```
/* ************************************************************************
          INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
                    WS23SQL1001, 2023/04/03 to 2023/05/03
                  https://folvera.commons.gc.cuny.edu/?cat=33
 ************************************************************************

  SESSION #4 (2023/04/12): 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
```

```
53                                                  --    field `VendorID`
54    AP1.Invoices.InvoiceNumber,
55    AP1.Invoices.InvoiceDate,
56    AP1.Invoices.InvoiceTotal,
57    AP1.Invoices.PaymentTotal,
58    AP1.Invoices.CreditTotal,
59    AP1.Invoices.TermsID,
60    AP1.Invoices.InvoiceDueDate,
61    AP1.Invoices.PaymentDate
62  FROM AP1.Vendors
63  INNER JOIN AP1.Invoices
64    ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;
65
66
67  /* ********************************************************************************
68      As an alternative, we can use an alias for table.
69
70                      `v` for `AP1.Vendors`
71                      `i` for `AP1.Invoices`
72   ********************************************************************************* */
73
74  SELECT DISTINCT v.VendorID,
75    v.VendorName,
76    v.VendorAddress1,
77    v.VendorAddress2,
78    v.VendorCity,
79    v.VendorState,
80    v.VendorZipCode,
81    REPLACE(CONCAT (
82        '(',
83        LEFT(v.VendorPhone, 3),
84        ') ',
85        SUBSTRING(v.VendorPhone, 4, 3),
86        '-',
87        RIGHT(v.VendorPhone, 4)
88        ), '() -', '') AS VendorPhone v.VendorContactLName,
89    v.VendorContactFName,
90    v.DefaultTermsID,
91    v.DefaultAccountNo,
92    i.InvoiceID,
93    -- i.VendorID AS Expr1,
94    i.InvoiceNumber,
95    i.InvoiceDate,
96    i.InvoiceTotal,
97    i.PaymentTotal,
98    i.CreditTotal,
99    i.TermsID,
100   i.InvoiceDueDate,
101   i.PaymentDate
102 FROM AP1.Vendors AS v
103 INNER JOIN AP1.Invoices AS i
104   ON v.VendorID = i.VendorID;
```

```
105
106
107 /* ***************************************************************************
108  2. ``In mathematical sets, the null set, also called the empty set, is the set
109     that does not contain anything. It is symbolized ∅ or { }.  There is only
110     one null set.  This is because there is logically only one way that a set
111     can contain nothing.
112     The null set makes it possible to explicitly define the results of
113     operations on certain sets that would otherwise not be explicitly
114     definable.  The intersection of two disjoint sets (two sets that contain no
115     elements in common) is the null set. For example:
116     {1, 3, 5, 7, 9, ...}∩{2, 4, 6, 8, 10, ...} = ∅                [∩ = U+2229]
117                                                                   [∅ = U+2205]
118     The null set provides a foundation for building a formal theory of numbers.
119     In axiomatic mathematics, zero is defined as the cardinality of (that is,
120     the number of elements in) the null set.  From this starting point,
121     mathematicians can build the set of natural numbers, and from there, the
122     sets of integers and rational numbers.``
123     http://whatis.techtarget.com/definition/null-set
124
125     As such, NULL refers to a memory allocation with no value -- not an empty
126     space since the latter has a value of `CHAR(32)`.
127
128     Note that concatenating any VARCHAR (ANSI-complaint accepting ASCII,
129     UTF-8) or NVARCHAR (Microsoft proprietary data type, not ANSI-complaint
130     accepting ASCII, UTF-8 and especially Unicode) field to a NULL (no value,
131     not a blank character) field using `+` instead of using the `CONCAT()`
132     function will return NULL.
133
134     In the example below, we lose data when concatenating `VendorAddress1`
135     and `VendorAddress2` in the `AP1.Vendors` table when using `+`.
136
137  3. ``An aggregate function performs a calculation on a set of values, and
138     returns a single value. Except for COUNT, aggregate functions ignore null
139     values.  Aggregate functions are often used with the GROUP BY clause of the
140     SELECT statement.``
141     https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-
        transact-sql
142
143     3.01. In the example below, we search for the count of records from table
144           `AP1.Vendors` where column `VendorState` has a value of `NY` and
145           `NJ`.  Since a field (a single data allocation) cannot have two
146           values at the same time, the query returns no values.
