


```

53      REPLACE(
54          REPLACE(AP1.Vendors.VendorName, ' ', ''))
55          , '.', '')
56          , ',', '')
57          , '/', '')
58          , '&', '')
59          , '...', '') +
60
61
62
63
64
65      '.foo'
66      AS VendorContactEmail,
67
68      AP1.Vendors.DefaultTermsID,
69      AP1.Vendors.DefaultAccountNo,
70      -- AP1.ContactUpdates.VendorID AS Expr1,
71      AP1.ContactUpdates.FirstName +
72      ' ' +
73      AP1.ContactUpdates.LastName
74      AS ContactName,
75
76      LEFT(AP1.ContactUpdates.FirstName, 1) +
77      AP1.ContactUpdates.LastName +
78      '@' +
79
80
81
82      REPLACE(
83          REPLACE(
84              REPLACE(
85                  REPLACE(
86                      REPLACE(
87                          REPLACE(AP1.Vendors.VendorName, ' ', ''))
88                          , '.', '')
89                          , ',', '')
90                          , '/', '')
91                          , '&', '')
92                          , '...', '') +
93
94
95
96
97
98      '.foo'
99      AS VendorContactEmail
100
101 FROM AP1.Vendors
102 LEFT JOIN AP1.ContactUpdates
103     ON AP1.Vendors.VendorID = AP1.ContactUpdates.VendorID;
104

```

-- 2.05. getting value #1
-- 2.06. generating value #1
-- 2.07. generating value #2
-- 2.08. generating value #3
-- 2.09. generating value #4
-- 2.10. generating value #5
-- 2.11. generating value #6
-- where ''
-- represents a single
-- quote (` ``) used an
-- escape character
-- and ` .foo` to complete
-- email with alias
-- `VendorContactEmail`

-- 3. concatenation of
-- `VendorContactFName`, a
-- single space and
-- `LastName` using `+` with
-- alias `VendorContactLName`

-- 4. concatenation of
-- `VendorContactFName`,
-- a single space and
-- `VendorContactLName`,
-- the `@` sign, followed by
-- `VendorName` (value #6)

-- 4.01. getting value #5
-- 4.02. getting value #4
-- 4.03. getting value #3
-- 4.04. getting value #2
-- 4.05. getting value #1

-- 4.06. generating value #1
-- 4.07. generating value #2
-- 4.08. generating value #3
-- 4.09. generating value #4
-- 4.10. generating value #5
-- 4.11. generating value #6
-- where ''
-- represents a single
-- quote (` ``) used an
-- escape character
-- and ` .foo` to complete
-- email with alias
-- `VendorContactEmail`

```
105
106 /* ****
107     As with previous example, we can use an alias for each table, which
108     in this case, allows us to present neater code.
109
110         `v` for `AP1.Vendors`
111         `c` for `AP1.ContactUpdates`
112 **** */
113
114 SELECT v.VendorID,
115     v.VendorName,
116     v.VendorAddress1,
117     v.VendorAddress2,
118     v.VendorCity,
119     v.VendorState,
120     v.VendorZipCode,
121     v.VendorPhone,
122     v.VendorContactFName +
123         ' ' +
124     v.VendorContactLName AS ContactName,
125     LEFT(v.VendorContactFName, 1) +
126     v.VendorContactLName + '@' +
127     REPLACE(
128         REPLACE(
129             REPLACE(
130                 REPLACE(
131                     REPLACE(
132                         REPLACE(v.VendorName, ' ', ' ')
133                         , ' ', ' ')
134                         , ' ', ' ')
135                         , '/', ' ')
136                         , '&', ' ')
137                         , '...', ' ')
138         + '.foo' AS VendorContactEmail,
139     v.DefaultTermsID,
140     v.DefaultAccountNo,
141     -- c.VendorID AS Expr1,
142     c.FirstName +
143         ' ' +
144     c.LastName AS ContactName,
145     LEFT(c.FirstName, 1) +
146         c.LastName +
147         '@' +
148     REPLACE(
149         REPLACE(
150             REPLACE(
151                 REPLACE(
152                     REPLACE(
153                         REPLACE(v.VendorName, ' ', ' ')
154                         , ' ', ' ')
155                         , ' ', ' ')
156                         , '/', ' '))
```

```

157      , '&', '')
158      , '...', '')
159      + '.foo' AS VendorContactEmail
160  FROM AP1.Vendors AS v
161  LEFT JOIN AP1.ContactUpdates AS c
162    ON v.VendorID = c.VendorID;
163
164
165 /* *****
166      Instead of using a plus sign (`+`), we can use `CONCAT()` since
167      adding a value and NULL returns NULL. In other words, we lose data,
168      which would be logical error (not syntax error).
169 *****/
170
171 SELECT AP1.Vendors.VendorID,
172       AP1.Vendors.VendorName,
173       AP1.Vendors.VendorAddress1,
174       AP1.Vendors.VendorAddress2,
175       AP1.Vendors.VendorCity,
176       AP1.Vendors.VendorState,
177       AP1.Vendors.VendorZipCode,
178       AP1.Vendors.VendorPhone,
179       CONCAT (
180             AP1.Vendors.VendorContactFName,
181             ' ',
182             AP1.Vendors.VendorContactLName
183           ) AS ContactName,
184       CONCAT (
185             LEFT(AP1.Vendors.VendorContactFName, 1),
186             AP1.Vendors.VendorContactLName,
187             '@',
188
189
190       REPLACE(
191         REPLACE(
192           REPLACE(
193             REPLACE(
194               REPLACE(
195                 REPLACE(AP1.Vendors.VendorName, ' ', ''),
196
197                 , ' . ', '')
198                 , ' , ', '')
199                 , '/ ', '')
200                 , '&', '')
201                 , '...', '')
202
203
204
205
206       , '.foo'
207     ) AS VendorContactEmail,
208

```

-- 1. concatenation of
-- `FirstName`, a single
-- space and `LastName` with
-- alias `ContactName`

-- 2. concatenation of
-- `VendorContactFName`,
-- a single space and
-- `VendorContactLName`,
-- the `@` sign, followed by
-- `VendorName` (value #6)

-- 2.01. getting value #5
-- 2.02. getting value #4
-- 2.03. getting value #3
-- 2.04. getting value #2
-- 2.05. getting value #1

