```
/* ************************************************************************
          INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
                   SF25SQL6172025, 2025/11/17 - 2025/12/17
                 https://folvera.commons.gc.cuny.edu/?cat=35
 ************************************************************************

  SESSION #4 (2025/11/26): MANIPULATING DATA

  1. Using built-in functions for numeric values including aggregate functions
     and `GROUP BY`
  2. Using clauses `ORDER BY`, `CASE`, `WHERE` and operators
  3. Sub-queries
 ************************************************************************

 1. LAB #2
    Write a query without duplicate rows (`SELECT DISTINCT`)
    1.01. to call all columns from `AP1.Vendors` and `AP1.Invoices`, shared
          data only (`INNER JOIN`)
    1.02. to present `VendorPhone` in `(123) 456-7890` structure.
 ************************************************************************ */

SELECT DISTINCT                                  -- 1. to retrieve unique rows
  AP1.Vendors.VendorID,
  AP1.Vendors.VendorName,
  AP1.Vendors.VendorAddress1,
  AP1.Vendors.VendorAddress2,
  AP1.Vendors.VendorCity,
  AP1.Vendors.VendorState,
  AP1.Vendors.VendorZipCode,
  REPLACE(                                       -- 2. replacing `() -` from the
                                                 --    concatenation in #5
                                                 --    instead of a CASE clause
                                                 --    (logic block), which we
                                                 --    will cover later
    CONCAT (                                     -- 3. concatenating an opening
      '(',                                       --    parenthesis, first three
      LEFT(Vendors.VendorPhone, 3),              --    characters of
      ') ',                                      --    `VendorPhone`, a closing
      SUBSTRING(Vendors.VendorPhone, 4, 3),      --    parenthesis with a space,
      '-',                                       --    the substring of
      RIGHT(Vendors.VendorPhone, 4)              --    `VendorPhone` starting
      ), '() -', '') AS VendorPhone              --    from the fourth character
                                                 --    and taking 3, a hyphen
                                                 --    and the last four
                                                 --    characters of
                                                 --    `VendorPhone`
  AP1.Vendors.VendorContactLName,
  AP1.Vendors.VendorContactFName,
  AP1.Vendors.DefaultTermsID,
  AP1.Vendors.DefaultAccountNo,
  AP1.Invoices.InvoiceID,
  -- AP1.Invoices.VendorID AS Expr1,             -- 4. commenting out duplicate
                                                 --    field `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
FROM AP1.Vendors
INNER JOIN AP1.Invoices
  ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;


/* ************************************************************************
    As an alternative, we can use an alias (`AS`) for each table to make our
    code tidier and to avoid repeating the database and schema before each
```

```
    table (`<database>.<schema>.<table>` or `<schema>.<table>`) throughout the
    query.

                    `v` for `AP1.Vendors`
                    `i` for `AP1.Invoices`
 ************************************************************************** */

SELECT DISTINCT v.VendorID,
  v.VendorName,
  v.VendorAddress1,
  v.VendorAddress2,
  v.VendorCity,
  v.VendorState,
  v.VendorZipCode,
  REPLACE(CONCAT (
      '(',
      LEFT(v.VendorPhone, 3),
      ') ',
      SUBSTRING(v.VendorPhone, 4, 3),
      '-',
      RIGHT(v.VendorPhone, 4)
      ), '() -', '') AS VendorPhone v.VendorContactLName,
  v.VendorContactFName,
  v.DefaultTermsID,
  v.DefaultAccountNo,
  i.InvoiceID,
  -- i.VendorID AS Expr1,
  i.InvoiceNumber,
  i.InvoiceDate,
  i.InvoiceTotal,
  i.PaymentTotal,
  i.CreditTotal,
  i.TermsID,
  i.InvoiceDueDate,
  i.PaymentDate
FROM AP1.Vendors AS v
INNER JOIN AP1.Invoices AS i
  ON v.VendorID = i.VendorID;


/* ************************************************************************
 2. ``In mathematical sets, the null set, also called the empty set, is the set
    that does not contain anything. It is symbolized Ø or { }.  There is only
    one null set.  This is because there is logically only one way that a set
    can contain nothing.
    The null set makes it possible to explicitly define the results of
    operations on certain sets that would otherwise not be explicitly
    definable.  The intersection of two disjoint sets (two sets that contain no
    elements in common) is the null set. For example:
    {1, 3, 5, 7, 9, ...}∩{2, 4, 6, 8, 10, ...} = Ø                [∩ = U+2229]
                                                                  [Ø = U+2205]
    The null set provides a foundation for building a formal theory of numbers.
    In axiomatic mathematics, zero is defined as the cardinality of (that is,
    the number of elements in) the null set.  From this starting point,
    mathematicians can build the set of natural numbers, and from there, the
    sets of integers and rational numbers.``
    http://whatis.techtarget.com/definition/null-set

    As such, NULL refers to a memory allocation with no value -- not an empty
    space since the latter has a value of `CHAR(32)`.

