.Vendors`
         where `VendorAddress2` is NULL (no-value) using `NOT` as it negates
         operators `LIKE` as `NOT LIKE`, `IN` as `NOT IN`, `BETWEEN` as
         `NOT BETWEEN` and `IS NULL` as `IS NOT NULL`.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorAddress2 IS NULL;                   -- asking for no-value


/* *****************************************************************************
    5.7. In the example below, we retrieve all values from table `AP1.Vendors`
         where `VendorAddress2` is not NULL (not a no-value).  Refer to
         https://techonthenet.com/sql_server/is_not_null.php for more
         information.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorAddress2 IS NOT NULL;               -- asking for not `NOT NULL`
                                                -- (no no-value)


/* *****************************************************************************
    5.8. In the example below, we rewrite #6.3 in a cleaner fashion to retrieve
         all values from table `AP1.Vendors` where `VendorState` is equal to
         `CA` and `VendorCity` could either be `Fresno` or `Sacramento`.  We
         use operator `IN` (https://techonthenet.com/sql_server/in.php) to
         specify the list of values that can be true for `VendorCity`.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorState = 'CA'                        -- 1. first condition as in
                                                --    original example
  AND VendorCity IN (                           -- 2. second condition using
    'Fresno',                                   --    `IN` to list all possible
    'Sacramento'                                --    values that can be true
    );                                          --    (satisfied)


/* *****************************************************************************
    5.9. In the example below, we retrieve all values from table `AP1.Vendors`
         where `VendorState` could either be `CA` or `NJ` and `VendorCity`
         could either be `Fresno` or `Sacramento`.

         This query looks for the combination of

                `CA` and `Fresno`     (true)
                `CA` and `Sacramento` (true)

         as well as

                `NJ` and `Fresno`     (false)
                `NJ` and `Sacramento` (false)

         The query only returns only the first set of values since we do not
         have any records where `VendorCity` is `NJ` and VendorCity` is either
         `Fresno` or `Sacramento`.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE (
    VendorState IN (                            -- 1. first condition using
      'CA',                                     --    `IN` to list all possible
      'NJ'                                      --    values that can be true
      )                                         --    (satisfied)
    AND VendorCity IN (                         -- 2. second condition using
      'Fresno',                                 --    `IN` to list all possible
      'Sacramento'                              --    values that can be true
      )                                         --    (satisfied)
    )
ORDER BY VendorState,
  VendorCity;


/* *****************************************************************************
    5.10. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorState` could either be `CA` and `VendorCity` could
          either be `Fresno` or `Sacramento` as one condition or `VendorState`
          is `NJ` as another condition.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE (                                         -- 1. first condition where
    VendorState IN ('CA')                       --    `VendorState` could be
```

```sql
    AND VendorCity IN (                         --    `CA` and `VendorCity`
      'Fresno',                                 --    could either be `Fresno`
      'Sacramento'                              --    or `Sacramento`
      )                                         --    looking the combinations
    )                                           --    of `CA` and `Fresno` or
                                                --    `CA` and `Sacramento`
  OR VendorState IN ('NJ')                      -- 2. second condition starting
                                                --    with `OR` to specify that
                                                --    `VendorState` could also
                                                --    be `NJ`
ORDER BY VendorState,                           -- 3. ordering results first by
  VendorCity;                                   --    `VendorState` and then by
                                                --    `VendorCity`


/* *****************************************************************************
    5.11. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorName` has as a value starting with `am` (not case
          sensitive) using wild card `%` to represent any character or group of
          after `am`.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorName LIKE 'am%';                    -- returns values
                                                -- `American Booksellers Assoc`
                                                -- and `American Express`


/* *****************************************************************************
    5.12. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorName` has as a value with pattern `data` (not case
          sensitive) using wild card `%` before and after the given string.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorName LIKE '%data%';                 -- returns various values like
                                                -- `Expedata Inc`,
                                                -- `California Data Marketing`
                                                -- and `Quality Education Data`


/* *****************************************************************************
    5.13. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorPhone` has as a value starting with `800` (string, not a
          numeric value).
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorPhone LIKE '800%';
```

```sql


/* *****************************************************************************
    5.14. In the example below, we retrieve all values from table `AP1.Vendors`
          where `VendorPhone` has as a value NOT starting with `800`.
 **************************************************************************** */

SELECT *
FROM AP1.Vendors
WHERE VendorPhone NOT LIKE '800%';


/* *****************************************************************************
    5.15. In the example below, we retrieve all values from table
          `AP1.Invoices` where `InvoiceDueDate` has values within the range of
          two dates -- `2012-01-01` and `2012-01-30` (dates always in single
          quotes).
 **************************************************************************** */

SELECT *
FROM AP1.Invoices
WHERE InvoiceDueDate BETWEEN '2012-01-01'       -- range between `2012-01-01`
    AND '2012-01-30';                           -- and `2012-01-30`


/* *****************************************************************************
    5.16. In the example below, we retrieve all values from table `AP1.Vendors`
          where InvoiceTotal has values within 100 and 1000.  Then we organize
          the results in descending order using an `ORDER BY` clause
          (https://techonthenet.com/sql/order_by.php).