147  ************************************************************************** */
148
149 SELECT COUNT(VendorState) AS CountVendorState
150 FROM AP1.Vendors
151 WHERE VendorState = 'NJ'
152   AND VendorState = 'NY';                        -- returns 0 (zero)
153
154
155 /* ***************************************************************************
```

```
156      3.02. In the example below, we search for the count of records from table
157            `AP1.Vendors` where column `VendorState` has a value of `NY` or `NJ`.
158            In other words, the field can have either value.
159   ********************************************************************** */
160
161  SELECT COUNT(VendorState) AS CountVendorState
162  FROM AP1.Vendors
163  WHERE VendorState = 'NJ'
164    OR VendorState = 'NY';                          -- returns 7 (4 `NJ` & 3 `NY`)
165
166
167  /* **********************************************************************
168      3.03. In the example below, we search for the count of records from table
169            `AP1.Vendors` with `DISTINCT` values in column `VendorState` -- in
170            other words, the number of unique states.
171   ********************************************************************** */
172
173  SELECT COUNT(DISTINCT VendorState) AS CountVendorState
174  FROM AP1.Vendors;                                 -- returns 22
175
176
177  /* **********************************************************************
178      3.04. In the example below, we search for the count of records from table
179            `AP1.Vendors`.  We can use `*` (read as ``all``) since we are looking
180            for the number of all values -- in other words, of all records.
181   ********************************************************************** */
182
183  SELECT COUNT(*) AS CountOfRows
184  FROM AP1.Vendors;                                 -- returns 114
185
186
187  /* **********************************************************************
188      3.05. In the examples below, we retrieve the sum of values in column
189            `InvoiceTotal` (`SUM(InvoiceTotal)`), average value of column
190            `InvoiceTotal` (`AVG(InvoiceTotal)`), maximum value of column
191            `InvoiceTotal` (`MAX(InvoiceTotal)`) and minimum value of column
192            `InvoiceTotal` (`MIN(InvoiceTotal)`) from table `AP1.Invoices`.
193
194            Note that these values do not have commas as dividers (1,000) or
195            currency symbols.  If you need to include dividers, you would need to
196            use the `FORMAT()` function.
197
198            Also note that there is no need to use GROUP BY since all fields (in
199            this case the same field, `InvoiceTotal`) below are subject to
200            aggregate functions.
201   ********************************************************************** */
202
203  SELECT SUM(InvoiceTotal) AS InvoiceTotalSUM,      -- 1. returns 214290.51
204    AVG(InvoiceTotal) AS InvoiceTotalAVG,           -- 2. returns 1879.7413
205    MAX(InvoiceTotal) AS InvoiceTotalMAX,           -- 3. returns 37966.19
206    MIN(InvoiceTotal) AS InvoiceTotalMIN            -- 4. returns 6.00
207  FROM AP1.Invoices;
```

```
208                                             -- 5. no need for `GROUP BY`
209                                             --    since all fields are
210                                             --    affected by aggregate
211                                             --    functions
212
213
214 /* ***************************************************************************
215           If we need to use the `FORMAT()` function, we have to this after the
216           aggregate function since the value is still numeric at this point.
217           Once we format the number, the value is converted to a string, which
218           can no longer be used for any math operation.
219  *************************************************************************** */
220
221 SELECT FORMAT(SUM(InvoiceTotal), 'c', 'en-us')  -- 1. value formatted as
222     AS InvoiceTotalSUM,                         --    currency (`c`) with
223                                                 --    culture `en-us`
224                                                 --    (English-US);  returns
225                                                 --    $214,290.51
226   FORMAT(AVG(InvoiceTotal), 'c', 'en-us')       -- 2. value formatted as
227     AS InvoiceTotalAVG,                         --    currency (`c`) with
228                                                 --    culture `en-us`
229                                                 --    (English-US);  returns
230                                                 --    $1,879.7413
231   FORMAT(MAX(InvoiceTotal), 'c', 'en-us')       -- 3. value formatted as
232     AS InvoiceTotalMAX,                         --    currency (`c`) with
233                                                 --    culture `en-us`
234                                                 --    (English-US);  returns
235                                                 --    $37,966.19
236   FORMAT(MIN(InvoiceTotal), 'c', 'en-us')       -- 4. value formatted as
237     AS InvoiceTotalMIN                          --    currency (`c`) with
238                                                 --    culture `en-us`
239                                                 --    (English-US);  returns
240                                                 --    $6.00
241 FROM AP1.Invoices;
242                                                 -- 5. no need for `GROUP BY`
243                                                 --    since all fields are
244                                                 --    affected by aggregate
245                                                 --    functions
246
247
248 /* ***************************************************************************
249     3.06. In the examples below, we search for the sum, average, maximum and
250           minimum value of column `InvoiceTotal` from table `AP1.Invoices`
251           respectively as (nested queries) sub-queries.