    Note that concatenating any VARCHAR (ANSI-complaint accepting ASCII,
    UTF-8) or NVARCHAR (Microsoft proprietary data type, not ANSI-complaint
    accepting ASCII, UTF-8 and especially Unicode) field to a NULL (no value,
    not a blank character) field using `+` instead of using the `CONCAT()`
    function will return NULL.

    In the example below, we lose data when concatenating `VendorAddress1`
    and `VendorAddress2` in the `AP1.Vendors` table when using `+`.
```

```
139
140    3. ``An aggregate function performs a calculation on a set of values, and
141       returns a single value. Except for COUNT, aggregate functions ignore null
142       values.  Aggregate functions are often used with the GROUP BY clause of the
143       SELECT statement.``
144       https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql
145
146          3.01. In the example below, we search for the count of records from table
147                `AP1.Vendors` where column `VendorState` has a value of `NY` and
148                `NJ`.  Since a field (a single data allocation) cannot have two
149                values at the same time, the query returns no values.
150     ********************************************************************** */
151
152   SELECT COUNT(VendorState) AS CountVendorState
153   FROM AP1.Vendors
154   WHERE VendorState = 'NJ'
155     AND VendorState = 'NY';                          -- returns 0 (zero)
156
157
158   /* **********************************************************************
159       3.02. In the example below, we search for the count of records from table
160             `AP1.Vendors` where column `VendorState` has a value of `NY` or `NJ`.
161             In other words, the field can have either value.
162     ********************************************************************** */
163
164   SELECT COUNT(VendorState) AS CountVendorState
165   FROM AP1.Vendors
166   WHERE VendorState = 'NJ'
167     OR VendorState = 'NY';                           -- returns 7 (4 `NJ` & 3 `NY`)
168
169
170   /* **********************************************************************
171       3.03. In the example below, we search for the count of records from table
172             `AP1.Vendors` with `DISTINCT` values in column `VendorState` -- in
173             other words, the number of unique states.
174     ********************************************************************** */
175
176   SELECT COUNT(DISTINCT VendorState) AS CountVendorState
177   FROM AP1.Vendors;                                  -- returns 22
178
179
180   /* **********************************************************************
181       3.04. In the example below, we search for the count of records from table
182             `AP1.Vendors`.  We can use `*` (read as ``all``) since we are looking
183             for the number of all values -- in other words, of all records.
184     ********************************************************************** */
185
186   SELECT COUNT(*) AS CountOfRows
187   FROM AP1.Vendors;                                  -- returns 114
188
189
190   /* **********************************************************************
191       3.05. In the examples below, we retrieve the sum of values in column
192             `InvoiceTotal` (`SUM(InvoiceTotal)`), average value of column
193             `InvoiceTotal` (`AVG(InvoiceTotal)`), maximum value of column
194             `InvoiceTotal` (`MAX(InvoiceTotal)`) and minimum value of column
195             `InvoiceTotal` (`MIN(InvoiceTotal)`) from table `AP1.Invoices`.
196
197             Note that these values do not have commas as dividers (1,000) or
198             currency symbols.  If you need to include dividers, you would need to
199             use the `FORMAT()` function.
200
201             Also note that there is no need to use GROUP BY since all fields (in
202             this case the same field, `InvoiceTotal`) below are subject to
203             aggregate functions.
204     ********************************************************************** */
205
206   SELECT SUM(InvoiceTotal) AS InvoiceTotalSUM,       -- 1. returns 214290.51
207     AVG(InvoiceTotal) AS InvoiceTotalAVG,            -- 2. returns 1879.7413
```

```
  MAX(InvoiceTotal) AS InvoiceTotalMAX,          -- 3. returns 37966.19
  MIN(InvoiceTotal) AS InvoiceTotalMIN           -- 4. returns 6.00
FROM AP1.Invoices;
                                                 -- 5. no need for `GROUP BY`
                                                 --    since all fields are
                                                 --    affected by aggregate
                                                 --    functions


/* ************************************************************************
          If we need to use the `FORMAT()` function, we have to this after the
          aggregate function since the value is still numeric at this point.
          Once we format the number, the value is converted to a string, which
          can no longer be used for any math operation.