          The default option for `ORDER BY` is `ASC` (ascending), which can be
          omitted.

          The opposite option for `ORDER BY` is `DESC` (descending), which
          needs to be specified.
 **************************************************************************** */

SELECT *
FROM AP1.Invoices
WHERE InvoiceTotal BETWEEN 100                  -- range between 100 and 1000
    AND 1000
ORDER BY InvoiceTotal DESC,                     -- organizing results first by
                                                -- `InvoiceTotal` in descending
                                                -- order,
  PaymentTotal DESC,                            -- then by `PaymentTotal` in
                                                -- descending order
  TermsID DESC;                                 -- and finally by `TermsID`
                                                -- also in descending order


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

```
 6. As we have mentioned several times, when calling multiple tables, we need
    to `JOIN` them (https://techonthenet.com/sql_server/joins.php).

    `INNER JOIN` returns ``all rows from multiple tables where the join
    condition is met.``

    6.1. In the example below, we retrieve all records shared in tables
         `AP1.Invoices` and `AP1.Invoices`.
 *************************************************************************** */

SELECT *
FROM AP1.Vendors
INNER JOIN AP1.Invoices
  ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;


/* ****************************************************************************
    6.2. `LEFT JOIN` returns ``all rows from the LEFT-hand table specified in
         the ON condition and only those rows from the other table where the
         joined fields are equal (join conditions met).``

         6.2.1. In the example below, we retrieve all records in `AP1.Vendors`
                (left table) and any records in `AP1.Invoices` (if any in the
                right table).
 *************************************************************************** */

SELECT *
FROM AP1.Vendors                                -- retrieves all records from
LEFT JOIN AP1.Invoices                          -- the left table/dataset
  ON AP1.Vendors.VendorID = AP1.Invoices.VendorID; -- (first table/dataset
                                                -- called in the statement,
                                                -- `AP1.Vendors`) and related
                                                -- records from the right
                                                -- table/dataset (second
                                                -- table/dataset called in the
                                                -- statement, `AP1.Invoices`);
                                                -- returns 202


/* ****************************************************************************
         6.2.2. In the example below, we retrieve all records in `AP1.Invoices`
                (left table) and any records in `AP1.Vendors` (if any in the
                right table).
 *************************************************************************** */

SELECT *
FROM AP1.Invoices                               -- retrieves all records from
LEFT JOIN AP1.Vendors                           -- the left table/dataset
  ON AP1.Vendors.VendorID = AP1.Invoices.VendorID; -- (first table/dataset
                                                -- called in the statement,
                                                -- `AP1.Invoices`) and related
                                                -- records from the right
```

```sql
                                                -- table/dataset (second
                                                -- table/dataset called in the
                                                -- statement, `AP1.Vendors`)


/* *****************************************************************************
    6.3. `RIGHT JOIN` returns ``all rows from the RIGHT-hand table specified in
         the ON condition and only those rows from the other table where the
         joined fields are equal (join condition is met).``

         6.3.1. In the example below, we retrieve all records in `AP1.Invoices`
                (right table) and any records in `AP1.Vendors` (if any in the
                left table).
 **************************************************************************** */

SELECT *
FROM AP1.Vendors                                -- retrieves all records from
RIGHT JOIN AP1.Invoices                         -- the right table/dataset
  ON AP1.Invoices.VendorID = AP1.Vendors.VendorID; -- (second table/dataset
                                                -- called in the statement,
                                                -- `AP1.Invoices`) and related
                                                -- records from the left
                                                -- table/dataset (first
                                                -- table/dataset called in the
                                                -- statement, `AP1.Invoices`)


/* *****************************************************************************
    6.4. `FULL JOIN` returns ``all rows from the LEFT-hand table and RIGHT hand
         table with nulls in place where the join condition is not met.``

         6.4.1. Depending on the size of the tables, this query might make the
                server run slowly or crash it.

         6.4.2. In the example below, we retrieve all records in `AP1.Invoices`
                (left table) and all records in `AP1.Vendors` (if any in the
                right table).
 **************************************************************************** */

SELECT *
FROM AP1.Invoices                               -- retrieves all records from
FULL JOIN AP1.Vendors                           -- the left table/dataset
  ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;  -- (first table/dataset
                                                -- called in the statement,
                                                -- `AP1.Vendors`) and all
                                                -- records from the right
                                                -- table/dataset (second
                                                -- table/dataset called in the
                                                -- statement, `AP1.Invoices`)


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

```
 7. In the example below, we make some changes to `AP1.ContactUpdates` and
    `AP1.Vendors`.