-- 2.06. generating value #1
-- 2.07. generating value #2
-- 2.08. generating value #3
-- 2.09. generating value #4
-- 2.10. generating value #5
-- 2.11. generating value #6
-- where `''`
-- represents a single
-- quote (`'`) used an
-- escape character
-- and `'.foo` to complete
-- email with alias
-- `VendorContactEmail`

```

209  AP1.Vendors.DefaultTermsID,
210  AP1.Vendors.DefaultAccountNo,
211  -- AP1.ContactUpdates.VendorID AS Expr1,
212  CONCAT (
213      AP1.ContactUpdates.FirstName,
214      ' ',
215      AP1.ContactUpdates.LastName
216  ) AS ContactName,
217  CONCAT (
218      LEFT(AP1.ContactUpdates.FirstName, 1),
219      AP1.ContactUpdates.LastName,
220      '@',
221
222
223  REPLACE(
224      REPLACE(
225          REPLACE(
226              REPLACE(
227                  REPLACE(
228                      REPLACE(AP1.Vendors.VendorName, '''', ''')
229                          , '.', '')
230                          , ',', '')
231                          , '/', '')
232                          , '&', '')
233                          , '''', '''),
234
235
236
237
238
239  '.foo'
240  ) AS VendorContactEmail
241
242 FROM AP1.Vendors
243 LEFT JOIN AP1.ContactUpdates
244  ON AP1.Vendors.VendorID = AP1.ContactUpdates.VendorID;
245
246
247 /* *****
248     Using `CONCAT()` also returns a logical error (not syntax error)
249     since the concatenation to make the second email returns values like
250     `@USPostalService.foo` since there is no corresponding `FirstName`
251     and `LastName`. We could use a CASE clause.
252 *****/
253
254 SELECT AP1.Vendors.VendorID,
255     AP1.Vendors.VendorName,
256     AP1.Vendors.VendorAddress1,
257     AP1.Vendors.VendorAddress2,
258     AP1.Vendors.VendorCity,
259     AP1.Vendors.VendorState,
260     AP1.Vendors.VendorZipCode,

```

```

261     AP1.Vendors.VendorPhone,
262     CONCAT (
263         AP1.Vendors.VendorContactFName,
264         ' ',
265         AP1.Vendors.VendorContactLName
266     ) AS ContactName,
267     CONCAT (
268         LEFT(AP1.Vendors.VendorContactFName, 1),
269         AP1.Vendors.VendorContactLName,
270         '@',
271
272     REPLACE(
273         REPLACE(
274             REPLACE(
275                 REPLACE(
276                     REPLACE(
277                         REPLACE(
278                             REPLACE(AP1.Vendors.VendorName, '','')
279
280                             , ' ', ' ')
281                             , ' ', ' ')
282                             , '/ ', ' ')
283                             , '&', ' ')
284                             , ' ', ' ')
285
286
287
288
289     , '.foo'
290 ) AS VendorContactEmail,
291
292 AP1.Vendors.DefaultTermsID,
293 AP1.Vendors.DefaultAccountNo,
294 -- AP1.ContactUpdates.VendorID AS Expr1,
295 CONCAT (
296     AP1.ContactUpdates.FirstName,
297     ' ',
298     AP1.ContactUpdates.LastName
299 ) AS ContactName,
300
301 CASE
302     WHEN (AP1.ContactUpdates.FirstName <> '')
303     OR AP1.ContactUpdates.FirstName <> ''
304     OR AP1.ContactUpdates.FirstName IS NOT NULL) --
305
306
307
308
309
310
311     AND (AP1.ContactUpdates.LastName <> '')
312     OR AP1.ContactUpdates.LastName <> ''

```

-- 1. concatenation of
-- `FirstName`, a single
-- space and `LastName` with
-- alias `ContactName`

-- 2. concatenation of
-- `VendorContactFName`,
-- a single space and
-- `VendorContactLName`,
-- the `@` sign, followed by
-- `VendorName` (value #6)

-- 2.01. getting value #5
-- 2.02. getting value #4
-- 2.03. getting value #3
-- 2.04. getting value #2
-- 2.05. getting value #1

-- 2.06. generating value #1
-- 2.07. generating value #2
-- 2.08. generating value #3
-- 2.09. generating value #4
-- 2.10. generating value #5
-- 2.11. generating value #6
-- where `''`
-- represents a single
-- quote (` `` `) used as
-- an escape character
-- and `foo` to complete
-- email with alias
`VendorContactEmail`

-- 3. concatenation of
-- `VendorContactFName`, a
-- single space and
-- `LastName` with alias
`VendorContactLName`

-- 4. checking if
`ContactUpdates` is not
an empty string or not
equal to a space or
`IS NOT NULL` (in other
words, not no-value; must
have a value) using
parenthesis to make a
block of three conditions
and that `ContactUpdates`
is not an empty string or

```

313      OR AP1.ContactUpdates.LastName IS NOT NULL)-- not equal to a space or
314                                         -- `IS NOT NULL` (in other
315                                         -- words, not no-value; must
316                                         -- have a value) using
317                                         -- parenthesis to make a
318                                         -- block of three conditions
319      THEN CONCAT (
320          LEFT(AP1.ContactUpdates.FirstName, 1),
321          AP1.ContactUpdates.LastName,
322          '@',
323
324          REPLACE(
325              REPLACE(
326                  REPLACE(
327                      REPLACE(
328                          REPLACE(
329                              REPLACE(
330                                  REPLACE(
331                                      REPLACE(
332                                          REPLACE(AP1.Vendors.VendorName, '''', '')
333
334                                          , '.', '')
335                                          , '.', '')
336                                          , '/', '')
337                                          , '&', '')
338                                          , ''', '''),
339
340
341
342
343      '.foo'
344  )
345 ELSE ''
346 END
347 AS VendorContactEmail
348
349 FROM AP1.Vendors
350 LEFT JOIN AP1.ContactUpdates
351   ON AP1.Vendors.VendorID = AP1.ContactUpdates.VendorID;
352
353
354 /* ****
355      As with previous examples, we can use an alias for each table, which
356      in this case, allows us to present neater code.
357
358      `v` for `AP1.Vendors`
359      `c` for `AP1.ContactUpdates`
360 **** */
361
362 SELECT v.VendorID,
363       v.VendorName,
364       v.VendorAddress1,