 ************************************************************************ */

SELECT FORMAT(SUM(InvoiceTotal), 'c', 'en-us')   -- 1. value formatted as
    AS InvoiceTotalSUM,                          --    currency (`c`) with
                                                 --    culture `en-us`
                                                 --    (English-US);  returns
                                                 --    $214,290.51
  FORMAT(AVG(InvoiceTotal), 'c', 'en-us')        -- 2. value formatted as
    AS InvoiceTotalAVG,                          --    currency (`c`) with
                                                 --    culture `en-us`
                                                 --    (English-US);  returns
                                                 --    $1,879.7413
  FORMAT(MAX(InvoiceTotal), 'c', 'en-us')        -- 3. value formatted as
    AS InvoiceTotalMAX,                          --    currency (`c`) with
                                                 --    culture `en-us`
                                                 --    (English-US);  returns
                                                 --    $37,966.19
  FORMAT(MIN(InvoiceTotal), 'c', 'en-us')        -- 4. value formatted as
    AS InvoiceTotalMIN                           --    currency (`c`) with
                                                 --    culture `en-us`
                                                 --    (English-US);  returns
                                                 --    $6.00
FROM AP1.Invoices;
                                                 -- 5. no need for `GROUP BY`
                                                 --    since all fields are
                                                 --    affected by aggregate
                                                 --    functions


/* ************************************************************************
    3.06. In the examples below, we search for the sum, average, maximum and
          minimum value of column `InvoiceTotal` from table `AP1.Invoices`
          respectively as (nested queries) sub-queries.
 ************************************************************************ */

SELECT InvoiceID,
  VendorID,
  InvoiceNumber,
  InvoiceDate,
  InvoiceTotal,
  (                                              -- 1. beginning of nested query
    SELECT                                       --    between parenthesis to
      MAX(InvoiceTotal)                          --    get `MAX(InvoiceTotal)`
    FROM AP1.Invoices                            --    from table `AP1.Invoices`
    ) AS InvoiceTotalMAX,                        --    with alias
                                                 --    `InvoiceTotalMAX` for
                                                 --    nested query
  (                                              -- 2. beginning of nested query
    SELECT                                       --    between parenthesis to
      MIN(InvoiceTotal)                          --    get `MIN(InvoiceTotal)`
    FROM AP1.Invoices                            --    from table `AP1.Invoices`
    ) AS InvoiceTotalMIN,                        --    with alias
                                                 --    `InvoiceTotalMIN` for
                                                 --    nested query
  ROUND                                          -- 3. rounding value of the sub
```

```
    (                                                   --    query to 2 decimal spaces
      (                                                 --    3.1. beginning of nested
        SELECT                                          --         query between
          AVG(InvoiceTotal)                             --         parenthesis to get
                                                        --         `AVG(InvoiceTotal)`
        FROM AP1.Invoices                               --         from table
        ),                                              --         `AP1.Invoices`
      2)                                                --    3.2. rounding the value
                                                        --         the nested query to
                                                        --         2 decimal spaces
    AS InvoiceTotalAVG,                                 -- 4. with alias
                                                        --    `InvoiceTotalAVG` for
                                                        --    nested query & `ROUND()`
    PaymentTotal,
    CreditTotal,
    TermsID,
    InvoiceDueDate,
    PaymentDate
  FROM AP1.Invoices
  ORDER BY VendorID,
    InvoiceTotal;


  /* ************************************************************************
      3.07. When using aggregate functions, we need to use `GROUP BY`.  Otherwise
            we would get the following error.

                      ``Msg 8120, Level 16, State 1, Line 2
                     Column `AP1.Invoices.InvoiceID` is invalid in the select
                     list because it is not contained in either an aggregate
                     function or the GROUP BY clause.``

            When using `GROUP BY`, we need to list each column that we are
            calling (from `InvoiceID` to `PaymentDate`) not affected by the
            aggregate function.

            Note that `AVG(InvoiceTotal)` returns the same value as
            `InvoiceTotal` since the average only affects a single value
            (`InvoiceTotal`) within a single row.
   ************************************************************************ */

  SELECT InvoiceID,
    VendorID,
    InvoiceNumber,
    InvoiceDate,
    InvoiceTotal,
    AVG(InvoiceTotal) AS InvoiceTotalAVG,           -- 1. aggregate function
                                                    --    `AVG()` only affecting
                                                    --    field `InvoiceTotal` and
                                                    --    returning the average of
                                                    --    the value per line
    PaymentTotal,
    CreditTotal,
    TermsID,
    InvoiceDueDate,
    PaymentDate
  FROM AP1.Invoices
  GROUP BY                                          -- 2. must use `GROUP BY`
    InvoiceID,                                      --    because of the aggregate
    VendorID,                                       --    function; all fields not
    InvoiceNumber,                                  --    affected by any aggregate
    InvoiceDate,                                    --    function to be listed;
    InvoiceTotal,                                   --    no exceptions to this
    PaymentTotal,                                   --    rule
    CreditTotal,
    TermsID,
    InvoiceDueDate,
    PaymentDate
  ORDER BY VendorID,                                -- 3. `ORDER BY` placed after
```

```
346     InvoiceTotal;                                   --    `GROUP BY`; no exceptions
347                                                     --    to this rule
348
349
350   /* ************************************************************************
351    https://folvera.commons.gc.cuny.edu/?p=1345
352    ************************************************************************ */
```