

```

1  /* *****
2      INTRODUCTION TO STRUCTURED QUERY LANGUAGE FOR DATA ANALYTICS
3          WS24SQL10001, 2024/03/11 - 2024/04/10
4          https://folvera.commons.gc.cuny.edu/?cat=34
5  *****
6
7  SESSION #7 (2024/04/01): CREATING DATABASE OBJECTS
8
9  1. Understanding functions `CONVERT()`, `CAST()`, `DAY()`, `MONTH()`,
10     `YEAR()` and `GETDATE()`
11  2. Creating, dropping and altering views
12  *****
13
14  1. LAB #5
15     Write a query
16     1.01. to call all columns and values shared by tables `AP1.ContactUpdates`
17           and `AP1.Vendors` (`INNER JOIN`),
18     1.02. retrieving only rows with `AP1.Vendors.VendorState` with values of
19           `NY`, `NJ` and `CA`
20     1.03. using `CASE` to replace `NY` to `New York`, `NJ` to `New Jersey`,
21           `CA` to `California` and any other value to `Other`
22     1.04. ordered first by `AP1.Vendors.VendorState` and then by
23           `AP1.Vendors.VendorID`.
24     BONUS: Make a view in schema `lab05` in database `labs`.
25  ***** */
26
27  SELECT AP1.ContactUpdates.VendorID,
28         AP1.ContactUpdates.LastName,
29         AP1.ContactUpdates.FirstName,
30         -- AP1.Vendors.VendorID AS Expr1,           -- 1. duplicate column name
31                                                -- commented out
32         AP1.Vendors.VendorName,
33         AP1.Vendors.VendorAddress1,
34         AP1.Vendors.VendorAddress2,
35         AP1.Vendors.VendorCity,
36         CASE                                       -- 2. beginning of logic
37             WHEN AP1.Vendors.VendorState = 'NY'   -- 2.1. checking for value
38                 THEN 'New York'                  -- `NY` and return
39                                                -- value `New York`
40             WHEN AP1.Vendors.VendorState = 'NJ'   -- 2.2. checking for value
41                 THEN 'New Jersey'                -- `NY` and return
42                                                -- value `New Jersey`
43             WHEN AP1.Vendors.VendorState = 'CA'   -- 2.3. checking for value
44                 THEN 'California'                -- `NY` and return
45                                                -- value `California`
46             ELSE 'Other'                          -- 2.4. checking for other
47                                                -- values and return
48                                                -- value `Other`
49         END AS VendorState,
50         AP1.Vendors.VendorZipCode,
51         AP1.Vendors.VendorPhone,
52         AP1.Vendors.VendorContactLName,
53         AP1.Vendors.VendorContactFName,
54         AP1.Vendors.DefaultTermsID,
55         AP1.Vendors.DefaultAccountNo
56  FROM AP1.ContactUpdates
57  INNER JOIN AP1.Vendors
58         ON AP1.ContactUpdates.VendorID = AP1.Vendors.VendorID
59  WHERE AP1.Vendors.VendorState IN (             -- 3. indicating what values we
60         'NY',                                   -- query to return
61         'NJ',
62         'CA'
63     );
64
65
66  /* *****
67     At this point, we make a view in schema `lab05` in database `labs`
68     excluding the `ORDER BY` clause as it cannot be used when creating views.
69  ***** */

```

```

70
71 CREATE SCHEMA lab05; -- 1. creating schema `labs05`
72
73 CREATE VIEW lab05.ContactUpdatesVendorsVW -- 2. creating view
74 AS -- `ContactUpdatesVendorsVW`
75 SELECT WS24SQL10001.AP1.ContactUpdates.VendorID, -- in database `labs`
76 WS24SQL10001.AP1.ContactUpdates.LastName, -- hence calling database
77 WS24SQL10001.AP1.ContactUpdates.FirstName, -- `WS24SQL10001` for each
78 -- WS24SQL10001.AP1.Vendors.VendorID AS Expr1, -- table
79 WS24SQL10001.AP1.Vendors.VendorName,
80 WS24SQL10001.AP1.Vendors.VendorAddress1,
81 WS24SQL10001.AP1.Vendors.VendorAddress2,
82 WS24SQL10001.AP1.Vendors.VendorCity,
83 -- case to replace NY with New York...
84 CASE
85 WHEN WS24SQL10001.AP1.Vendors.VendorState = 'NY'
86 THEN 'New York'
87 WHEN WS24SQL10001.AP1.Vendors.VendorState = 'NJ'
88 THEN 'New Jersey'
89 WHEN WS24SQL10001.AP1.Vendors.VendorState = 'CA'
90 THEN 'California'
91 ELSE 'Other'
92 END AS VendorState,
93 WS24SQL10001.AP1.Vendors.VendorZipCode,
94 WS24SQL10001.AP1.Vendors.VendorPhone,
95 WS24SQL10001.AP1.Vendors.VendorContactLName,
96 WS24SQL10001.AP1.Vendors.VendorContactFName,
97 WS24SQL10001.AP1.Vendors.DefaultTermsID,
98 WS24SQL10001.AP1.Vendors.DefaultAccountNo
99 FROM WS24SQL10001.AP1.ContactUpdates
100 INNER JOIN WS24SQL10001.AP1.Vendors
101 ON WS24SQL10001.AP1.ContactUpdates.VendorID = WS24SQL10001.AP1.Vendors.VendorID
102 WHERE WS24SQL10001.AP1.Vendors.VendorState IN (
103 'NY',
104 'NJ',
105 'CA'
106 );
107
108
109 /* *****
110 As with previous examples, we can use an alias for each table, which
111 in this case, allows us to present neater code.
112
113 `c` for `WS24SQL10001.AP1.ContactUpdates`
114 `v` for `WS24SQL10001.AP1.Vendors`
115 ***** */
116
117 CREATE VIEW lab05.ContactUpdatesVendorsVW
118 AS
119 SELECT c.VendorID,
120 c.LastName,
121 c.FirstName,
122 -- v.VendorID AS Expr1,
123 v.VendorName,
124 v.VendorAddress1,
125 v.VendorAddress2,
126 v.VendorCity,
127 CASE
128 WHEN v.VendorState = 'NY'
129 THEN 'New York'
130 WHEN v.VendorState = 'NJ'
131 THEN 'New Jersey'
132 WHEN v.VendorState = 'CA'
133 THEN 'California'
134 ELSE 'Other'
135 END AS VendorState,
136 v.VendorZipCode,
137 v.VendorPhone,
138 v.VendorContactLName,