```

```
365     v.VendorAddress2,
366     v.VendorCity,
367     v.VendorState,
368     v.VendorZipCode,
369     v.VendorPhone,
370     CONCAT (
371         v.VendorContactFName,
372         ' ',
373         v.VendorContactLName
374     ) AS ContactName,
375     CONCAT (
376         LEFT(v.VendorContactFName, 1),
377         v.VendorContactLName,
378         '@',
379         REPLACE(
380             REPLACE(
381                 REPLACE(
382                     REPLACE(
383                         REPLACE(
384                             REPLACE(v.VendorName, ' ', ' ')
385                             , ' ', ' ')
386                             , ' ', ' ')
387                             , '/', ' ')
388                             , '&', ' ')
389                             , '...', ' '),
390             '.foo'
391         ) AS VendorContactEmail,
392     v.DefaultTermsID,
393     v.DefaultAccountNo,
394     -- c.VendorID AS Expr1,
395     CONCAT (
396         c.FirstName,
397         ' ',
398         c.LastName
399     ) AS ContactName,
400     CASE
401         WHEN (
402             c.FirstName <> ''
403             OR c.FirstName <> ''
404             OR c.FirstName IS NOT NULL
405         )
406         AND (
407             c.LastName <> ''
408             OR c.LastName <> ''
409             OR c.LastName IS NOT NULL
410         )
411         THEN CONCAT (
412             LEFT(c.FirstName, 1),
413             c.LastName,
414             REPLACE(
415                 REPLACE(
416                     REPLACE(
```

```
417      REPLACE(
418          REPLACE(
419              REPLACE(v.VendorName, ' ', '')
420              , ',', '')
421              , '.', '')
422              , '/', '')
423              , '&', '')
424              , '...', ''),
425      '.foo'
426  )
427 ELSE ''
428 END AS VendorContactEmail
429 FROM AP1.Vendors AS v
430 LEFT JOIN AP1.ContactUpdates AS c
431 ON v.VendorID = c.VendorID;
432
433
434 /* *****
435 2. Before you continue learning about SQL
436 (https://searchsqlserver.techtarget.com/definition/SQL) syntax
437 (https://whatis.techtarget.com/definition/syntax), we should cover some
438 important theory, which you will need whether you need to learn SQL to run
439 queries at work and/or you decide to become a database administrator (DBA).
440
441 2.01. SQL (Structured Query Language) is a standardized programming
442 language used for managing relational databases and performing
443 various operations on the data in them. Initially created in the
444 1970s, SQL is regularly used by database administrators, as well as
445 by developers writing data integration scripts and data analysts
446 looking to set up and run analytical queries.
447 https://searchsqlserver.techtarget.com/definition/SQL
448
449 2.02. ISO/IEC 9075-1:2016 [SQL:2016] describes the conceptual framework
450 used in other parts of ISO/IEC 9075 to specify the grammar of SQL and
451 the result of processing statements in that language by an
452 SQL-implementation.
453 ISO/IEC 9075-1:2016 also defines terms and notation used in the other
454 parts of ISO/IEC 9075.
455 https://www.iso.org/standard/63555.html
456
457 2.03. T-SQL (Transact-SQL) is a set of programming extensions from Sybase
458 and Microsoft that add several features to the Structured Query
459 Language (SQL), including transaction control, exception and error
460 handling, row processing and declared variables.
461 https://searchsqlserver.techtarget.com/definition/T-SQL
462
463 2.04. A relational database is a set of tables containing data fitted into
464 predefined categories. Each table (which is sometimes called a
465 relation) contains one or more data categories in columns. Each row
466 contains a unique instance of data for the categories defined by the
467 columns.
468 http://searchsqlserver.techtarget.com/definition/relational-database
```

469
470 2.05. Microsoft SQL Server is a relational database management system, or
471 RDBMS, that supports a wide variety of transaction processing,
472 business intelligence and analytics applications in corporate IT
473 environments. It's one of the three market-leading database
474 technologies, along with Oracle Database and IBM's DB2.
475 Like other RDBMS software, Microsoft SQL Server is built on top of
476 SQL, a standardized programming language that database administrators
477 (DBAs) and other IT professionals use to manage databases and query
478 the data they contain. SQL Server is tied to Transact-SQL (T-SQL),
479 an implementation of SQL from Microsoft that adds a set of
480 proprietary programming extensions to the standard language. The
481 original SQL Server code was developed in the 1980s by the former
482 Sybase Inc., which is now owned by SAP. Sybase initially built the
483 software to run on Unix systems and minicomputer platforms. It,
484 Microsoft and Ashton-Tate Corp., then the leading vendor of PC
485 databases, teamed up to produce the first version of what became
486 Microsoft SQL Server, designed for the OS/2 operating system and
487 released in 1989.
488 <https://searchsqlserver.techtarget.com/definition/SQL-Server>
489
490 2.06. Another form of flat file is one in which table data is gathered in
491 lines of ASCII text with the value from each table cell separated by
492 a comma and each row represented with a new line. This type of flat
493 file is also known as a comma-separated values file (CSV) file.
494 <http://searchsqlserver.techtarget.com/definition/flat-file>
495
496 2.07. A hierarchical database is a design that uses a one-to-many
497 relationship for data elements. Hierarchical database models use a
498 tree structure that links a number of disparate elements to one
499 `owner,` or `parent,` primary record.
500 <https://www.techopedia.com/definition/19782/hierarchical-database>
501
502 2.08. Data Manipulation Language (DML) is the ``vocabulary used to retrieve
503 and work with data... to add, modify, query, or remove data``
504 (<https://msdn.microsoft.com/en-us/library/ff848766.aspx>).
505
506 SELECT to retrieve records from one or more tables
507 <https://techonthenet.com/sql/select.php>
508
509 INSERT to insert a one or more records into a table
510 <https://techonthenet.com/sql/insert.php>
511
512 UPDATE to update existing records in the tables
513 <https://techonthenet.com/sql/update.php>
514
515 DELETE to delete a one or more records from a table
516 <https://techonthenet.com/sql/delete.php>
517
518 MERGE to insert, update, or delete operations on a target table
519 based on the results of a join with a source table
520 <https://msdn.microsoft.com/en-us/library/bb510625.aspx>