    7.1. We add column `Email` to `AP1.ContactUpdates`, which should be
         `VARCHAR(100)` and `NOT NULL` (HINT: `UPDATE` first, then `NOT NULL`).

         7.1.1. First you need to add the column to the table.
 *************************************************************************** */

ALTER TABLE AP1.ContactUpdates
ADD Email VARCHAR(100);


/* ****************************************************************************
    7.2. Then you have to populate the column (every field).

         If you use `LastName` as part of the email, you should remove the
         apostrophe in `O'Sullivan`.

         Make sure to push the new values to an existant row in lower case
         (HINT: `UPDATE`).
 *************************************************************************** */

UPDATE AP1.ContactUpdates
SET Email = LOWER(CONCAT (
      LEFT(FirstName, 1),                       -- 1. from `Geraldine`
                                                --    returns `G`
      REPLACE(LastName, '''', ''),              -- 2. from `O'Sullivan`
                                                --    returns `OSullivan`
      '@domain.web'                             -- 3. returns
                                                --    `GOSullivan@domain.web`
      ));                                       -- 4. returns
                                                --    `gosullivan@domain.web`

/* ****************************************************************************
    7.3. Then you can change the column to `NOT NULL`.
 *************************************************************************** */

ALTER TABLE AP1.ContactUpdates
ALTER COLUMN Email VARCHAR(100) NOT NULL;


/* ****************************************************************************
     7.4. We then add column `VendorAddress` to `AP1.Vendors`, which should be
          `VARCHAR(150)` and `NOT NULL`.
 *************************************************************************** */

ALTER TABLE AP1.Vendors
ADD VendorAddress VARCHAR(150);


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

```
     7.5. Move the values of `VendorAddress1` and `VendorAddress2` to
          `VendorAddress`.
 *************************************************************************** */

UPDATE AP1.Vendors
SET VendorAddress = CONCAT (
    VendorAddress1,
    ' ',
    VendorAddress2
    );

/* ****************************************************************************
     7.6. Make sure the new column has the data and delete the original two
          columns.
 *************************************************************************** */

ALTER TABLE AP1.Vendors
DROP COLUMN VendorAddress1;

ALTER TABLE AP1.Vendors
DROP COLUMN VendorAddress2;


/* ****************************************************************************
     7.7. Change the new column to `NOT NULL`.
 *************************************************************************** */

ALTER TABLE AP1.Vendors
ALTER COLUMN VendorAddress VARCHAR(150) NOT NULL;


/* ****************************************************************************
     7.8. Call all the values from `AP1.ContactUpdates` with any corresponding
          values in `AP1.Vendors` (HINT: `LEFT JOIN` to get 8 records).
 *************************************************************************** */

SELECT *
FROM AP1.ContactUpdates
LEFT JOIN AP1.Vendors
  ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID;


/* ****************************************************************************
     7.9. As a bonus, make a view named `AP1.ContactUpdates_VendorsVW` from the
          prior query (#7.8).  See #9 for more information regarding views.
 *************************************************************************** */

CREATE VIEW AP1.ContactUpdates_VendorsVW
AS
(
    SELECT AP1.ContactUpdates.VendorID,
      AP1.ContactUpdates.LastName,
```

```sql
      AP1.ContactUpdates.FirstName,
      AP1.ContactUpdates.Email,
      -- AP1.Vendors.VendorID AS Expr1,
      AP1.Vendors.VendorName,
      AP1.Vendors.VendorCity,
      AP1.Vendors.VendorState,
      AP1.Vendors.VendorZipCode,
      AP1.Vendors.VendorPhone,
      AP1.Vendors.VendorContactLName,
      AP1.Vendors.VendorContactFName,
      AP1.Vendors.DefaultTermsID,
      AP1.Vendors.DefaultAccountNo,
      AP1.Vendors.VendorAddress
    FROM AP1.ContactUpdates
    LEFT JOIN AP1.Vendors
      ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
    );


/* *****************************************************************************
 8. Now that we have reviewed most of the material so far, we start views.

         ``In a database management system, a view is a way of portraying
         information in the database.  This can be done by arranging the data
         items in a specific order, by highlighting certain items, or by
         showing only certain items.  For any database, there are a number of
         possible views that may be specified.  Databases with many items tend
         to have more possible views than databases with few items.  Often
         thought of as a virtual table, the view doesn't actually store
         information itself, but just pulls it out of one or more existing
         tables.  Although impermanent, a view may be accessed repeatedly by
         storing its criteria in a query.``

         http://searchsqlserver.techtarget.com/definition/view

                    CREATE VIEW view_name AS
                      SELECT columns
                      FROM tables
                      [WHERE conditions];

    8.1. In the example below, we modify table `AP1.Invoices` adding column
         `CustomerID` in order to establish a relation between this table and
         `AP2.Customers`.
 **************************************************************************** */

ALTER TABLE AP1.Invoices
ADD CustomerID INT NULL;

UPDATE AP1.Invoices
SET CustomerID = 1
WHERE VendorID = 34;