```

```

139     v.VendorContactFName,
140     v.DefaultTermsID,
141     v.DefaultAccountNo
142 FROM WS24SQL10001.AP1.ContactUpdates AS c
143 INNER JOIN WS24SQL10001.AP1.Vendors AS v
144     ON c.VendorID = v.VendorID
145 WHERE v.VendorState IN (
146     'NY',
147     'NJ',
148     'CA'
149 );
150
151
152 /* *****
153 2. LAB #6
154 Write a query without duplicate rows (`SELECT DISTINCT`)
155 2.01. to call all shared values from tables `AP1.Invoices` and `AP1.Terms`,
156 2.02. to format all dates as `yyyy-MM-dd` and currency as pounds sterling
157     (culture `en-gb`)
158 2.03. where `AP1.Invoices.PaymentTotal` is greater than the average value
159     of `AP1.Invoices.InvoiceTotal` (sub-query within the `WHERE` clause)
160     and `AP1.Invoices.PaymentDate` is not null.
161 HINT: `AVG(AP1.Invoices.InvoiceTotal)` FROM AP1.Invoices` for the nested
162     query value
163 BONUS: Make a view in schema `lab06` in database `labs` (already created in
164     lab #5).
165 ***** */
166
167 SELECT AP1.Invoices.InvoiceID,
168     AP1.Invoices.VendorID,
169     AP1.Invoices.InvoiceNumber,
170     FORMAT (AP1.Invoices.InvoiceDate,
171         'yyyy-MM-dd', 'en-gb')
172         AS InvoiceDate,
173     FORMAT (AP1.Invoices.InvoiceTotal,
174         'c', 'en-gb')
175         AS InvoiceTotal,
176     FORMAT (AP1.Invoices.PaymentTotal,
177         'c', 'en-gb')
178         AS PaymentTotal,
179     FORMAT (AP1.Invoices.CreditTotal,
180         'c', 'en-gb')
181         AS CreditTotal,
182     AP1.Invoices.TermsID,
183     FORMAT (AP1.Invoices.InvoiceDueDate,
184         'yyyy-MM-dd', 'en-gb')
185         AS InvoiceDueDate,
186     FORMAT (AP1.Invoices.PaymentDate,
187         'yyyy-MM-dd', 'en-gb')
188         AS PaymentDate,
189     AP1.Terms.TermsDescription,
190     AP1.Terms.TermsDueDays
191 FROM AP1.Invoices
192 INNER JOIN AP1.Terms
193     ON AP1.Invoices.TermsID = AP1.Terms.TermsID
194 WHERE (
195     AP1.Invoices.PaymentTotal > (
196         SELECT AVG(PaymentTotal)
197     )
198 )

```

```

-- 1. formatting column
-- `InvoiceDate` as
-- `yyyy-MM-dd` date with
-- culture `en-gb` using
-- alias `InvoiceDate`
-- 2. formatting column
-- `InvoiceTotal` as
-- `c` (currency) with
-- culture `en-gb` using
-- alias `InvoiceTotal`
-- 3. formatting column
-- as `c` (currency) with
-- culture `en-gb` using
-- alias `PaymentTotal`
-- 4. formatting column
-- as `c` (currency) with
-- culture `en-gb` using
-- alias `CreditTotal`
-- 5. formatting column
-- as `yyyy-MM-dd` date with
-- culture `en-gb` using
-- alias `InvoiceDueDate`
-- 6. formatting column
-- as `yyyy-MM-dd` date with
-- culture `en-gb` using
-- alias `PaymentDate`
-- 7. from table `AP1.Invoices`
-- using `INNER JOIN` to
-- retrieve all shared data
-- connecting both tables on
-- shared field `TermsID`
-- 8. where the value of
-- `PaymentTotal` is larger
-- than (`>`) the single
-- value of sub-query

```

```

208         FROM AP1.Invoices
209
210
211
212     )
213 )
214
215
216
217
218
219
220
221
222     AND AP1.Invoices.PaymentDate IS NOT NULL;
223
224
225
226
227 /* *****
228     At this point, we make a view in schema `lab06` in database `labs`
229     excluding the `ORDER BY` clause as it cannot be used when creating views.
230     ***** */
231
232 CREATE SCHEMA lab06;
233
234 CREATE VIEW lab06.InvoicesTermsVW
235 AS
236 SELECT WS24SQL10001.AP1.Invoices.InvoiceID,
237        WS24SQL10001.AP1.Invoices.VendorID,
238        WS24SQL10001.AP1.Invoices.InvoiceNumber,
239        FORMAT(WS24SQL10001.AP1.Invoices.InvoiceDate,
240              'yyyy-MM-dd', 'en-gb') AS InvoiceDate,
241        FORMAT(WS24SQL10001.AP1.Invoices.InvoiceTotal,
242              'c', 'en-gb') AS InvoiceTotal,
243        FORMAT(WS24SQL10001.AP1.Invoices.PaymentTotal,
244              'c', 'en-gb') AS PaymentTotal,
245        FORMAT(WS24SQL10001.AP1.Invoices.CreditTotal,
246              'c', 'en-gb') AS CreditTotal,
247        WS24SQL10001.AP1.Invoices.TermsID,
248        FORMAT(WS24SQL10001.AP1.Invoices.InvoiceDueDate,
249              'yyyy-MM-dd', 'en-gb') AS InvoiceDueDate,
250        FORMAT(WS24SQL10001.AP1.Invoices.PaymentDate,
251              'yyyy-MM-dd', 'en-gb') AS PaymentDate,
252        WS24SQL10001.AP1.Terms.TermsDescription,
253        WS24SQL10001.AP1.Terms.TermsDueDays
254 FROM WS24SQL10001.AP1.Invoices
255 INNER JOIN WS24SQL10001.AP1.Terms
256 ON WS24SQL10001.AP1.Invoices.TermsID = WS24SQL10001.AP1.Terms.TermsID
257 WHERE (
258     WS24SQL10001.AP1.Invoices.PaymentTotal > (
259         SELECT AVG(PaymentTotal)
260         FROM WS24SQL10001.AP1.Invoices
261     )
262 )
263 AND WS24SQL10001.AP1.Invoices.PaymentDate IS NOT NULL;
264
265
266 /* *****
267     As with previous examples, we can use an alias for each table, which
268     in this case, allows us to present neater code.
269
270     `i` for `WS24SQL10001.AP1.Invoices`
271     `t` for `WS24SQL10001.AP1.Terms`
272     ***** */
273
274 CREATE VIEW lab06.InvoicesTermsVW
275 AS
276 SELECT i.InvoiceID,

```

```

277 i.VendorID,
278 i.InvoiceNumber,
279 FORMAT(i.InvoiceDate, 'yyyy-MM-dd', 'en-gb') AS InvoiceDate,
280 FORMAT(i.InvoiceTotal, 'c', 'en-gb') AS InvoiceTotal,
281 FORMAT(i.PaymentTotal, 'c', 'en-gb') AS PaymentTotal,
282 FORMAT(i.CreditTotal, 'c', 'en-gb') AS CreditTotal,
283 i.TermsID,
284 FORMAT(i.InvoiceDueDate, 'yyyy-MM-dd', 'en-gb') AS InvoiceDueDate,
285 FORMAT(i.PaymentDate, 'yyyy-MM-dd', 'en-gb') AS PaymentDate,
286 t.TermsDescription,
287 t.TermsDueDays
288 FROM WS24SQL10001.AP1.Invoices AS i
289 INNER JOIN WS24SQL10001.AP1.Terms AS t
290 ON i.TermsID = t.TermsID
291 WHERE (
292     i.PaymentTotal > (
293         SELECT AVG(PaymentTotal)           -- no need for a table alias as
294         FROM WS24SQL10001.AP1.Invoices     -- the table is called once in
295     )                                       -- the sub-query
296 )
297 AND i.PaymentDate IS NOT NULL;
298
299

```

```

300 /* *****

```

- 301 3. As a quick review, SQL is the language to interact with a relational
302 database.
- 303 * to request data (`SELECT`) from database objects like databases,
304 schemata, tables and views
 - 305 * to create (`CREATE`) where to store data, database objects like
306 databases, schemata, tables including columns, etc. `
 - 307 * to modify (`ALTER`) database objects
 - 308 * to delete (`DROP`) database objects, automatic `COMMIT` in SQL Server
309 hence no `ROLLBACK` (no way to rescue the data or objects)
 - 310 * and to manipulate data either affecting the data or not (showing data).

```

311
312     CREATE obj_type object_name
313     [other_code]
314
315     DROP obj_type object_name
316     [other_code]
317
318     ALTER obj_type object_name
319     ALTER|ADD|DROP obj_type obj_name data_type [other_code]
320
321     INSERT INTO table_name
322     VALUES
323     (
324         field1 datatype1,
325         field2 datatype2
326         ...
327     )
328
329     DELETE FROM table_name
330     [other_code]
331
332     TRUNCATE TABLE table_name
333
334     UPDATE table_name
335     SET field = new_value
336

```

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

338 We can use functions to change the output of data as well as the data
339 itself, which we will see later in the course.