521
522 2.09. Data Definition Language (DDL) is the ``vocabulary used to define
523 data structures... to create, alter, or drop data structures``
524 (<https://msdn.microsoft.com/en-us/library/ff848799.aspx>).
525
526 USE to select any existing database in SQL schema [or output
527 from another query]
528 <http://tutorialspoint.com/sql/sql-select-database.htm>
529
530 CREATE to create and define a table [or other database object]
531 https://techonthenet.com/sql/tables/create_table.php
532
533 ALTER to add a column, modify a column, drop a column, rename a
534 column or rename a table [or other database object]
535 https://techonthenet.com/sql/tables/alter_table.php
536
537 DROP to remove or delete a table [or other database object]
538 https://techonthenet.com/sql/tables/drop_table.php
539
540 TRUNCATE to remove all records from a table
541 <https://techonthenet.com/sql/truncate.php>
542
543 DELETE to delete a one or more records from a table
544 <https://techonthenet.com/sql/delete.php>
545
546 2.10. Note that some of these statements can do more than what is covered
547 in these notes for our first sessions.
548
549 For example, the `CREATE` statement is also used to create other
550 database objects as well as access management, but we will not cover
551 these other statements yet. Refer to
552 <https://msdn.microsoft.com/en-us/library/cc879262.aspx> for more
553 information on the `CREATE` statement.
554
555 On a personal note, when looking for information and/or explanation
556 on how to use Microsoft technologies, in this case SQL Server, go to
557 <https://techonthenet.com/> or <http://tutorialspoint.com/> as
558 <https://msdn.microsoft.com/> and other Microsoft websites often seem
559 to be written for advanced users.
560
561 We will use DML and DDL in detail later in the course.
562
563 3. There are several data types
564 (<https://msdn.microsoft.com/en-us/library/ms187752.aspx>) that you need to
565 know if you are interested in taking the certification exam for Database
566 Fundamentals. In everyday use, these are the most often used data types in
567 T-SQL (<http://searchsqlserver.techtarget.com/definition/T-SQL>) -- the
568 version of SQL (<http://searchsqlserver.techtarget.com/definition/SQL>) used
569 in SQL Server (<http://searchsqlserver.techtarget.com/definition/SQL-Server>)
570 -- are the following.
571
572 INT -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647)

```

573          https://technet.microsoft.com/en-us/library/ms187745.aspx
574
575      DECIMAL    fixed precision and scale numbers, 10^38+1 through 10^38-1
576          https://msdn.microsoft.com/en-us/library/ms187746.aspx
577          * instead of DOUBLE or FLOAT, indicating the whole value
578          followed by the number of decimals where pi(1,10) can
579          hold 3.1415926536, but not 3.14159265359 for eleven (11)
580          decimal spaces
581
582      VARCHAR(n) 2^31-1 bytes (2 GB); variable-length, ASCII
583          (http://whatis.techtarget.com/definition/ASCII-American-
584          Standard-Code-for-Information-Interchange) ↗
585          string data
586          https://technet.microsoft.com/en-us/library/ms176089.aspx
587          not to be confused with NVARCHAR(n) -- variable-length,
588          2^31-1 bytes (2 GB), Unicode
589          (http://whatis.techtarget.com/definition/Unicode) string
590          data, not part of most relational database management
591          systems (RDBMS)
592          https://technet.microsoft.com/en-us/library/ms186939.aspx
593
594      DATE        date
595          https://technet.microsoft.com/en-us/library/bb630352.aspx
596
597      TIME        time
598          https://technet.microsoft.com/en-us/library/bb677243.aspx
599
600      DATETIME   defines a date that is combined with a time of day with
601          fractional seconds that is based on a 24-hour clock
602          https://technet.microsoft.com/en-us/library/ms187819.aspx
603
604      MONEY       money, not part of most relational database management
605          systems (RDBMS)
606          https://technet.microsoft.com/en-us/library/ms179882.aspx
607
608      3.01. Conversion may only take place between data similar types.

```

CONVERSION INPUT	CONVERSION OUTPUT
INT	to DECIMAL no loss; decimal spaces added (.00)
DECIMAL	to INT possible loss of decimal spaces; truncated, value not rounded up or down
DECIMAL	to MONEY truncated and rounded to four decimal spaces for mathematical calculations (.0000 to .9999); two decimal spaces shown for cents (.00 to .99)

```
624      +-----+  
625      | DATETIME to DATE    | date only; time dropped |  
626      +-----+  
627      | DATETIME to TIME   | time only; date dropped |  
628      +-----+  
629      | DATE      to DATETIME | date with default value of  
630      |                   | `00:00.00.000`  
631      +-----+  
632      | TIME      to DATETIME | time with default value of  
633      |                   | `1900/01/01`  
634      +-----+  
635      | INT          | converted to text; no longer  
636      | DECIMAL       | numeric data and cannot be used  
637      | DATETIME     to VARCHAR | in mathematical calculations  
638      | DATE          NVARCHAR|  
639      | TIME          |  
640      +-----+  
641      |           INT | straight conversion to proper  
642      |           DECIMAL | data type as long as the string  
643      | VARCHAR      to DATETIME | field only has numbers and  
644      | NVARCHAR     DATE | structure is correct (for  
645      |           TIME | example, text with value of  
646      |                   | `2019/03/11` to DATE); no  
647      |                   | conversion if the string has  
648      |                   | letters or special characters  
649      +-----+  
650      | VARCHAR      to NVARCHAR | straight conversion; no data  
651      |                   | loss  
652      +-----+  
653      | NVARCHAR     to VARCHAR | straight conversion if string is  
654      |                   | encoded as ACIII or UTF-8;  
655      |                   | possible data loss if string is  
656      |                   | encoded as Unicode or no  
657      |                   | conversion at all  
658      +-----+  
659  
660      3.02. Refer to https://technet.microsoft.com/en-us/library/ms187912.aspx  
661      for information on approximate numeric data types -- FLOAT and REAL.  
662      If you are considering taking the certification, you should know the  
663      concept below and why Microsoft recommends not using approximate  
664      numeric data types.  
665  
666      ``The float and real data types are known as approximate  
667      data types. The behavior of float and real follows the  
668      IEEE 754 specification on approximate numeric data types.  
669      Approximate numeric data types do not store the exact  
670      values specified for many numbers; they store an extremely  
671      close approximation of the value. For many applications,  
672      the tiny difference between the specified value and the  
673      stored approximation is not noticeable. At times, though,  
674      the difference becomes noticeable. Because of the  
675      approximate nature of the float and real data types, do not
```