```

```sql
UPDATE AP1.Invoices
SET CustomerID = 2
WHERE VendorID = 37;

UPDATE AP1.Invoices
SET CustomerID = 3
WHERE VendorID = 89;


/* ****************************************************************************
    8.2. Now that relationship has been created, we can now query tables
         `AP1.Invoices` and `AP2.Customers` (each tables in a different
         databases).
 *************************************************************************** */

SELECT DISTINCT AP1.Invoices.InvoiceID,
  AP1.Invoices.VendorID,
  AP1.Invoices.InvoiceNumber,
  FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
  FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
  FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
  FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
  AP1.Invoices.TermsID,
  FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
  FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
  AP1.Invoices.CustomerID,
  AP2.Customers.LastName,
  AP2.Customers.FirstName,
  AP2.Customers.Address,
  AP2.Customers.City,
  AP2.Customers.STATE,
  AP2.Customers.ZipCode,
  AP2.Customers.Email
FROM AP1.Invoices
INNER JOIN AP2.Customers
  ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
ORDER BY AP1.Invoices.VendorID;


/* ****************************************************************************
    8.3. In the example below, we can create a view using the query in the
          example above using tables `AP1.Invoices` and `AP2.Customers` without
          `ORDER BY`, which would return an error when creating the view.

         Tables and views cannot share names since both data objects are of the
         same hierarchy.

         We can query, alter and/or drop a view just like a table.

         In most relational databases, we cannot update data using a view since
         this action only take place in tables.

         In SQL Server (T-SQL), we can update data from the base table.

                    ``Requires UPDATE, INSERT, or DELETE permissions on the
                    target table, depending on the action being performed.``
                    https://msdn.microsoft.com/en-us/library/ms180800.aspx
 **************************************************************************** */

CREATE VIEW AP1.InvoicesCustomersVW
AS
(
    SELECT DISTINCT AP1.Invoices.InvoiceID,
      AP1.Invoices.VendorID,
      AP1.Invoices.InvoiceNumber,
      FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
      FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
      FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
      FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
      AP1.Invoices.TermsID,
      FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
      FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
      AP1.Invoices.CustomerID,
      AP2.Customers.LastName,
      AP2.Customers.FirstName,
      AP2.Customers.Address,
      AP2.Customers.City,
      AP2.Customers.STATE,
      AP2.Customers.ZipCode,
      AP2.Customers.Email
    FROM AP1.Invoices
    INNER JOIN AP2.Customers
      ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
    );


/* *****************************************************************************
    8.4. We can modify a view simply changing `CREATE` for `ALTER`.
 **************************************************************************** */

ALTER VIEW AP1.InvoicesCustomersVW
AS
(
    SELECT DISTINCT AP1.Invoices.InvoiceID,
      AP1.Invoices.VendorID,
      AP1.Invoices.InvoiceNumber,
      FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us')
        AS InvoiceDate,
      FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
      FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
      FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
      AP1.Invoices.TermsID,
      FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
      FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
```

```sql
      AP1.Invoices.CustomerID,
      AP2.Customers.LastName,
      AP2.Customers.FirstName,
      AP2.Customers.Address,
      AP2.Customers.City,
      AP2.Customers.STATE,
      AP2.Customers.ZipCode,
      AP2.Customers.Email,
      GETDATE() AS SystemDate -- change in query
    FROM AP1.Invoices
    INNER JOIN AP2.Customers
      ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
    );


/* ****************************************************************************
    8.5. In the example below, we create view `AP1.InvoicesVW` only from table
         `AP1.Invoices` formatting the date and currency fields accordingly.
         This way we do not need to format the columns again and again every
         time we need to call them.
 *************************************************************************** */

CREATE VIEW AP1.InvoicesVW
AS
(
    SELECT DISTINCT InvoiceID,
      VendorID,
      InvoiceNumber,
      FORMAT(InvoiceDate, 'd', 'en-us') AS InvoiceDate,
      FORMAT(InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
      FORMAT(PaymentTotal, 'c', 'en-us') AS PaymentTotal,
      FORMAT(CreditTotal, 'c', 'en-us') AS CreditTotal,
      TermsID,
      FORMAT(InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
      FORMAT(PaymentDate, 'd', 'en-us') AS PaymentDate,
      CustomerID
    FROM AP1.Invoices
    );