252  *************************************************************************** */
253
254 SELECT InvoiceID,
255   VendorID,
256   InvoiceNumber,
257   InvoiceDate,
258   InvoiceTotal,
259   (                                             -- 1. beginning of nested query
```

```
260     SELECT                                  --    between parenthesis to
261       MAX(InvoiceTotal)                     --    get `MAX(InvoiceTotal)`
262     FROM AP1.Invoices                       --    from table `AP1.Invoices`
263     ) AS InvoiceTotalMAX,                   --    with alias
264                                             --    `InvoiceTotalMAX` for
265                                             --    nested query
266   (                                         -- 2. beginning of nested query
267     SELECT                                  --    between parenthesis to
268       MIN(InvoiceTotal)                     --    get `MIN(InvoiceTotal)`
269     FROM AP1.Invoices                       --    from table `AP1.Invoices`
270     ) AS InvoiceTotalMIN,                   --    with alias
271                                             --    `InvoiceTotalMIN` for
272                                             --    nested query
273   ROUND                                     -- 3. rounding value of the sub
274   (                                         --    query to 2 decimal spaces
275     (                                       --    3.1. beginning of nested
276       SELECT                                --         query between
277         AVG(InvoiceTotal)                   --         parenthesis to get
278                                             --         `AVG(InvoiceTotal)`
279       FROM AP1.Invoices                     --         from table
280       ),                                    --         `AP1.Invoices`
281     2)                                      --    3.2. rounding the value
282                                             --         the nested query to
283                                             --         2 decimal spaces
284   AS InvoiceTotalAVG,                       -- 4. with alias
285                                             --    `InvoiceTotalAVG` for
286                                             --    nested query & `ROUND()`
287   PaymentTotal,
288   CreditTotal,
289   TermsID,
290   InvoiceDueDate,
291   PaymentDate
292 FROM AP1.Invoices
293 ORDER BY VendorID,
294   InvoiceTotal;
295
296
297 /* ********************************************************************************
298     3.07. When using aggregate functions, we need to use `GROUP BY`.  Otherwise
299           we would get the following error.
300
301                     ``Msg 8120, Level 16, State 1, Line 2
302                     Column `AP1.Invoices.InvoiceID` is invalid in the select
303                     list because it is not contained in either an aggregate
304                     function or the GROUP BY clause.``
305
306           When using `GROUP BY`, we need to list each column that we are
307           calling (from `InvoiceID` to `PaymentDate`) not affected by the
308           aggregate function.
309
310           Note that `AVG(InvoiceTotal)` returns the same value as
311           `InvoiceTotal` since the average only affects a single value
```

```
312           (`InvoiceTotal`) within a single row.
313   ************************************************************************ */
314
315  SELECT InvoiceID,
316    VendorID,
317    InvoiceNumber,
318    InvoiceDate,
319    InvoiceTotal,
320    AVG(InvoiceTotal) AS InvoiceTotalAVG,          -- 1. aggregate function
321                                                   --    `AVG()` only affecting
322                                                   --    field `InvoiceTotal` and
323                                                   --    returning the average of
324                                                   --    the value per line
325    PaymentTotal,
326    CreditTotal,
327    TermsID,
328    InvoiceDueDate,
329    PaymentDate
330  FROM AP1.Invoices
331  GROUP BY                                        -- 2. must use `GROUP BY`
332    InvoiceID,                                    --    because of the aggregate
333    VendorID,                                     --    function; all fields not
334    InvoiceNumber,                                --    affected by any aggregate
335    InvoiceDate,                                  --    function to be listed;
336    InvoiceTotal,                                 --    no exceptions to this
337    PaymentTotal,                                 --    rule
338    CreditTotal,
339    TermsID,
340    InvoiceDueDate,
341    PaymentDate
342  ORDER BY VendorID,                              -- 3. `ORDER BY` placed after
343    InvoiceTotal;                                 --    `GROUP BY`; no exceptions
344                                                  --    to this rule
345
346
347  /* ************************************************************************
348   https://folvera.commons.gc.cuny.edu/?p=1219
349   ************************************************************************ */
```