676 use these data types when exact numeric behavior is
677 required, such as in financial applications, in operations
678 involving rounding, or in equality checks. Instead, use
679 the integer, decimal, money, or smallmoney data types.
680 Avoid using float or real columns in WHERE clause search
681 conditions, especially the = and <> operators. It is best
682 to limit float and real columns to > or < comparisons. The
683 IEEE 754 specification provides four rounding modes: round
684 to nearest, round up, round down, and round to zero.
685 Microsoft SQL Server uses round up. All are accurate to
686 the guaranteed precision but can result in slightly
687 different floating-point values. Because the binary
688 representation of a floating-point number may use one of
689 many legal rounding schemes, it is impossible to reliably
690 quantify a floating-point value.``
691 <https://technet.microsoft.com/en-us/library/ms187912.aspx>
692
693 Note that FLOAT is commonly used in other relational database
694 management systems (RDBMS) like Oracle (<http://oracle.com/>) and in
695 most programming languages including those distributed by Microsoft.
696
697 4. As we start, we keep in mind that the most basic structure of a `SELECT`
698 statement (<https://techonthenet.com/sql/select.php>) is the following.
699
700 SELECT field1, field2...
701 FROM table1
702
703 From the previous structure, you can add clauses in the following order.
704 If you organize the clauses any other order, the query will not work.
705
706 SELECT table1.field1, -- 1. calling columns/fields
707 table1.field2, -- (data)
708 ...
709 table2.field1,
710 table2.field2,
711 ...
712 table3.field1,
713 table3.field2,
714 ...
715
716 FROM table1 -- 2. where to find data
717 -- (tables/views)
718 INNER|LEFT|RIGHT JOIN table2
719 ON table1.shared_field1 = table2.shared_field1
720 AND table1.shared_field2 = table2.shared_field2
721 ...
722 INNER|LEFT|RIGHT JOIN table3
723 ON table1.shared_field1 = table3.shared_field1
724 AND table1.shared_field2 = table3.shared_field2
725 ...
726
727 WHERE condition1 -- 3. filtering output, what

```
728          AND|OR condition2      --  rows/records you want to
729          AND|OR condition3      --  retrieve
730          ...
731
732          GROUP BY table1.field1,   -- 4. grouping fields not in an
733                  table1.field2,       --  aggregate function
734          ...
735          table2.field1,
736          table2.field2,
737          ...
738          table3.field1,
739          table3.field2,
740          ...
741
742          ORDER BY                -- 5. organizing rows/records
743                  table1.field1 ASC|DESC, -- (output) in ascending
744                  table1.field2 ASC|DESC, -- ('ASC') or descending
745          ...
746                  table2.field1 ASC|DESC,
747                  table2.field2 ASC|DESC,
748          ...
749                  table3.field1 ASC|DESC,
750                  table3.field2 ASC|DESC,
751          ...
752
753      4.01. In the example below, we retrieve all (`*) columns from table
754          `AP1.Vendors`.
755 ****
756
757 SELECT *
758 FROM AP1.Vendors;                      -- retrieves all values from
759                                     -- table `AP1.Vendors`
760
761 ****
762      4.02. The only time you can use `SELECT` without `FROM` is when you want
763          the machine to return a value, similar to `PRINT`.
764 ****
765
766 SELECT 9 * 8;                          -- returns integer 72 (a
767                                     -- mathematical equation)
768
769 SELECT 'Hello there';                 -- returns string `Hello there`
770                                     -- (a simple string)
771
772 ****
773 ****
774      4.02. As you can see in the examples above, we are not retrieving data from
775          any table. You can get the same results using `PRINT`.
776 ****
777
778 PRINT 9 * 8;                         -- prints integer 72 (a
```

```
780                                         -- mathematical equation)
781
782 PRINT 'Hello there';                      -- prints string `Hello there`
783                                         -- (a simple string)
784
785
786 /* *****
787 5. We have covered built-in functions that affect strings.
788
789     CONCAT()      allows you to concatenate strings together
790                                         https://techonthenet.com/sql\_server/functions/concat.php
791
792     `+`           allows you to concatenate 2 or more strings together
793                                         https://techonthenet.com/sql\_server/functions/concat2.php
794
795     LEFT()        allows you to extract a substring from a string, starting
796                                         from the left-most character
797                                         https://techonthenet.com/sql\_server/functions/left.php
798
799     LEN()         returns the length of the specified string... does not
800                                         include trailing space characters at the end the string
801                                         when calculating the length
802                                         https://techonthenet.com/sql\_server/functions/len.php
803
804     LTRIM()       removes all space characters from the left-hand side of a
805                                         string
806                                         https://techonthenet.com/sql\_server/functions/ltrim.php
807
808     LOWER()       converts all letters in the specified string to lowercase
809                                         https://techonthenet.com/sql\_server/functions/lower.php
810
811     REPLACE()    replaces a sequence of characters in a string with another
812                                         set of characters, not case-sensitive
813                                         https://techonthenet.com/sql\_server/functions/replace.php
814
815     RIGHT()      allows you to extract a substring from a string, starting
816                                         from the right-most character
817                                         https://techonthenet.com/sql\_server/functions/right.php
818
819     RTRIM()      removes all space characters from the right-hand side of a
820                                         string
821                                         https://techonthenet.com/sql\_server/functions/rtrim.php
822
823     SUBSTRING()  allows you to extract a substring from a string
824                                         https://techonthenet.com/sql\_server/functions/substring.php
825
826     UPPER()      converts all letters in the specified string to uppercase
827                                         https://techonthenet.com/sql\_server/functions/upper.php
828
829     Now we will see functions used with numeric values.
830
831     AVG()        returns the average value of an expression
```

```
832          https://techonthenet.com/sql_server/functions/avg.php
833
834  CEILING()      returns the smallest integer value that is greater than or
835          equal to a number
836          https://techonthenet.com/sql_server/functions/ceiling.php
837
838  COUNT()        returns the count of an expression
839          https://techonthenet.com/sql_server/functions/count.php
840
841  FLOOR()        returns the largest integer value that is equal to or less
842          than a number
843          https://techonthenet.com/sql_server/functions/floor.php
844
845  MAX()          returns the maximum value of an expression
846          https://techonthenet.com/sql_server/functions/max.php
847
848  MIN()          returns the minimum value of an expression
849          https://techonthenet.com/sql_server/functions/min.php
850
851  RAND()         returns a random number or a random number within a range
852          https://techonthenet.com/sql_server/functions/rand.php
853
854  ROUND()        returns a number rounded to a certain number of decimal
855          places
856          https://techonthenet.com/sql_server/functions/round.php
857
858  SUM()          returns the summed value of an expression
859          https://techonthenet.com/sql_server/functions/sum.php
860
861 6. In the examples below, we use each one of the numeric functions with the
862  answer for each on the comment on the right.
863 **** */
864
865 SELECT SUM(InvoiceTotal) AS InvoiceTotalSUM,      -- returns 214290.51
866     AVG(InvoiceTotal) AS InvoiceTotalAVG,        -- returns 1879.7413
867     COUNT(InvoiceTotal) AS InvoiceTotalCOUNT,    -- returns 114
868     ROUND(InvoiceTotal, 1) AS InvoiceTotalROUND, -- returns 3813.30
869                           --      40.20 ...
870     FLOOR(InvoiceTotal) AS InvoiceTotalFLOOR,    -- returns 3813.00
871                           --      40.00 ...
872     CEILING(InvoiceTotal) AS InvoiceTotalCEILING, -- returns 3814.00
873                           --      41.00 ...
874     MAX(InvoiceTotal) AS InvoiceTotalMAX,        -- returns 37966.19
875     MIN(InvoiceTotal) AS InvoiceTotalMIN,        -- returns 6.00
876     RAND(InvoiceTotal) AS InvoiceTotalRAND,      -- returns 0.713591993212924
877                           --      0.713610626184182...
878     FORMAT(InvoiceTotal, 'c', 'en-us')           -- `c` for currency with
879     AS InvoiceTotal,                            -- culture `en-us` (English US)
880                           -- returns $3,813.33
881                           --      $40.20 ...
882     FORMAT(InvoiceDueDate, 'd', 'en-us')         -- `d` (lower case) for short
883     AS InvoiceDueDate,                          -- date returning no leading
```

```
884                                         -- zeros with culture `en-us`  
885                                         -- (English US);  
886                                         -- returns 1/8/2012  
887                                         -- 1/10/2012 ...  
888     FORMAT(InvoiceDueDate, 'D', 'en-us')  
889         AS InvoiceDueDate,  
890                                         -- `D` (upper case) for long  
891                                         -- date returning full day of  
892                                         -- the week, full month, no  
893                                         -- leading zeros with culture  
894                                         -- `en-us` (English US);  
895                                         -- returns  
896                                         -- Sunday, January 8, 2012  
897                                         -- Tuesday, January 10, 2012  
898                                         -- ...  
899                                         -- custom date using format  
900                                         -- `MM/dd/yyyy` which overrides  
901                                         -- culture `en-us` (English  
902                                         -- US); returns 01/08/2012  
903                                         -- 01/10/2012 ...  
904 FROM AP1.Invoices  
905 GROUP BY InvoiceTotal,  
906     AP1.Invoices.InvoiceDueDate  
907 /* *****  
908    6.01. When using an aggregate function, we must use `GROUP BY` and list all  
909    columns not in affected by any aggregate function.  
910  
911    In the example below, we retrieve `VendorState` plus the count of  
912    column `VendorState` for each `VendorState` (`COUNT(VendorState)`).  
913  
914    We can use `DISTINCT` to make sure that duplicate values (rows) are  
915    not included in the output of a query.  
916  
917    We can use `ORDER BY` to organize output by a specific column or list  
918    of columns.  
919  
920    The default option for `ORDER BY` is ascending (`ASC`), which can be  
921    omitted (1, 2, 3... a, b, c...).  
922  
923    The opposite option for `ORDER BY` is descending (`DESC`), which must  
924    be used if needed (...3, 2, 1 ...c, b, a).  
925 *****/  
926  
927 SELECT DISTINCT  
928     VendorState,  
929  
930     COUNT(VendorState)  
931  
932 FROM AP1.Vendors  
933 GROUP BY VendorState  
934  
935
```

```

936                                         -- columns not in the
937                                         -- aggregate function
938 ORDER BY VendorState ASC;           -- 6. organizing results by
939                                         -- column `VendorState` in
940                                         -- ascending order
941
942
943 /* *****
944     6.02. In the example below, we retrieve `VendorID` plus the sum of column
945         `PaymentTotal` for each `VendorID` (`SUM(PaymentTotal)`).
946 *****/
947
948 SELECT DISTINCT
949     VendorID,
950
951     SUM(PaymentTotal)
952
953 FROM AP1.Invoices
954 GROUP BY VendorID
955
956
957
958
959 ORDER BY VendorID DESC;
960                                         -- 1. to avoid duplicates
961                                         -- 2. column not in aggregate
962                                         -- function
963                                         -- 3. column in aggregate
964                                         -- function (calculation)
965                                         -- 4. from table `AP1.Invoices`
966                                         -- 5. must use `GROUP BY` when
967                                         -- using any aggregate
968                                         -- function, listing all
969                                         -- columns not in the
970                                         -- aggregate function
971                                         -- 6. organizing results by
972                                         -- column `VendorID` in
973                                         -- descending order
974
975
976
977
978
979
980
981
982
983
984
985
986
987