/* ****************************************************************************
    8.6. In the example below, we create view `AP1.InvoicesVendorsVW` from
         tables `AP1.Invoices` and `AP1.Vendors`.

         Unless we indicate in which database to store the view, it would most
         likely be in the same database where the previous view was stored
         (`AP2`).
 *************************************************************************** */

CREATE VIEW AP1.InvoicesVendorsVW
AS
(
    SELECT DISTINCT AP1.Invoices.InvoiceID,
      AP1.Invoices.VendorID,
      AP1.Invoices.InvoiceNumber,
      AP1.Invoices.InvoiceDate,
      AP1.Invoices.InvoiceTotal,
      AP1.Invoices.PaymentTotal,
      AP1.Invoices.CreditTotal,
      AP1.Invoices.TermsID,
      AP1.Invoices.InvoiceDueDate,
      AP1.Invoices.PaymentDate,
      AP1.Vendors.VendorName,
      CASE
        WHEN AP1.Vendors.VendorAddress2 IS NOT NULL
          THEN CONCAT (
              AP1.Vendors.VendorAddress1,
              ' ',
              AP1.Vendors.VendorAddress2
              )
        WHEN AP1.Vendors.VendorAddress1 IS NULL
          AND AP1.Vendors.VendorAddress2 IS NULL
          THEN 'No Address'
        ELSE AP1.Vendors.VendorAddress1
        END AS VendorAddress,
      AP1.Vendors.VendorCity,
      AP1.Vendors.VendorState,
      AP1.Vendors.VendorZipCode,
      AP1.Vendors.DefaultAccountNo
    FROM AP1.Invoices
    LEFT JOIN AP1.Vendors
      ON AP1.Invoices.VendorID = AP1.Vendors.VendorID
    );


/* *****************************************************************************
    8.7. In the example below, we create view
         `AP1.Invoices_Customers_Vendors_VW` from views (like we would do with
         tables) `AP1.InvoicesCustomersVW` and `AP1.InvoicesVendorsVW`.

         As mentioned, unless we indicate in which database to store the new
         view, it is saved in `AP2`.

         We do not need to call the database and schema (`dbo`), but it is
         always a good idea -- good practice.
 **************************************************************************** */

CREATE VIEW AP1.Invoices_Customers_Vendors_VW
AS
(
    SELECT DISTINCT AP1.InvoicesCustomersVW.InvoiceID,
      AP1.InvoicesCustomersVW.VendorID,
      AP1.InvoicesCustomersVW.InvoiceNumber,
      AP1.InvoicesCustomersVW.InvoiceDate,
      AP1.InvoicesCustomersVW.InvoiceTotal,
      AP1.InvoicesCustomersVW.PaymentTotal,
      AP1.InvoicesCustomersVW.CreditTotal,
      AP1.InvoicesCustomersVW.TermsID,
      AP1.InvoicesCustomersVW.InvoiceDueDate,
      AP1.InvoicesCustomersVW.PaymentDate,
      AP1.InvoicesCustomersVW.CustomerID,
      AP1.InvoicesCustomersVW.LastName,
      AP1.InvoicesCustomersVW.FirstName,
      AP1.InvoicesCustomersVW.Address,
      AP1.InvoicesCustomersVW.City,
      AP1.InvoicesCustomersVW.STATE,
      AP1.InvoicesCustomersVW.ZipCode,
      AP1.InvoicesCustomersVW.Email,
      AP1.InvoicesVendorsVW.VendorName,
      AP1.InvoicesVendorsVW.VendorAddress,
      AP1.InvoicesVendorsVW.VendorCity,
      AP1.InvoicesVendorsVW.VendorState,
      AP1.InvoicesVendorsVW.VendorZipCode,
      AP1.InvoicesVendorsVW.DefaultAccountNo
    FROM AP1.InvoicesCustomersVW
    LEFT OUTER JOIN AP1.InvoicesVendorsVW
      ON AP1.InvoicesCustomersVW.VendorID = AP1.InvoicesVendorsVW.VendorID
    );


/* ******************************************************************************
 9. Depending on the relational database management system (RDBMS) and even the
    product related to each RDBMS, the date format might vary.  In SQL Server,
    we can query data using format `YYYY/MM/DD` (including quotes) although the
    system returns format `YYYY-MM-DD` plus time in format `hh:mm:ss.nnnnnnn`.
    Refer to https://msdn.microsoft.com/en-us/library/bb630352.aspx and
    https://msdn.microsoft.com/en-us/library/bb677243.aspx for information on
    date and time respectively.

    9.1. The most common date functions are the following.

         9.1.1. DAY       returns the day of the month (1 to 31) given a date
                          value
                          https://techonthenet.com/sql_server/functions/day.php

         9.1.2. MONTH     returns the month (1 to 12) given a date value
                          https://techonthenet.com/sql_server/functions/month.php

         9.1.3. YEAR      returns a four-digit year (as a number) given a date
                          value
                          https://techonthenet.com/sql_server/functions/year.php

         9.1.4. GETDATE   returns the current date and time
                          https://techonthenet.com/sql_server/functions/getdate.php
 **************************************************************************** */

SELECT DAY('2021/09/15') AS Day,                -- 1. returns `15` from
                                                --    `2021/09/15` without
                                                --    leading zeros (`d`)
  MONTH('2021/09/15') AS Month,                 -- 2. returns `9` from
                                                --    `2021/09/15` without
                                                --    leading zeros (`M`)
  YEAR('2021/09/15') AS Year;                   -- 3. returns `2021` from
                                                --    `2021/09/15 (`yyyy`)