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

```

345 ***** */

```

```

346
347 SELECT FORMAT(InvoiceTotal, 'yyyy-MM-dd')           -- no culture (`en-us`) needed
348 FROM AP1.Invoices;                                 -- because of the custom format
349
350 SELECT FORMAT(InvoiceTotal,                          -- but good practice to include
351 'yyyy-MM-dd', 'en-us')                             -- the culture (`en-us`) even
352 FROM AP1.Invoices;                                 -- when overridden by custom
353                                                    -- a format like `yyyy-MM-dd`
354
355
356 /* *****
357 5. As mentioned several times, `FORMAT()` changes numeric values to strings.
358 We can also use `CONVERT()` to change ``an expression from a data type to
359 another data type`` -- in other words, numeric values to strings or vice
360 versa (https://techonthenet.com/sql\_server/functions/convert.php).
361
362         CONVERT(new_data_type, column)
363
364 `CONVERT()` does not change the currency sign or adds commas to divide
365 thousands or millions as `FORMAT()` does.
366
367 5.01. In the example below, we change the data type of `InvoiceTotal` to
368 VARCHAR(50) -- an allocation in RAM to hold a variable character
369 value with a maximum size of fifty (50) characters.
370 ***** */
371
372 SELECT CONVERT(VARCHAR(50), InvoiceTotal)           -- changing data type of column
373 AS InvoiceTotal                                   -- `InvoiceTotal` (`FLOAT`) to
374 FROM AP1.Invoices;                               -- `VARCHAR(50)`
375
376
377 /* *****
378 5.02. In the example below, we use `CONVERT()` to return the value of
379 `AP1.Invoices.InvoiceTotal` as a dollar amount concatenating the
380 dollar sign (`$`) at the beginning.
381 ***** */
382
383 SELECT CONCAT (
384     '$',
385     CONVERT(VARCHAR(50), InvoiceTotal)
386 ) AS InvoiceTotal
387 FROM AP1.Invoices;
388
389
390 /* *****
391 We could also use `CONVERT()` to return the value of
392 `AP1.Invoices.InvoiceTotal` as a dollar amount with `USD` rather than
393 the dollar sign (`$`).
394 ***** */
395
396 SELECT CONCAT (
397     'USD ',
398     CONVERT(VARCHAR(50), InvoiceTotal)
399 ) AS InvoiceTotal
400 FROM AP1.Invoices;
401
402
403 /* *****
404 Of course, if you are ``dressing up`` a numeric value like
405 `AP1.Invoices.InvoiceTotal` as currency, it is better to just use
406 `FORMAT()` to keep your code simple.
407 ***** */
408
409 SELECT FORMAT(InvoiceTotal, 'c', 'en-us') AS InvoiceTotal
410 FROM AP1.Invoices;
411
412
413 /* *****
414 5.03. In the example below, we use `CONVERT()` to change the data type of

```

```

415         `AP1.Invoices.InvoiceID` and `AP1.Invoices.VendorID` from FLOAT to
416         `VARCHAR(50)` before concatenating these values to a string.
417     *****/
418
419     SELECT CONCAT (
420
421         'Invoice ',
422         CONVERT(VARCHAR(3), InvoiceID),
423
424         ' from vendor ',
425         CONVERT(VARCHAR(3), VendorID)
426
427     ) AS InvoiceVendor
428
429     FROM AP1.Invoices;
430
431
432
433
434
435

```

```

436 /* *****/
437 6. We use the `WHERE` (https://techonthenet.com/sql\_server/where.php)
438 clause to filter the results from a SELECT, INSERT, UPDATE, or DELETE
439 statement.``
440

```

```

441     SELECT table1.field1, table1.field2 ...
442           table2.field1, table2.field2 ...
443     FROM table1
444           INNER|LEFT|RIGHT JOIN table2
445           ON table1.shared_field1 = table2.shared_field1
446           AND table1.shared_field2 = table2.shared_field2
447           ...
448     WHERE condition1
449           AND|OR condition2
450           ...
451

```

452 6.01. We use conditions in order to filter data.

```

453
454     AND           to test for two or more conditions
455                 https://techonthenet.com/sql\_server/and.php
456
457     OR           to test multiple conditions where records are returned when
458                 any one of the conditions are met
459                 https://techonthenet.com/sql\_server/or.php
460

```

461 6.02. We use operators to compare values.

```

462
463     =           equal to
464                 https://techonthenet.com/sql\_server/comparison\_operators.php
465
466     <>          not equal to
467                 https://techonthenet.com/sql\_server/comparison\_operators.php
468
469     !=          not equal to
470                 https://techonthenet.com/sql\_server/comparison\_operators.php
471
472     <           less than
473                 https://techonthenet.com/sql\_server/comparison\_operators.php
474
475     >           greater than
476                 https://techonthenet.com/sql\_server/comparison\_operators.php
477
478     <=          less than or equal to
479                 https://techonthenet.com/sql\_server/comparison\_operators.php
480
481     >=          greater than or equal to
482                 https://techonthenet.com/sql\_server/comparison\_operators.php
483

```

```

484      !>      not greater than (same as <=)
485              https://techonthenet.com/sql\_server/comparison\_operators.php
486
487      !<      not less than (same as >=)
488              https://techonthenet.com/sql\_server/comparison\_operators.php
489
490      LIKE    allows wild cards to be used in the WHERE clause of a
491              SELECT, INSERT, UPDATE, or DELETE statement [allowing] you
492              to perform pattern matching
493              https://techonthenet.com/sql\_server/like.php
494
495      IN      to help reduce the need to use multiple OR conditions in a
496              SELECT, INSERT, UPDATE, or DELETE statement
497              https://techonthenet.com/sql\_server/in.php
498
499      BETWEEN used to retrieve values within a range in a SELECT, INSERT,
500              UPDATE, or DELETE statement
501              https://techonthenet.com/sql\_server/between.php
502
503      IS NULL condition... used to test for a NULL no value
504              https://techonthenet.com/sql\_server/is\_null.php
505
506      NOT     to negate a condition in a SELECT, INSERT, UPDATE, or
507              DELETE statement
508              https://techonthenet.com/sql\_server/not.php
509
510          * NOT LIKE
511          * NOT IN
512          * NOT BETWEEN
513          * IS NOT NULL
514              https://techonthenet.com/sql\_server/is\_not\_null.php
515

```

6.03. 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

```

523      (
524          VendorCity = 'Fresno'
525          OR VendorCity = 'Sacramento'
526      )

```

before the outer condition.

```

529      ***** */

```

```

531  SELECT *
532  FROM AP1.Vendors
533  WHERE VendorState = 'CA'
534
535         AND (
536
537
538
539         VendorCity = 'Fresno'
540
541         OR VendorCity = 'Sacramento'
542
543         );