```

(

SELECT FORMAT(AVG(AP1.Invoices.InvoiceTotal),'c','en-us')

```
988          FROM AP1.Invoices
989      )
990      AS AvgInvoiceTotal
991
992      There are various values for culture (one per language and country
993      combination). The following are just a few, probably the most common
994      in American businesses. Refer to
995      http://sql-server-helper.com/sql-server-2012/format-string-function-culture.aspx ↗
996      for a more detailed list of cultures.
997
998      +-----+
999      | CULTURE | LANGUAGE    | COUNTRY   | RESULT    |
1000     +-----+
1001     | en-us   | English     | USA        | dollar    |
1002     +-----+
1003     | en-gb   | English     | Great Britain | pound    |
1004     +-----+
1005     | de-de   | German      | Germany   | euro      |
1006     +-----+
1007     | zh-cn   | Simplified | China     | yuan      |
1008     |           | Chinese     |            |           |
1009     +-----+
1010    | jp-jp   | Japanese    | Japan     | yen       |
1011    +-----+
1012
1013    Refer to https://www.iso.org/iso-4217-currency-codes.html for more
1014    information on currency codes (ISO 4217).
1015
1016    When formatting DATETIME fields, you can use any of the formats below
1017    and the culture (`en-us`). The default format in data type DATETIME
1018    is `yyyy-MM-dd hh:mm:ss.nnnnnnnn`. Refer to
1019    https://docs.microsoft.com/en-us/sql/t-sql/functions/datename-transact-sql ↗
1020    for more information about dates.
1021
1022      +-----+
1023      | OPTION | OUTPUT      | FORMAT    |
1024      +-----+
1025      | c       | currency    | `c` , `en-us` |
1026      |           | depending on |
1027      |           | culture (`$`) |
1028      +-----+
1029      | d       | day without | `d` , `en-us` |
1030      |           | leading zero, |
1031      |           | day without |
1032      |           | leading zero |
1033      |           | and complete |
1034      |           | year          |
1035      |           | (04/17/2023) |
1036      +-----+
1037      | D       | whole day of | `D` , `en-us` |
```