SELECT GETDATE() AS CurrentDateTime;            -- returns
                                                -- `2021-09-15 20:20:34.053`
                                                -- from `GETDATE()` that calls
                                                -- system date and time

SELECT DAY(GETDATE()) AS Day,                   -- 1. returns `15` from system
                                                --    DATETIME without leading
                                                --    zeros (`d`)
  MONTH(GETDATE()) AS Month,                    -- 2. returns `9` from system
                                                --    DATETIME without leading
                                                --    zeros (`M`)
  YEAR(GETDATE()) AS Year;                      -- 3. returns `2021` from
                                                --    system DATETIME (`yyyy`)

SELECT FORMAT(GETDATE(), 'd', 'en-us')          -- returns system date and time
  AS FormattedCurrentDateTime;                  -- formatted as `9/15/2021`


/* *****************************************************************************
    9.2. Instead of hard-coding the date in the example above (#3.1), we can
         use parameter `@date` in all instances that we need to pass the value
         returned by `GETDATE()`.

         We must declare each parameter with its proper data type.

         We can then have to pass (`SET`) a value for each parameter.
 **************************************************************************** */

DECLARE @date DATETIME = GETDATE()              -- 1. declaring parameter
                                                --    `@date` as DATETIME (the
                                                --    proper data type) and
                                                --    passing value of
                                                --    `GETDATE()`

SELECT DAY(@date) AS Day,                       -- 2. returns `9` from system
                                                --    DATETIME without leading
                                                --    zeros (`d`)
  MONTH(@date) AS Month,                        -- 3. returns `15` from system
                                                --    DATETIME without leading
                                                --    zeros (`M`)
  YEAR(@date) AS Year;                          -- 4. returns `2021` from
```

```
                                                --    system DATETIME (`yyyy`)


/* ****************************************************************************
    9.3. We can also use date function `GETDATE()` to calculate age in months,
         days and years.  The following script is based on the answer found at
         http://stackoverflow.com/q/57599/, which is explained below in detail.

         9.3.1. We declare variables `@start_date`, `@end_date` and `@tmp_date`
                as data type DATETIME
                (https://msdn.microsoft.com/en-us/library/ms187819.aspx).

         9.3.2. It is good practice to use a second variable (in this case,
                `@tmp_date`) for calculations or other forms of data
                manipulation.

         9.3.3. We declare `@years`, `@months` and `@days` as INT
                (https://msdn.microsoft.com/en-us/library/ms187745.aspx) for
                date functions `DATEADD()` and `DATEDIFF()`.
 *************************************************************************** */

DECLARE @persons_name VARCHAR(100),             -- 1. person's first and last
                                                --    names
  @start_date DATETIME,                         -- 2. person's birthday
  @end_date DATETIME,                           --    today's date from system
                                                --    date and time
  @tmp_date DATETIME,                           -- 3. variable for calculations
  @years INT,                                   -- 4. variable for number of
                                                --    years
  @months INT,                                  -- 5. variable for number of
                                                --    months
  @days INT;                                    -- 6. variable for number of
                                                --    days


/* ****************************************************************************
         9.3.4. We assign a value to `@start_date` as shown below since there
                is no way for SQL Server to prompt the user to enter a value.
                In this example, we are using the date of birth of Linus
                Torvalds (creator of the Linux kernel;
                http://searchenterpriselinux.techtarget.com/definition/Linus-Torvalds).
                We also assign `GETDATE()` to `@end_date`.  This way we can
                change the end date as needed (change from original query).
 *************************************************************************** */

SET @persons_name = 'Linus Torvalds';           -- person's name
SET @start_date =   '12/28/1969';               -- person's date of birth
SET @end_date =     GETDATE();                  -- today's system date and time


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