```

-- 1. inner criterion that must be true (satisfied)
-- 2. outer criterion that must be true composed of two sections where either could be true (satisfied)
-- 3.1. first criteria that could be met
-- 3.2. second criteria that could be met

```

546  /* ***** */
547      6.04. In the example below, we retrieve all values from table `AP1.Vendors`
548      where `VendorState` is not (<> or !=) `NY`.
549      ***** */

```

```

551  SELECT *
552  FROM AP1.Vendors

```

-- can also be written as


```

553 WHERE VendorState <> 'NY'; -- `VendorState` != `NY``
554
555
556 /* *****
557 6.05. In the example below, we retrieve all values from table `AP1.Vendors`
558 where `VendorState` is either `DC` or `IA`.
559 ***** */
560
561 SELECT *
562 FROM AP1.Vendors
563 WHERE VendorState = 'DC' -- checking if either criterion
564 OR VendorState = 'IA'; -- is true
565
566
567 /* *****
568 6.06. In the example below, we retrieve all values from table `AP1.Vendors`
569 where `VendorAddress2` is NULL (no-value) using `NOT` as it negates
570 operators `LIKE` as `NOT LIKE`, `IN` as `NOT IN`, `BETWEEN` as
571 `NOT BETWEEN` and `IS NULL` as `IS NOT NULL`.
572 ***** */
573
574 SELECT *
575 FROM AP1.Vendors
576 WHERE VendorAddress2 IS NULL; -- asking for no-value
577
578
579 /* *****
580 6.07. In the example below, we retrieve all values from table `AP1.Vendors`
581 where `VendorAddress2` is not NULL (not a no-value). Refer to
582 https://techonthenet.com/sql\_server/is\_not\_null.php for more
583 information.
584 ***** */
585
586 SELECT *
587 FROM AP1.Vendors
588 WHERE VendorAddress2 IS NOT NULL; -- asking for not `NOT NULL`
589 -- (no no-value)
590
591
592 /* *****
593 6.08. In the example below, we rewrite #6.3 in a cleaner fashion to
594 retrieve all values from table `AP1.Vendors` where `VendorState` is
595 equal to `CA` and `VendorCity` could either be `Fresno` or
596 `Sacramento`. We use operator `IN`
597 (https://techonthenet.com/sql\_server/in.php) to specify the list of
598 values that can be true for `VendorCity`.
599 ***** */
600
601 SELECT *
602 FROM AP1.Vendors
603 WHERE VendorState = 'CA' -- 1. first condition as in
604 -- original example
605 AND VendorCity IN ( -- 2. second condition using
606 'Fresno', -- `IN` to list all possible
607 'Sacramento' -- values that can be true
608 ); -- (satisfied)
609
610
611 /* *****
612 6.09. In the example below, we retrieve all values from table `AP1.Vendors`
613 where `VendorState` could either be `CA` or `NJ` and `VendorCity`
614 could either be `Fresno` or `Sacramento`.
615
616 This query looks for the combination of
617
618 `CA` and `Fresno` (true)
619 `CA` and `Sacramento` (true)
620
621 as well as

```

```

622
623         `NJ` and `Fresno`      (false)
624         `NJ` and `Sacramento` (false)
625
626     The query only returns only the first set of values since we do not
627     have any records where `VendorCity` is `NJ` and VendorCity` is either
628     `Fresno` or `Sacramento`.
629     ***** */
630
631 SELECT *
632 FROM AP1.Vendors
633 WHERE (
634     VendorState IN (
635         'CA',
636         'NJ'
637     )
638     AND VendorCity IN (
639         'Fresno',
640         'Sacramento'
641     )
642 )
643 ORDER BY VendorState,
644     VendorCity;
645
646
647 /* *****
648     6.10. In the example below, we retrieve all values from table `AP1.Vendors`
649     where `VendorState` could either be `CA` and `VendorCity` could
650     either be `Fresno` or `Sacramento` as one condition or `VendorState`
651     is `NJ` as another condition.
652     ***** */
653
654 SELECT *
655 FROM AP1.Vendors
656 WHERE (
657     VendorState IN ('CA')
658     AND VendorCity IN (
659         'Fresno',
660         'Sacramento'
661     )
662 )
663
664     OR VendorState IN ('NJ')
665
666 ORDER BY VendorState,
667     VendorCity;
668
669
670
671
672
673 /* *****
674     6.11. In the example below, we retrieve all values from table `AP1.Vendors`
675     where `VendorName` has as a value starting with `am` (not case
676     sensitive) using wild card `%` to represent any character or group of
677     after `am`.
678     ***** */
679
680 SELECT *
681 FROM AP1.Vendors
682 WHERE VendorName LIKE 'am%';
683
684
685
686
687 /* *****
688     6.12. In the example below, we retrieve all values from table `AP1.Vendors`
689     where `VendorName` has as a value with pattern `data` (not case
690     sensitive) using wild card `%` before and after the given string.

```

```

691      ***** */
692
693 SELECT *
694 FROM AP1.Vendors
695 WHERE VendorName LIKE '%data%';           -- returns various values like
696                                           -- `Expedata Inc`,
697                                           -- `California Data Marketing`
698                                           -- and `Quality Education Data`
699
700
701 /* *****
702     6.13. In the example below, we retrieve all values from table `AP1.Vendors`
703         where `VendorPhone` has as a value starting with `800` (string, not a
704         numeric value).
705     ***** */
706
707 SELECT *
708 FROM AP1.Vendors
709 WHERE VendorPhone LIKE '800%';
710
711
712 /* *****
713     6.14. In the example below, we retrieve all values from table `AP1.Vendors`
714         where `VendorPhone` has as a value NOT starting with `800`.
715     ***** */
716
717 SELECT *
718 FROM AP1.Vendors
719 WHERE VendorPhone NOT LIKE '800%';
720
721
722 /* *****
723     6.15. In the example below, we retrieve all values from table
724         `AP1.Invoices` where `InvoiceDueDate` has values within the range of
725         two dates -- `2012-01-01` and `2012-01-30` (dates always in single
726         quotes).
727     ***** */
728
729 SELECT *
730 FROM AP1.Invoices
731 WHERE InvoiceDueDate BETWEEN '2012-01-01'   -- range between `2012-01-01`
732         AND '2012-01-30';                 -- and `2012-01-30`
733
734
735 /* *****
736     6.16. In the example below, we retrieve all values from table `AP1.Vendors`
737         where InvoiceTotal has values within 100 and 1000. Then we organize
738         the results in descending order using an `ORDER BY` clause
739         (https://techonthenet.com/sql/order\_by.php).
740
741         The default option for `ORDER BY` is `ASC` (ascending), which can be
742         omitted.
743
744         The opposite option for `ORDER BY` is `DESC` (descending), which
745         needs to be specified for each field.
746     ***** */
747
748 SELECT *
749 FROM AP1.Invoices
750 WHERE InvoiceTotal BETWEEN 100           -- range between 100 and 1000
751         AND 1000
752 ORDER BY InvoiceTotal DESC,           -- organizing results
753                                           -- 1. first by `InvoiceTotal`
754                                           --    in descending order
755                                           --    (`DESC`)
756         PaymentTotal DESC,           -- 2. then by `PaymentTotal` in
757                                           --    descending order
758                                           --    (`DESC`)
759         TermsID DESC;                 -- 3. then finally by `TermsID`