1038		the week,	
1039		first letter	
1040		capitalized;	
1041		whole month,	
1042		first letter	
1043		capitalized;	
1044		day without	
1045		leading zero	
1046		and complete	
1047		year (Monday,	
1048		April 17,	
1049		2023)	
1050	+	-----+	-----+
1051			
1052	+	-----+	-----+
1053	DATEPART	OUTPUT	FORMAT
1054	+	-----+	-----+
1055	dw	whole day of	`dw MMMM dd, yyyy`
1056		the week,	`dw MMMM d, yyyy`
1057		first letter	`dw MMMM dd, yy`
1058		capitalized	`dw MMMM d, yy`
1059		(Monday)	
1060	+	-----+	-----+
1061	MMMM	whole month,	`MMMM dd, yyyy`
1062		first letter	`MMMM d, yyyy`
1063		capitalized	`MMMM dd, yy`
1064		(April)	`MMMM d, yy`
1065	+	-----+	-----+
1066	MMM	month in	`MMM dd, yyyy`
1067		abbreviation,	`MMM d, yyyy`
1068		first letter	`MM dd, yy`
1069		capitalized	`MM d, yy`
1070		(Apr)	`dd-MMM-yy` (default Oracle)
1071			`d-MMM-yy` (default Oracle)
1072	+	-----+	-----+
1073	MM	month number	`MM/dd/yyyy`
1074		with leading	`MM/d/yyyy`
1075		zero (04)	`MM/dd/yy`
1076			`MM/d/yy`
1077	+	-----+	-----+
1078	M	month number	`M/dd/yyyy`
1079		without	`M/d/yyyy`
1080		leading zero	`M/dd/yy`
1081		(4)	`M/d/yy`
1082	+	-----+	-----+
1083	dddd	day of week	`dddd, MMM d, yyyy`
1084		(Monday)	`dddd, MMMM d, yyyy`
1085	+	-----+	-----+
1086	ddd	day of week	`ddd, MMM d, yyyy`
1087		abbreviation	`ddd, MMMM d, yyyy`
1088		(Mon)	
1089	+	-----+	-----+

1090	dd	day with leading zero (17)	`MM/dd/yyyy` `M/dd/yyyy` `MM/dd/yy` `M/dd/yy`
1094	d	day without leading zero (17)	`MM/d/yyyy` `M/d/yyyy` `MM/d/yy` `M/d/yy`
1099	yy	last two digits of year (23)	`M/dd/yy` `M/d/yy` `MM/d/yy` `M/d/yy`
1104	yyyy	complete year (2023)	`M/dd/yyyy` `M/d/yyyy` `MM/d/yyyy` `M/d/yyyy`
1110	HH	24-hour, military time with leading zero (20)	`HH:mm:ss`
1115	H	24-hour, military time without leading zero (20)	`H:mm:ss`
1121	hh	12-hour (AM/PM), with leading zero (08 PM)	`hh:mm:ss`
1126	h	12-hour (AM/PM), without leading zero (8 PM)	`h:mm:ss`
1132	mm	minutes (13)	`HH:mm:ss` `H:mm:ss`
1134	ss	seconds (58)	`hh:mm:ss` `h:mm:ss`
1137	nnnnnnn	six decimal spaces, fractions of a second	`HH:mm:ss.nnnnnnn` `H:mm:ss.nnnnnnn` `hh:mm:ss.nnnnnnn` `h:mm:ss.nnnnnnn`

```

1143
1144     Although we are using aggregate function `AVG()`, we do not need to
1145     use `GROUP BY` since the function is inside the sub-query.
1146
1147     Go to https://docs.microsoft.com/en-us/sql/t-sql/functions/format-transact-sql
1148     for more information on `FORMAT()`.

1149 ****
1150
1151     SELECT DISTINCT AP1.Vendors.VendorID,
1152         AP1.Vendors.VendorName,
1153         CONCAT (
1154             AP1.Vendors.VendorAddress1,
1155             ' ',
1156             AP1.Vendors.VendorAddress2
1157         ) AS VendorAddress,
1158         AP1.Vendors.VendorCity,
1159         AP1.Vendors.VendorState,
1160         CONCAT (
1161             AP1.Vendors.VendorZipCode,
1162             '-0000'
1163         ) AS VendorZipCode,
1164
1165     CASE
1166         WHEN AP1.Vendors.VendorPhone <> ''
1167             OR AP1.Vendors.VendorPhone <> ' '
1168             OR AP1.Vendors.VendorPhone IS NOT NULL
1169
1170
1171
1172
1173         THEN CONCAT (
1174             '(',
1175             LEFT(AP1.Vendors.VendorPhone, 3),
1176
1177             ') ',
1178             SUBSTRING(AP1.Vendors.VendorPhone, 4, 3),
1179
1180
1181
1182
1183
1184             '-',
1185             RIGHT(AP1.Vendors.VendorPhone, 4)
1186
1187
1188             )
1189         ELSE ''
1190     END
1191         AS VendorPhone,
1192         LTRIM(RTRIM(
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3369
3370
3371
3372
3373
3374
3375