```sql
         9.3.5. We assign the value of `@start_date` to `@tmp_date` to run
                calculations against it and use `@start_date` as a constant.
 *************************************************************************** */

SELECT @tmp_date = @start_date;


/* ***************************************************************************
         9.3.6. Date functions `DATEADD()` returns ``a specified date with the
                specified number interval (signed integer) added to a specified
                datepart of that date``
                (https://msdn.microsoft.com/en-us/library/ms186819.aspx) and
                `DATEDIFF()` returns ``the count (signed integer) of the
                specified datepart boundaries crossed between the specified
                start_date and end_date``
                (https://msdn.microsoft.com/en-us/library/ms189794.aspx).

               `YEAR()` retrieves the year (`yy`) from the date.

               `MONTH()` retrieves the month (`m`) from the date.

               `DAY()` retrieves the day (`d`) from the date.

         9.3.7. The `CASE WHEN` statement uses a true value (situation we are
                looking for) coming from `WHEN... THEN` to trigger an action
                and an `ELSE` value to trigger an alternative action using the
                following syntax.

         9.3.8. Below `@years` is assigned the difference of `@tmp_date` and
                `@end_date` in years when the month in the year (`yy`) in
                `@start_date` is less than the month in `@end_date` or it is
                the same as the month in `@end_date`

                    MONTH(@start_date) > MONTH(@end_date))
                      OR (MONTH(@start_date) = MONTH(@end_date))

                and the day in `@start_date` is less than the day in
                `@end_date`.

                    AND DAY(@start_date) > DAY(@end_date)

                If both conditions are true, the query returns `1` (under a
                full year).  Otherwise it returns `0` (full year).
 *************************************************************************** */

SELECT @years = DATEDIFF(yy, @tmp_date, @end_date) - CASE
    WHEN (MONTH(@start_date) > MONTH(@end_date))
      OR (
        MONTH(@start_date) = MONTH(@end_date)
        AND DAY(@start_date) > DAY(@end_date)
        )
      THEN 1
    ELSE 0
    END;


/* ****************************************************************************
         9.3.9. We add the value of `@years` (`yy`) to `@tmp_date` returning 1
                or 0.
 *************************************************************************** */

SELECT @tmp_date = DATEADD(yy, @years, @tmp_date);


/* ****************************************************************************
         9.3.10. Below `@months` is assigned the difference of `@tmp_date` and
                 `@end_date` in months when the month (`m`) in `@start_date` is
                 less than the month in `@end_date` or it is the same as the
                 month in `@end_date`.

                    DAY(@start_date) > DAY(@end_date)

                 If the condition is true, the query returns `1` (under a full
                 month).  Otherwise it returns `0` (full month).
 *************************************************************************** */

SELECT @months = DATEDIFF(m, @tmp_date, @end_date) - CASE
    WHEN DAY(@start_date) > DAY(@end_date)
      THEN 1
    ELSE 0
    END;


/* ****************************************************************************
         9.3.11. We add the value of `@months` (`m`) to `@tmp_date` returning 1
                 or 0.
 *************************************************************************** */

SELECT @tmp_date = DATEADD(m, @months, @tmp_date);


/* ****************************************************************************
         9.3.12. Below `@days` is assigned the difference of `@tmp_date` and
                 `@end_date` in days.
 *************************************************************************** */

SELECT @days = DATEDIFF(d, @tmp_date, @end_date);


/* ****************************************************************************
         9.3.13. We finally display the values for `@years`, `@months` and
                 `@days`.

                    +----------------+-------+--------+------+
                    | Person's Name  | Years | Months | Days |
                    +----------------+-------+--------+------+
                    | Linus Torvalds | 51    | 11     | 2    |
                    +----------------+-------+--------+------+

         9.3.14. You can also use the script to calculate your age or any
                 difference between any two dates by changing the values in
                 section #9.3.4.

                 The value returned by `GETDATE()` when running this script was
                 2021/11/29 and the end result will change according to the
                 current date when the script is run.
 ************************************************************************** */

SELECT @persons_name AS 'Person''s Name',       -- two single quotes (``) to
                                                -- escape and show only one (`)
  @years AS 'Years',
  @months AS 'Months',
  @days AS 'Days';


/* ****************************************************************************
10. LAB #6
    Write a query without duplicate rows (`SELECT DISTINCT`)
    10.1. to get all shared values from tables `AP1.InvoiceLineItems` and
          `AP1.GLAccounts` (`INNER JOIN`),
    10.2. adding today's date as `TodaysDate` formatted as short date
    10.3. where `AP1.GLAccounts.AccountDescription` starts with `book`
          (`AP1.GLAccounts.AccountDescription LIKE('book%')`) and
          `AP1.InvoiceLineItems.InvoiceLineItemAmount` is at least 1000.00
          (inclusive) -- first condition composed of two conditions
    10.4. or where `AP1.GLAccounts.AccountDescription` contains `mail` and
          `AP1.InvoiceLineItems.InvoiceLineItemAmount` is no more than 100.00
          (inclusive) -- second condition composed of two conditions (second
          condition in parenthesis (OR secondary_codition1 AND
          secondary_condition2))
    10.5. ordered first by `AP1.GLAccounts.AccountDescription` and then by
          `AP1.InvoiceLineItems.InvoiceLineItemAmount`.
 ************************************************************************** */

SELECT DISTINCT AP1.InvoiceLineItems.InvoiceID,
  AP1.InvoiceLineItems.InvoiceSequence,
  AP1.InvoiceLineItems.AccountNo,
  InvoiceLineItemAmount,
  AP1.InvoiceLineItems.InvoiceLineItemDescription,
  -- AP1.GLAccounts.AccountNo AS Expr1,
  AP1.GLAccounts.AccountDescription
  /*,
  FORMAT(GETDATE(), 'd', 'en-us') AS TodaysDate*/
FROM AP1.InvoiceLineItems
INNER JOIN AP1.GLAccounts
  ON AP1.InvoiceLineItems.AccountNo = AP1.GLAccounts.AccountNo
```

