

```

1  /******
2  ..... INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
3  ..... SF25SQL6172025, 2025/11/17 - 2025/12/17
4  ..... https://folvera.commons.gc.cuny.edu/?cat=35
5  .....
6
7  -- SESSION #4 (2025/11/26): MANIPULATING DATA
8
9  -- 1. Using built-in functions for numeric values including aggregate functions
10 -- and `GROUP BY`
11 -- 2. Using clauses `ORDER BY`, `CASE`, `WHERE` and operators
12 -- 3. Sub-queries
13 .....
14
15 -- 1. LAB #2
16 -- Write a query without duplicate rows (`SELECT DISTINCT`)
17 -- 1.01. to call all columns from `AP1.Vendors` and `AP1.Invoices`, shared
18 -- data only (`INNER JOIN`)
19 -- 1.02. to present `VendorPhone` in `(123) 456-7890` structure.
20 ..... */
21
22 SELECT DISTINCT ..... -- 1. to retrieve unique rows
23   AP1.Vendors.VendorID,
24   AP1.Vendors.VendorName,
25   AP1.Vendors.VendorAddress1,
26   AP1.Vendors.VendorAddress2,
27   AP1.Vendors.VendorCity,
28   AP1.Vendors.VendorState,
29   AP1.Vendors.VendorZipCode,
30   REPLACE ( ..... -- 2. replacing `()` - ` ` from the
31   ..... -- concatenation in #5
32   ..... -- instead of a CASE clause
33   ..... -- (logic block), which we
34   ..... -- will cover later
35   CONCAT ( ..... -- 3. concatenating an opening
36   ..... -- parenthesis, first three
37   ..... -- characters of
38   ..... -- `VendorPhone`, a closing
39   ..... -- parenthesis with a space,
40   ..... -- the substring of
41   ..... -- `VendorPhone` starting
42   ..... -- from the fourth character
43   ..... -- and taking 3, a hyphen
44   ..... -- and the last four
45   ..... -- characters of
46   ..... -- `VendorPhone`
47   AP1.Vendors.VendorContactLName,
48   AP1.Vendors.VendorContactFName,
49   AP1.Vendors.DefaultTermsID,
50   AP1.Vendors.DefaultAccountNo,
51   AP1.Invoices.InvoiceID,
52   -- 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 (`AS`) for each table to make our
69 -- code tidier and to avoid repeating the database and schema before each

```

```

70  SELECT `<database>.<schema>.<table>` or `<schema>.<table>`) throughout the
71  query.
72
73  .....`v` for `AP1.Vendors`
74  .....`i` for `AP1.Invoices`
75  .....*/
76
77  SELECT DISTINCT v VendorID,
78     v VendorName,
79     v VendorAddress1,
80     v VendorAddress2,
81     v VendorCity,
82     v VendorState,
83     v VendorZipCode,
84     REPLACE(CONCAT(
85         '(',
86         LEFT(v VendorPhone, 3),
87         ') - ',
88         SUBSTRING(v VendorPhone, 4, 3),
89         '- ',
90         RIGHT(v VendorPhone, 4)
91         ), '() - ', '-') AS VendorPhone v VendorContactLName,
92     v VendorContactFName,
93     v DefaultTermsID,
94     v DefaultAccountNo,
95     i InvoiceID,
96     -- i VendorID AS Expr1,
97     i InvoiceNumber,
98     i InvoiceDate,
99     i InvoiceTotal,
100    i PaymentTotal,
101    i CreditTotal,
102    i TermsID,
103    i InvoiceDueDate,
104    i PaymentDate
105  FROM AP1.Vendors AS v
106  INNER JOIN AP1.Invoices AS i
107    ON v VendorID = i VendorID;
108
109
110  /*.....
111  2. ``In mathematical sets, the null set, also called the empty set, is the set
112  that does not contain anything. It is symbolized  $\emptyset$  or  $\{\}$ . There is only
113  one null set. This is because there is logically only one way that a set
114  can contain nothing.
115  The null set makes it possible to explicitly define the results of
116  operations on certain sets that would otherwise not be explicitly
117  definable. The intersection of two disjoint sets (two sets that contain no
118  elements in common) is the null set. For example:
119   $\{1, 3, 5, 7, 9, \dots\} \cap \{2, 4, 6, 8, 10, \dots\} = \emptyset$  ..... [0 = U+2229]
120  ..... [0 = U+2205]
121  The null set provides a foundation for building a formal theory of numbers.
122  In axiomatic mathematics, zero is defined as the cardinality of (that is,
123  the number of elements in) the null set. From this starting point,
124  mathematicians can build the set of natural numbers, and from there, the
125  sets of integers and rational numbers.``
126  http://whatis.techtarget.com/definition/null-set
127
128  As such, NULL refers to a memory allocation with no value -- not an empty
129  space since the latter has a value of `CHAR(32)`.
130
131  Note that concatenating any VARCHAR (ANSI-complaint accepting ASCII,
132  UTF-8) or NVARCHAR (Microsoft proprietary data type, not ANSI-complaint
133  accepting ASCII, UTF-8 and especially Unicode) field to a NULL (no value,
134  not a blank character) field using `+` instead of using the `CONCAT()`
135  function will return NULL.
136
137  In the example below, we lose data when concatenating `VendorAddress1`
138  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