```

```

760                                     --      also in descending order
761                                     --      (`DESC`)
762
763
764 /* *****
765 7. As we have mentioned several times, when calling multiple tables, we need
766 to `JOIN` them (https://techonthenet.com/sql\_server/joins.php).
767
768     7.01. `INNER JOIN` returns ``all rows from multiple tables where the join
769 condition is met.``
770
771     In the example below, we retrieve all records shared in tables
772 `AP1.Invoices` and `AP1.Invoices`.
773 ***** */
774
775 SELECT *
776 FROM AP1.Vendors
777 INNER JOIN AP1.Invoices
778     ON AP1.Vendors.VendorID = AP1.Invoices.VendorID;
779
780
781 /* *****
782     7.02. `LEFT JOIN` returns ``all rows from the LEFT-hand table specified in
783 the ON condition and only those rows from the other table where the
784 joined fields are equal (join conditions met).``
785
786     In the example below, we retrieve all records in `AP1.Vendors` (left
787 table) and any records in `AP1.Invoices` (if any in the right table).
788 ***** */
789
790 SELECT *
791 FROM AP1.Vendors                                     -- retrieves all records from
792 LEFT JOIN AP1.Invoices                               -- the left table/dataset
793     ON AP1.Vendors.VendorID=AP1.Invoices.VendorID;-- (first table/dataset
794                                                         -- called in the statement,
795                                                         -- `AP1.Vendors`) and related
796                                                         -- records from the right
797                                                         -- table/dataset (second
798                                                         -- table/dataset called in the
799                                                         -- statement, `AP1.Invoices`);
800                                                         -- returns 202 records
801
802
803 /* *****
804     In the example below, we retrieve all records in `AP1.Invoices` (left
805 table) and any records in `AP1.Vendors` (if any in the right table).
806 ***** */
807
808 SELECT *
809 FROM AP1.Invoices                                     -- retrieves all records from
810 LEFT JOIN AP1.Vendors                               -- the left table/dataset
811     ON AP1.Vendors.VendorID=AP1.Invoices.VendorID;-- (first table/dataset called
812                                                         -- in the statement,
813                                                         -- `AP1.Invoices`) and related
814                                                         -- records from the right
815                                                         -- table/dataset (second
816                                                         -- table/dataset called in the
817                                                         -- statement, `AP1.Vendors`)
818
819
820 /* *****
821     7.03. `RIGHT JOIN` returns ``all rows from the RIGHT-hand table specified in
822 the ON condition and only those rows from the other table where the
823 joined fields are equal (join condition is met).``
824
825     In the example below, we retrieve all records in `AP1.Invoices`
826 (right table) and any records in `AP1.Vendors` (if any in the left
827 table).
828 ***** */

```

```

829
830 SELECT *
831 FROM AP1.Vendors -- retrieves all records from
832 RIGHT JOIN AP1.Invoices -- the right table/dataset
833 ON AP1.Invoices.VendorID=AP1.Vendors.VendorID;-- (second table/dataset called
834 -- in the statement,
835 -- `AP1.Invoices`) and related
836 -- records from the left
837 -- table/dataset (first
838 -- table/dataset called in the
839 -- statement, `AP1.Invoices`)
840
841
842 /* *****
843 7.04. `FULL JOIN` returns ``all rows from the LEFT-hand table and RIGHT-
844 hand table with nulls in place where the join condition is not met.``
845 Note that depending on the size of the tables, this query might make
846 the server run slowly or crash it.
847
848 In the example below, we retrieve all records in `AP1.Invoices` (left
849 table) and all records in `AP1.Vendors` (right table).
850 ***** */
851
852 SELECT *
853 FROM AP1.Invoices -- retrieves all records from
854 FULL JOIN AP1.Vendors -- the left table/dataset
855 ON AP1.Vendors.VendorID=AP1.Invoices.VendorID;-- (first table/dataset called
856 -- in the statement,
857 -- `AP1.Vendors`) and all
858 -- records from the right
859 -- table/dataset (second
860 -- table/dataset called in the
861 -- statement, `AP1.Invoices`)
862
863
864 /* *****
865 8. Now that we have reviewed most of the material so far, we start views.
866
867 ``In a database management system, a view is a way of portraying
868 information in the database. This can be done by arranging the data
869 items in a specific order, by highlighting certain items, or by
870 showing only certain items. For any database, there are a number of
871 possible views that may be specified. Databases with many items tend
872 to have more possible views than databases with few items. Often
873 thought of as a virtual table, the view doesn't actually store
874 information itself, but just pulls it out of one or more existing
875 tables. Although impermanent, a view may be accessed repeatedly by
876 storing its criteria in a query.``
877 http://searchsqlserver.techtarget.com/definition/view
878
879 CREATE VIEW view_name AS
880 SELECT columns
881 FROM tables
882 [WHERE conditions];
883
884 8.01. In the example below, we modify table `AP1.Invoices` adding column
885 `CustomerID` in order to establish a relation between this table and
886 `AP2.Customers`.
887 ***** */
888
889 ALTER TABLE AP1.Invoices
890 ADD CustomerID INT NULL;
891
892 UPDATE AP1.Invoices
893 SET CustomerID = 1
894 WHERE VendorID = 34;
895
896 UPDATE AP1.Invoices
897 SET CustomerID = 2

```

```

898 WHERE VendorID = 37;
899
900 UPDATE AP1.Invoices
901 SET CustomerID = 3
902 WHERE VendorID = 89;
903
904
905 /* *****
906     8.02. Now that relationship has been created, we can now query tables
907     `AP1.Invoices` and `AP2.Customers` (each tables in a different
908     databases).
909     ***** */
910
911 SELECT DISTINCT AP1.Invoices.InvoiceID,
912     AP1.Invoices.VendorID,
913     AP1.Invoices.InvoiceNumber,
914     FORMAT (AP1.Invoices.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
915     FORMAT (AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
916     FORMAT (AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
917     FORMAT (AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
918     AP1.Invoices.TermsID,
919     FORMAT (AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
920     FORMAT (AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
921     AP1.Invoices.CustomerID,
922     AP2.Customers.LastName,
923     AP2.Customers.FirstName,
924     AP2.Customers.Address,
925     AP2.Customers.City,
926     AP2.Customers.STATE,
927     AP2.Customers.ZipCode,
928     AP2.Customers.Email
929 FROM AP1.Invoices
930 INNER JOIN AP2.Customers
931     ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
932 ORDER BY AP1.Invoices.VendorID;
933
934
935 /* *****
936     As with previous examples, we can use an alias for each table, which
937     in this case, allows us to present neater code.
938
939     `i` for `AP1.Invoices`
940     `c` for `AP2.Customers`
941     ***** */
942
943 SELECT DISTINCT i.InvoiceID,
944     i.VendorID,
945     i.InvoiceNumber,
946     FORMAT (i.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
947     FORMAT (i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
948     FORMAT (i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
949     FORMAT (i.CreditTotal, 'c', 'en-us') AS CreditTotal,
950     i.TermsID,
951     FORMAT (i.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
952     FORMAT (i.PaymentDate, 'd', 'en-us') AS PaymentDate,
953     i.CustomerID,
954     c.LastName,
955     c.FirstName,
956     c.Address,
957     c.City,
958     c.STATE,
959     c.ZipCode,
960     c.Email
961 FROM AP1.Invoices AS i
962 INNER JOIN AP2.Customers AS c
963     ON i.CustomerID = c.CustomerID
964 ORDER BY i.VendorID;
965
966