```
WHERE
  (                                             -- 1. first block of two
                                                --    conditions that must be
                                                --    true
    AP1.GLAccounts.AccountDescription LIKE 'book%'
    AND AP1.InvoiceLineItems.InvoiceLineItemAmount >= 1000
    )
  OR                                            -- 2. `OR` to indicate that
                                                --    either the first block
                                                --    (above) or the second
                                                --    (below) must be true
  (                                             -- 3. second block of two
                                                --    conditions that must be
                                                --    true
    AP1.GLAccounts.AccountDescription LIKE '%mail%'
    AND AP1.InvoiceLineItems.InvoiceLineItemAmount <= 100
    )
ORDER BY AP1.GLAccounts.AccountDescription,
  AP1.InvoiceLineItems.InvoiceLineItemAmount,
  AP1.InvoiceLineItems.InvoiceID,
  AP1.InvoiceLineItems.InvoiceSequence,
  AP1.InvoiceLineItems.AccountNo,
  AP1.InvoiceLineItems.InvoiceLineItemDescription;


/* ****************************************************************************
  11. LAB #7
      11.1. Create database `labs`.
      11.2. Create schema `lab7` in database `labs`.
      11.3. Create table `my_family` in schema `lab7` with the following
            structure choosing the best file type for each column and assign
            `NOT NULL` to each.

                    row_id
                    person_fname
                    person_lname
                    relation

      11.4. Insert values accordingly.
      11.5. Modify table `my_family` to add a column `dob`.
      11.6. Update the table with data in `dob` (new values in an existing
            record in table `labs.lab7.my_family`).
      11.7. Change column `dob` to `NOT NULL`.
 *************************************************************************** */

CREATE DATABASE labs;                           -- 1. creating database `labs`
                                                --    1.1. run #1 (all `CREATE
                                                --         DATABASE` statements
                                                --         run together, but
                                                --         separately from
                                                --         other statements)

```

```sql
CREATE SCHEMA lab7;                             -- 2. creating schema `labs6`
                                                --    2.1. run #2 (each `CREATE
                                                --         SCHEMA` statement
                                                --         run separately)

CREATE TABLE lab7.my_family (                   -- 3. creating table
  row_id INT NOT NULL,                          --    `lab7.my_family`
  person_fname VARCHAR(25) NOT NULL,            --    3.1. run #3 (all `CREATE
  person_lname VARCHAR(25) NOT NULL,            --         TABLE` statement run
  relation VARCHAR(15) NOT NULL                 --         together, but
  );                                            --         separately from
                                                --         other statements)

INSERT INTO lab7.my_family                      -- 4. inserting new values into
VALUES (                                        --    table `lab7.my_family`
  1,                                            --    4.1. each row/record
  'John',                                       --         within a set of
  'Doe',                                        --         parenthesis followed
  'crazy uncle'                                 --         by a comma between
  ),                                            --         rows/records
  (                                             --    4.2. run #4 (all `INSERT`
  2,                                            --         statements run
  'Michael',                                    --         together, separately
  'Jones',                                      --         from other
  'cousin'                                      --         statements)
  ),
  (
  3,
  'Lucy',
  'Smith',
  'aunt'
  );

ALTER TABLE lab7.my_family                      -- 5. altering table
ADD dob DATE;                                   --    `lab7.my_family` to add
                                                --    column `dob` with data
                                                --    type `DATE`
                                                --    5.1. run #5 (all `ALTER`
                                                --         statements run
                                                --         together, separately
                                                --         from other
                                                --         statements)

UPDATE lab7.my_family                           -- 6. updating table
SET dob = '1970-01-01'                          --    `lab7.my_family` to pass
WHERE row_id = 1;                               --    a new values to column
                                                --    `dob` in the existing
UPDATE lab7.my_family                           --    table `lab7.my_family`
SET dob = '1980/05/09'                          --    6.1. run #6 (all `UPDATE`
WHERE row_id = 2;                               --         statements run
                                                --         together, separately
UPDATE lab7.my_family                           --         from other
```

```sql
SET dob = '1988/08/19'                          --         statements)
WHERE row_id = 3;

ALTER TABLE lab7.my_family                      -- 7. changing new column `dob`
ALTER COLUMN dob DATE NOT NULL;                 --    to `NOT NULL` as column
                                                --    now has values
                                                --    7.1. run #7 (this `ALTER`
                                                --         statement run after
                                                --         populating new
                                                --         column `dob`


/* ****************************************************************************
 https://folvera.commons.gc.cuny.edu/?p=1037
 *************************************************************************** */
```