```

```

2008      MAX(InvoiceTotal) AS InvoiceTotalMAX,      -- 3. returns 37966.19
2009      MIN(InvoiceTotal) AS InvoiceTotalMIN        -- 4. returns 6.00
2010 FROM AP1.Invoices;
2011
2012
2013
2014
2015
2016
2017
2018 /* *****
2019      If we need to use the `FORMAT()` function, we have to this after the
2020      aggregate function since the value is still numeric at this point.
2021      Once we format the number, the value is converted to a string, which
2022      can no longer be used for any math operation.
2023      ***** */
2024
2025 SELECT FORMAT(SUM(InvoiceTotal), 'c', 'en-us') -- 1. value formatted as
2026      AS InvoiceTotalSUM,                      -- currency (`c`) with
2027                                              -- culture `en-us`
2028                                              -- (English-US); returns
2029                                              -- $214,290.51
2030      FORMAT(AVG(InvoiceTotal), 'c', 'en-us')  -- 2. value formatted as
2031      AS InvoiceTotalAVG,                      -- currency (`c`) with
2032                                              -- culture `en-us`
2033                                              -- (English-US); returns
2034                                              -- $1,879.7413
2035      FORMAT(MAX(InvoiceTotal), 'c', 'en-us')  -- 3. value formatted as
2036      AS InvoiceTotalMAX,                      -- currency (`c`) with
2037                                              -- culture `en-us`
2038                                              -- (English-US); returns
2039                                              -- $37,966.19
2040      FORMAT(MIN(InvoiceTotal), 'c', 'en-us')  -- 4. value formatted as
2041      AS InvoiceTotalMIN                      -- currency (`c`) with
2042                                              -- culture `en-us`
2043                                              -- (English-US); returns
2044                                              -- $6.00
2045 FROM AP1.Invoices;
2046
2047
2048
2049
2050
2051
2052 /* *****
2053      3.06. In the examples below, we search for the sum, average, maximum and
2054      minimum value of column `InvoiceTotal` from table `AP1.Invoices`
2055      respectively as (nested queries) sub-queries.
2056      ***** */
2057
2058 SELECT InvoiceID,
2059      VendorID,
2060      InvoiceNumber,
2061      InvoiceDate,
2062      InvoiceTotal,
2063      (
2064          SELECT
2065              MAX(InvoiceTotal)
2066          FROM AP1.Invoices
2067          ) AS InvoiceTotalMAX,
2068      (
2069          SELECT
2070              MIN(InvoiceTotal)
2071          FROM AP1.Invoices
2072          ) AS InvoiceTotalMIN,
2073      ROUND
2074
2075
2076

```

```

277 ( ..... -- query to 2 decimal spaces
278 ( ..... -- 3.1. beginning of nested
279 SELECT ..... -- query between
280 AVG(InvoiceTotal) ..... -- parenthesis to get
281 ..... `AVG(InvoiceTotal)`
282 FROM AP1.Invoices ..... -- from table
283 ), ..... -- `AP1.Invoices`
284 2) ..... -- 3.2. rounding the value
285 ..... -- the nested query to
286 ..... -- 2 decimal spaces
287 AS InvoiceTotalAVG, ..... -- 4. with alias
288 ..... -- `InvoiceTotalAVG` for
289 ..... -- nested query & `ROUND()`
290 PaymentTotal,
291 CreditTotal,
292 TermsID,
293 InvoiceDueDate,
294 PaymentDate
295 FROM AP1.Invoices
296 ORDER BY VendorID,
297 InvoiceTotal;
298
299
300 /*.*****
301 3.07. When using aggregate functions, we need to use `GROUP BY`. Otherwise
302 we would get the following error.
303
304 `Msg 8120, Level 16, State 1, Line 2
305 Column `AP1.Invoices.InvoiceID` is invalid in the select
306 list because it is not contained in either an aggregate
307 function or the GROUP BY clause.`
308
309 When using `GROUP BY`, we need to list each column that we are
310 calling (from `InvoiceID` to `PaymentDate`) not affected by the
311 aggregate function.
312
313 Note that `AVG(InvoiceTotal)` returns the same value as
314 `InvoiceTotal` since the average only affects a single value
315 (`InvoiceTotal`) within a single row.
316 .******/
317
318 SELECT InvoiceID,
319 VendorID,
320 InvoiceNumber,
321 InvoiceDate,
322 InvoiceTotal,
323 AVG(InvoiceTotal) AS InvoiceTotalAVG, ..... -- 1. aggregate function
324 ..... -- `AVG()` only affecting
325 ..... -- field `InvoiceTotal` and
326 ..... -- returning the average of
327 ..... -- the value per line
328 PaymentTotal,
329 CreditTotal,
330 TermsID,
331 InvoiceDueDate,
332 PaymentDate
333 FROM AP1.Invoices
334 GROUP BY ..... -- 2. must use `GROUP BY`
335 InvoiceID, ..... -- because of the aggregate
336 VendorID, ..... -- function; all fields not
337 InvoiceNumber, ..... -- affected by any aggregate
338 InvoiceDate, ..... -- function to be listed;
339 InvoiceTotal, ..... -- no exceptions to this
340 PaymentTotal, ..... -- rule
341 CreditTotal,
342 TermsID,
343 InvoiceDueDate,
344 PaymentDate
345 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 ******/
```