```

```

967 /* *****
968 8.03. In the example below, we can create a view using the query in the
969 example above using tables `AP1.Invoices` and `AP2.Customers`
970 without `ORDER BY`, which would return an error when creating the
971 view.
972
973 Tables and views cannot share names since both data objects are of
974 the same hierarchy.
975
976 We can query, alter and/or drop a view just like a table.
977
978 In most relational databases, we cannot update data using a view
979 since this action only take place in tables.
980
981 In SQL Server (T-SQL), we can update data from the base table.
982
983     `Requires UPDATE, INSERT, or DELETE permissions on the
984     target table, depending on the action being performed.`
985     https://msdn.microsoft.com/en-us/library/ms180800.aspx
986 ***** */
987
988 CREATE VIEW AP1.InvoicesCustomersVW
989 AS
990 (
991     SELECT DISTINCT AP1.Invoices.InvoiceID,
992         AP1.Invoices.VendorID,
993         AP1.Invoices.InvoiceNumber,
994         FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
995         FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
996         FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
997         FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
998         AP1.Invoices.TermsID,
999         FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
1000         FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
1001         AP1.Invoices.CustomerID,
1002         AP2.Customers.LastName,
1003         AP2.Customers.FirstName,
1004         AP2.Customers.Address,
1005         AP2.Customers.City,
1006         AP2.Customers.STATE,
1007         AP2.Customers.ZipCode,
1008         AP2.Customers.Email
1009     FROM AP1.Invoices
1010     INNER JOIN AP2.Customers
1011         ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
1012 );
1013
1014
1015 /* *****
1016     As with previous examples, we can use an alias for each table, which
1017     in this case, allows us to present neater code.
1018
1019     `i` for `AP1.Invoices`
1020     `c` for `AP2.Customers`
1021 ***** */
1022
1023 CREATE VIEW AP1.InvoicesCustomersVW
1024 AS
1025 (
1026     SELECT DISTINCT i.InvoiceID,
1027         i.VendorID,
1028         i.InvoiceNumber,
1029         FORMAT(i.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
1030         FORMAT(i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1031         FORMAT(i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1032         FORMAT(i.CreditTotal, 'c', 'en-us') AS CreditTotal,
1033         i.TermsID,
1034         FORMAT(i.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
1035         FORMAT(i.PaymentDate, 'd', 'en-us') AS PaymentDate,

```

```

1036         i.CustomerID,
1037         c.LastName,
1038         c.FirstName,
1039         c.Address,
1040         c.City,
1041         c.STATE,
1042         c.ZipCode,
1043         c.Email
1044     FROM AP1.Invoices AS i
1045     INNER JOIN AP2.Customers AS c
1046         ON i.CustomerID = c.CustomerID
1047 );
1048
1049
1050 /* *****
1051     8.04. We can modify a view simply changing `CREATE` for `ALTER`.
1052     ***** */
1053
1054 ALTER VIEW AP1.InvoicesCustomersVW
1055 AS
1056 (
1057     SELECT DISTINCT AP1.Invoices.InvoiceID,
1058         AP1.Invoices.VendorID,
1059         AP1.Invoices.InvoiceNumber,
1060         FORMAT(AP1.Invoices.InvoiceDate, 'd', 'en-us')
1061             AS InvoiceDate,
1062         FORMAT(AP1.Invoices.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1063         FORMAT(AP1.Invoices.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1064         FORMAT(AP1.Invoices.CreditTotal, 'c', 'en-us') AS CreditTotal,
1065         AP1.Invoices.TermsID,
1066         FORMAT(AP1.Invoices.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
1067         FORMAT(AP1.Invoices.PaymentDate, 'd', 'en-us') AS PaymentDate,
1068         AP1.Invoices.CustomerID,
1069         AP2.Customers.LastName,
1070         AP2.Customers.FirstName,
1071         AP2.Customers.Address,
1072         AP2.Customers.City,
1073         AP2.Customers.STATE,
1074         AP2.Customers.ZipCode,
1075         AP2.Customers.Email,
1076         GETDATE() AS SystemDate           -- change from previous query
1077     FROM AP1.Invoices
1078     INNER JOIN AP2.Customers
1079         ON AP1.Invoices.CustomerID = AP2.Customers.CustomerID
1080 );
1081
1082
1083 /* *****
1084     As with previous examples, we can use an alias for each table, which
1085     in this case, allows us to present neater code.
1086
1087         `i` for `AP1.Invoices`
1088         `c` for `AP2.Customers`
1089     ***** */
1090
1091 ALTER VIEW AP1.InvoicesCustomersVW
1092 AS
1093 (
1094     SELECT DISTINCT i.InvoiceID,
1095         i.VendorID,
1096         i.InvoiceNumber,
1097         FORMAT(i.InvoiceDate, 'd', 'en-us') AS InvoiceDate,
1098         FORMAT(i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1099         FORMAT(i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1100         FORMAT(i.CreditTotal, 'c', 'en-us') AS CreditTotal,
1101         i.TermsID,
1102         FORMAT(i.InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
1103         FORMAT(i.PaymentDate, 'd', 'en-us') AS PaymentDate,
1104         i.CustomerID,

```



```

1105         c.LastName,
1106         c.FirstName,
1107         c.Address,
1108         c.City,
1109         c.STATE,
1110         c.ZipCode,
1111         c.Email,
1112         GETDATE() AS SystemDate           -- no table alias since value
1113                                           -- is returned by function
1114                                           -- `GETDATE()`, which calls the
1115                                           -- system DATETIME
1116 FROM AP1.Invoices AS i
1117 INNER JOIN AP2.Customers AS c
1118     ON i.CustomerID = c.CustomerID
1119 );
1120
1121
1122 /* *****
1123 8.05. In the example below, we create view `AP1.InvoicesVW` only from table
1124 `AP1.Invoices` formatting the date and currency fields accordingly.
1125 This way we do not need to format the columns again and again every
1126 time we need to call them.
1127 ***** */
1128
1129 CREATE VIEW AP1.InvoicesVW
1130 AS
1131 (
1132     SELECT DISTINCT InvoiceID,
1133         VendorID,
1134         InvoiceNumber,
1135         FORMAT(InvoiceDate, 'd', 'en-us') AS InvoiceDate,
1136         FORMAT(InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1137         FORMAT(PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1138         FORMAT(CreditTotal, 'c', 'en-us') AS CreditTotal,
1139         TermsID,
1140         FORMAT(InvoiceDueDate, 'd', 'en-us') AS InvoiceDueDate,
1141         FORMAT(PaymentDate, 'd', 'en-us') AS PaymentDate,
1142         CustomerID
1143 FROM AP1.Invoices           -- no need for a table alias as
1144 );                           -- all values come from the one
1145                             -- table (`AP1.Invoices`)
1146
1147
1148 /* *****
1149 8.06. In the example below, we create view `AP1.InvoicesVendorsVW` from
1150 tables `AP1.Invoices` and `AP1.Vendors`.
1151
1152 Unless we indicate in which database to store the view, it would most
1153 likely be in the same database where the previous view was stored
1154 (`AP2`).
1155 ***** */
1156
1157 CREATE VIEW AP1.InvoicesVendorsVW
1158 AS
1159 (
1160     SELECT DISTINCT AP1.Invoices.InvoiceID,
1161         AP1.Invoices.VendorID,
1162         AP1.Invoices.InvoiceNumber,
1163         AP1.Invoices.InvoiceDate,
1164         AP1.Invoices.InvoiceTotal,
1165         AP1.Invoices.PaymentTotal,
1166         AP1.Invoices.CreditTotal,
1167         AP1.Invoices.TermsID,
1168         AP1.Invoices.InvoiceDueDate,
1169         AP1.Invoices.PaymentDate,
1170         AP1.Vendors.VendorName,
1171     CASE
1172         WHEN AP1.Vendors.VendorAddress2 IS NOT NULL
1173     THEN CONCAT (

```

```

1174         AP1.Vendors.VendorAddress1,
1175         ' ',
1176         AP1.Vendors.VendorAddress2
1177     )
1178     WHEN AP1.Vendors.VendorAddress1 IS NULL
1179         AND AP1.Vendors.VendorAddress2 IS NULL
1180         THEN 'No Address'
1181     ELSE AP1.Vendors.VendorAddress1
1182     END AS VendorAddress,
1183     AP1.Vendors.VendorCity,
1184     AP1.Vendors.VendorState,
1185     AP1.Vendors.VendorZipCode,
1186     AP1.Vendors.DefaultAccountNo
1187 FROM AP1.Invoices
1188 LEFT JOIN AP1.Vendors
1189     ON AP1.Invoices.VendorID = AP1.Vendors.VendorID
1190 );
1191
1192
1193 /* *****
1194     As with previous examples, we can use an alias for each table, which
1195     in this case, allows us to present neater code.
1196
1197         `i` for `AP1.Invoices`
1198         `v` for `AP1.Vendors`
1199 ***** */
1200
1201 CREATE VIEW AP1.InvoicesVendorsVW
1202 AS
1203 (
1204     SELECT DISTINCT i.InvoiceID,
1205         i.VendorID,
1206         i.InvoiceNumber,
1207         i.InvoiceDate,
1208         i.InvoiceTotal,
1209         i.PaymentTotal,
1210         i.CreditTotal,
1211         i.TermsID,
1212         i.InvoiceDueDate,
1213         i.PaymentDate,
1214         v.VendorName,
1215     CASE
1216         WHEN v.VendorAddress2 IS NOT NULL
1217             THEN CONCAT (
1218                 v.VendorAddress1,
1219                 ' ',
1220                 v.VendorAddress2
1221             )
1222         WHEN v.VendorAddress1 IS NULL
1223             AND v.VendorAddress2 IS NULL
1224             THEN 'No Address'
1225         ELSE v.VendorAddress1
1226         END AS VendorAddress,
1227     v.VendorCity,
1228     v.VendorState,
1229     v.VendorZipCode,
1230     v.DefaultAccountNo
1231 FROM AP1.Invoices AS i
1232 LEFT JOIN AP1.Vendors AS v
1233     ON i.VendorID = v.VendorID
1234 );
1235
1236
1237 /* *****
1238     8.07. In the example below, we create view
1239         `AP1.Invoices_Customers_Vendors_VW` from views (like we would do with
1240         tables) `AP1.InvoicesCustomersVW` and `AP1.InvoicesVendorsVW` in
1241         schema `AP1`, which we must specify.
1242

```

1243 As mentioned, unless we indicate in which schema to store the new
1244 view, the new object would be saved in default schema `dbo`. In this
1245 case, we do not need to call the schema (`dbo`), but it is always a
1246 good idea -- good practice. Note that if we need to call the
1247 database name, we must call the schema (`db1.dbo.table1`).

***** */

1249 **CREATE VIEW** AP1.Invoices_Customers_Vendors_VW

1250 **AS**

1251 (
1252

1253 **SELECT DISTINCT** AP1.InvoicesCustomersVW.InvoiceID,

1254 AP1.InvoicesCustomersVW.VendorID,

1255 AP1.InvoicesCustomersVW.InvoiceNumber,

1256 AP1.InvoicesCustomersVW.InvoiceDate,

1257 AP1.InvoicesCustomersVW.InvoiceTotal,

1258 AP1.InvoicesCustomersVW.PaymentTotal,

1259 AP1.InvoicesCustomersVW.CreditTotal,

1260 AP1.InvoicesCustomersVW.TermsID,

1261 AP1.InvoicesCustomersVW.InvoiceDueDate,

1262 AP1.InvoicesCustomersVW.PaymentDate,

1263 AP1.InvoicesCustomersVW.CustomerID,

1264 AP1.InvoicesCustomersVW.LastName,

1265 AP1.InvoicesCustomersVW.FirstName,

1266 AP1.InvoicesCustomersVW.Address,

1267 AP1.InvoicesCustomersVW.City,

1268 AP1.InvoicesCustomersVW.**STATE**,

1269 AP1.InvoicesCustomersVW.ZipCode,

1270 AP1.InvoicesCustomersVW.Email,

1271 AP1.InvoicesVendorsVW.VendorName,

1272 AP1.InvoicesVendorsVW.VendorAddress,

1273 AP1.InvoicesVendorsVW.VendorCity,

1274 AP1.InvoicesVendorsVW.VendorState,

1275 AP1.InvoicesVendorsVW.VendorZipCode,

1276 AP1.InvoicesVendorsVW.DefaultAccountNo

1277 **FROM** AP1.InvoicesCustomersVW

1278 **LEFT OUTER JOIN** AP1.InvoicesVendorsVW

1279 **ON** AP1.InvoicesCustomersVW.VendorID = AP1.InvoicesVendorsVW.VendorID

1280);
1281

1282

1283

1284 /* *****

1285 As with previous examples, we can use an alias for each table, which

1286 in this case, allows us to present neater code.

1287

1288 **SELECT DISTINCT** `icv` for `AP1.InvoicesCustomersVW`

1289 **SELECT DISTINCT** `ivv` for `AP1.InvoicesVendorsVW`

***** */

1290

1291 **CREATE VIEW** AP1.Invoices_Customers_Vendors_VW

1292 **AS**

1293 (
1294

1295 **SELECT DISTINCT** ivv.InvoiceID,

1296 ivv.VendorID,

1297 ivv.InvoiceNumber,

1298 ivv.InvoiceDate,

1299 ivv.InvoiceTotal,

1300 ivv.PaymentTotal,

1301 ivv.CreditTotal,

1302 ivv.TermsID,

1303 ivv.InvoiceDueDate,

1304 ivv.PaymentDate,

1305 ivv.CustomerID,

1306 ivv.LastName,

1307 ivv.FirstName,

1308 ivv.Address,

1309 ivv.City,

1310 ivv.**STATE**,

1311 ivv.ZipCode,

1312 ivv.Email,

```

1312         icv.VendorName,
1313         icv.VendorAddress,
1314         icv.VendorCity,
1315         icv.VendorState,
1316         icv.VendorZipCode,
1317         icv.DefaultAccountNo
1318     FROM AP1.InvoicesCustomersVW AS icv
1319     LEFT OUTER JOIN AP1.InvoicesVendorsVW AS ivv
1320         ON ivv.VendorID = icv.VendorID
1321     );
1322
1323
1324 /* *****
1325 9. Depending on the relational database management system (RDBMS) and even the
1326 product related to each RDBMS, the date format might vary. In SQL Server,
1327 we can query data using format `YYYY/MM/DD` (including quotes) although the
1328 system returns format `YYYY-MM-DD` plus time in format `hh:mm:ss.nnnnnn`.
1329 Refer to https://msdn.microsoft.com/en-us/library/bb630352.aspx and
1330 https://msdn.microsoft.com/en-us/library/bb677243.aspx for information on
1331 date and time respectively.
1332
1333 9.01. The most common date functions are the following.
1334
1335         DAY()         returns the day of the month (1 to 31) given a date value
1336                       https://techonthenet.com/sql\_server/functions/day.php
1337
1338         MONTH()      returns the month (1 to 12) given a date value
1339                       https://techonthenet.com/sql\_server/functions/month.php
1340
1341         YEAR()       returns a four-digit year (as a number) given a date value
1342                       https://techonthenet.com/sql\_server/functions/year.php
1343
1344         GETDATE()    returns the current date and time
1345                       https://techonthenet.com/sql\_server/functions/getdate.php
1346 ***** */
1347
1348 SELECT DAY('2021/09/15') AS Day,           -- 1. returns `15` from
1349                                             -- `2021/09/15` without
1350                                             -- leading zeros (`d`)
1351     MONTH('2021/09/15') AS Month,        -- 2. returns `9` from
1352                                             -- `2021/09/15` without
1353                                             -- leading zeros (`M`)
1354     YEAR('2021/09/15') AS Year;         -- 3. returns `2021` from
1355                                             -- `2021/09/15` (`yyyy`)
1356
1357 SELECT GETDATE() AS CurrentDateTime;     -- returns
1358                                             -- `2021-09-15 20:20:34.053`
1359                                             -- from `GETDATE()` that calls
1360                                             -- system date and time
1361
1362 SELECT DAY(GETDATE()) AS Day,           -- 1. returns `15` from system
1363                                             -- DATETIME without leading
1364                                             -- zeros (`d`)
1365     MONTH(GETDATE()) AS Month,        -- 2. returns `9` from system
1366                                             -- DATETIME without leading
1367                                             -- zeros (`M`)
1368     YEAR(GETDATE()) AS Year;         -- 3. returns `2021` from
1369                                             -- system DATETIME (`yyyy`)
1370
1371 SELECT FORMAT(GETDATE(), 'd', 'en-us')   -- returns system date and time
1372     AS FormattedCurrentDateTime;       -- formatted as `9/15/2021`
1373
1374 /* *****
1375 9.02. Instead of hard-coding the date in the example above (#3.1), we can
1376 use parameter `@date` in all instances that we need to pass the value
1377 returned by `GETDATE()`.
1378
1379 We must declare each parameter with its proper data type.
1380

```



```

1450 /* *****
1451 Note that we can declare the parameter and assign a value
1452 in-line instead of using a `SET` statement.
1453
1454 DECLARE @persons_name = 'Linus Torvalds',
1455         @start_date = '12/28/1969',
1456         @end_date = GETDATE(),
1457         @tmp_date DATETIME,
1458         @years INT,
1459         @months INT,
1460         @days INT;
1461 ***** */
1462
1463 SELECT @tmp_date = @start_date;
1464
1465
1466 /* *****
1467 9.03.06. Date functions `DATEADD()` returns ``a specified date with
1468 the specified number interval (signed integer) added to a
1469 specified datepart of that date``
1470 (https://msdn.microsoft.com/en-us/library/ms186819.aspx) and
1471 `DATEDIFF()` returns ``the count (signed integer) of the
1472 specified datepart boundaries crossed between the specified
1473 start_date and end_date``
1474 (https://msdn.microsoft.com/en-us/library/ms189794.aspx).
1475
1476 `YEAR()` retrieves the year (`yy`) from the date.
1477
1478 `MONTH()` retrieves the month (`m`) from the date.
1479
1480 `DAY()` retrieves the day (`d`) from the date.
1481
1482 9.03.07. The `CASE WHEN` statement uses a true value (situation we are
1483 looking for) coming from `WHEN... THEN` to trigger an action
1484 and an `ELSE` value to trigger an alternative action using
1485 the following syntax.
1486
1487 9.03.08. Below `@years` is assigned the difference of `@tmp_date` and
1488 `@end_date` in years when the month in the year (`yy`) in
1489 `@start_date` is less than the month in `@end_date` or it is
1490 the same as the month in `@end_date`
1491
1492 MONTH(@start_date) > MONTH(@end_date)
1493 OR (MONTH(@start_date) = MONTH(@end_date))
1494
1495 and the day in `@start_date` is less than the day in
1496 `@end_date`.
1497
1498 AND DAY(@start_date) > DAY(@end_date)
1499
1500 If both conditions are true, the query returns `1` (under a
1501 full year). Otherwise it returns `0` (full year).
1502 ***** */
1503
1504 SELECT @years = DATEDIFF(yy, @tmp_date, @end_date) - CASE
1505     WHEN (MONTH(@start_date) > MONTH(@end_date))
1506     OR (
1507         MONTH(@start_date) = MONTH(@end_date)
1508         AND DAY(@start_date) > DAY(@end_date)
1509     )
1510     THEN 1
1511     ELSE 0
1512     END;
1513
1514
1515 /* *****
1516 9.03.09. We add the value of `@years` (`yy`) to `@tmp_date` returning
1517 1 or 0.
1518 ***** */

```

```

1519
1520 SELECT @tmp_date = DATEADD(yy, @years, @tmp_date);
1521
1522
1523 /* *****
1524     9.03.10. Below `@months` is assigned the difference of `@tmp_date` and
1525     `@end_date` in months when the month (`m`) in `@start_date`
1526     is less than the month in `@end_date` or it is the same as
1527     the month in `@end_date`.
1528
1529     DAY(@start_date) > DAY(@end_date)
1530
1531     If the condition is true, the query returns `1` (under a full
1532     month). Otherwise it returns `0` (full month).
1533     ***** */
1534
1535 SELECT @months = DATEDIFF(m, @tmp_date, @end_date) - CASE
1536     WHEN DAY(@start_date) > DAY(@end_date)
1537     THEN 1
1538     ELSE 0
1539     END;
1540
1541
1542 /* *****
1543     9.03.11. We add the value of `@months` (`m`) to `@tmp_date` returning
1544     1 or 0.
1545     ***** */
1546
1547 SELECT @tmp_date = DATEADD(m, @months, @tmp_date);
1548
1549
1550 /* *****
1551     9.03.12. Below `@days` is assigned the difference of `@tmp_date` and
1552     `@end_date` in days.
1553     ***** */
1554
1555 SELECT @days = DATEDIFF(d, @tmp_date, @end_date);
1556
1557
1558 /* *****
1559     9.03.13. We finally display the values for `@years`, `@months` and
1560     `@days`.
1561
1562     +-----+-----+-----+-----+
1563     | Person's Name | Years | Months | Days |
1564     +-----+-----+-----+-----+
1565     | Linus Torvalds | 53    | 3      | 28   |
1566     +-----+-----+-----+-----+
1567
1568     9.03.14. You can also use the script to calculate your age or any
1569     difference between any two dates by changing the values in
1570     section #9.03.04.
1571
1572     The value returned by `GETDATE()` when running this script
1573     was 2021/11/29 and the end result will change according to
1574     the current date when the script is run.
1575     ***** */
1576
1577 SELECT @persons_name AS 'Person''s Name',           -- two single quotes (``) to
1578                                                -- escape and show only one
1579                                                -- (``)
1580     @years AS 'Years',
1581     @months AS 'Months',
1582     @days AS 'Days';
1583
1584
1585 /* *****
1586     https://folvera.commons.gc.cuny.edu/?p=1287
1587     ***** */

```