```

```

1193      CONCAT(AP1.Vendors.VendorContactLName,          -- the concatenation of
1194      ', ',                                         -- `VendorContactLName` , a
1195      AP1.Vendors.VendorContactFName))               -- comma with a space and
1196      ) AS VendorContactName,                      -- `VendorContactFName` ,
1197      -- using alias
1198      -- `VendorContactName` '
1199      AP1.Vendors.DefaultAccountNo,
1200      AP1.Invoices.InvoiceID,
1201      AP1.Invoices.InvoiceNumber,
1202      FORMAT(AP1.Invoices.InvoiceDate,                -- 5. formatting column as
1203      'MM/dd/yyyy', 'en-us')                         -- `MM/dd/yyyy` (date) with
1204      -- culture `en-us` as
1205      AS InvoiceDate,                            -- `InvoiceDate`
1206      FORMAT(AP1.Invoices.InvoiceTotal,              -- 7. formatting column as
1207      'c', 'en-us')                                -- `c` (currency) with
1208      -- culture `en-us` with
1209      AS InvoiceTotal,                           -- alias `InvoiceTotal` '
1210      (
1211      SELECT
1212      FORMAT(AVG(AP1.Invoices.InvoiceTotal),           -- 8. embedded query calling
1213      'c', 'en-us')                                -- `AVG(InvoiceTotal)`
1214
1215
1216      FROM AP1.Invoices
1217
1218      ) AS AvgInvoiceTotal,                         -- formatted as `c`
1219      FORMAT(AP1.Invoices.PaymentTotal,              -- (currency) with culture
1220      'c', 'en-us')                                -- `en-us` '
1221      AS PaymentTotal,                            -- from all values in table
1222      FORMAT(AP1.Invoices.CreditTotal,              -- `AP1.Invoices` as
1223      'c', 'en-us')                                -- `AvgInvoiceTotal` '
1224      AS CreditTotal,                            -- 9. formatting column as `c`
1225      FORMAT(AP1.Invoices.InvoiceDueDate,            -- (currency) with culture
1226      'MM/dd/yyyy', 'en-us')                         -- `en-us` as `PaymentTotal` '
1227      AS InvoiceDueDate,                          -- 10. formatting column as `c`
1228      FORMAT(AP1.Invoices.PaymentDate,              -- (currency) with culture
1229      'MM/dd/yyyy', 'en-us')                         -- `en-us` as `CreditTotal` '
1230      AS PaymentDate,                            -- 11. formatting column as
1231      AP1.InvoiceLineItems.InvoiceSequence,         -- `MM/dd/yyyy` (date) with
1232      AP1.InvoiceLineItems.AccountNo,                -- culture `en-us` as
1233      FORMAT(AP1.InvoiceLineItems.InvoiceLineItemAmount, -- `InvoiceDueDate` '
1234      'c', 'en-us')                                -- 12. formatting column as
1235
1236
1237      AS InvoiceLineItemAmount,                    -- `MM/dd/yyyy` (date) with
1238      -- culture `en-us` as
1239      -- `InvoiceLineItemAmount` '
1240      AP1.InvoiceLineItems.InvoiceLineItemDescription,
1241      AP1.Terms.TermsDescription,
1242      AP1.Terms.TermsDueDays
1243  FROM AP1.InvoiceLineItems
1244

```

```

1245 INNER JOIN AP1.Invoices          --  using `INNER JOIN` to
1246                                         --  connect to
1247                                         --  `AP1.Invoices` to get
1248                                         --  all shared values from
1249     ON AP1.InvoiceLineItems.InvoiceID = AP1.Invoices.InvoiceID
1250                                         --  `AP1.InvoiceLineItems`
1251                                         --  and `AP1.Invoices`
1252 INNER JOIN AP1.Terms             --  using `INNER JOIN` to
1253                                         --  connect to `AP1.Terms`
1254                                         --  to get all shared values
1255                                         --  from
1256     ON AP1.Invoices.TermsID = AP1.Terms.TermsID -- (`AP1.InvoiceLineItems`
1257                                         --  and `AP1.Invoices`) and
1258                                         --  `AP1.Terms` using
1259 RIGHT JOIN AP1.Vendors          --  `RIGHT JOIN` to connect
1260                                         --  to `AP1.Vendors` to get
1261                                         --  values from
1262                                         --  `AP1.Vendors` and
1263                                         --  related data from
1264     ON AP1.Invoices.VendorID=AP1.Vendors.VendorID -- (`AP1.InvoiceLineItems`
1265                                         --  and `AP1.Invoices` and
1266                                         --  `AP1.Terms`)
1267 ORDER BY                         --  15. ordering results by
1268     AP1.Vendors.VendorName,        --  `VendorName` first and
1269     AP1.Invoices.InvoiceID;       --  then by `InvoiceID`

1270
1271
1272 /* ****
1273      As with previous example, we can use an alias for each table, which
1274      in this case, allows us to present neater code.
1275
1276         `il` for `AP1.InvoiceLineItems`
1277         `i`  for `AP1.Invoices`
1278         `t`  for `AP1.Terms`
1279         `v`  for `AP1.Vendors`
1280 **** */
1281
1282 SELECT DISTINCT v.VendorID,
1283     v.VendorName,
1284     CONCAT (
1285         v.VendorAddress1,
1286         ',',
1287         v.VendorAddress2
1288     ) AS VendorAddress,
1289     v.VendorCity,
1290     v.VendorState,
1291     CONCAT (
1292         v.VendorZipCode,
1293         '-0000'
1294     ) AS VendorZipCode,
1295 CASE
1296     WHEN v.VendorPhone <> ''

```

```
1297      OR v.VendorPhone <> ''
1298      OR v.VendorPhone IS NOT NULL
1299      THEN CONCAT (
1300          '(',
1301          LEFT(v.VendorPhone, 3),
1302          ') ',
1303          SUBSTRING(v.VendorPhone, 4, 3),
1304          RIGHT(v.VendorPhone, 4)
1305      )
1306      ELSE ''
1307      END AS VendorPhone,
1308      LTRIM(RTRIM(CONCAT (
1309          v.VendorContactLName,
1310          ',',
1311          v.VendorContactFName
1312      ))) AS VendorContactName,
1313      v.DefaultAccountNo,
1314      i.InvoiceID,
1315      i.InvoiceNumber,
1316      FORMAT(i.InvoiceDate, 'MM/dd/yyyy', 'en-us') AS InvoiceDate,
1317      FORMAT(i.InvoiceTotal, 'c', 'en-us') AS InvoiceTotal,
1318      (
1319          SELECT FORMAT(AVG(i.InvoiceTotal), 'c', 'en-us')
1320          FROM AP1.Invoices AS i
1321      ) AS AvgInvoiceTotal,
1322      FORMAT(i.PaymentTotal, 'c', 'en-us') AS PaymentTotal,
1323      FORMAT(i.CreditTotal, 'c', 'en-us') AS CreditTotal,
1324      FORMAT(i.InvoiceDueDate, 'MM/dd/yyyy', 'en-us') AS InvoiceDueDate,
1325      FORMAT(i.PaymentDate, 'MM/dd/yyyy', 'en-us') AS PaymentDate,
1326      il.InvoiceSequence,
1327      il.AccountNo,
1328      FORMAT(il.InvoiceLineItemAmount, 'c', 'en-us')
1329          AS InvoiceLineItemAmount,
1330      il.InvoiceLineItemDescription,
1331      t.TermsDescription,
1332      t.TermsDueDays
1333  FROM AP1.InvoiceLineItems AS il
1334  INNER JOIN AP1.Invoices AS i
1335    ON il.InvoiceID = i.InvoiceID
1336  INNER JOIN AP1.Terms AS t
1337    ON i.TermsID = t.TermsID
1338  RIGHT JOIN AP1.Vendors AS v
1339    ON i.VendorID = v.VendorID
1340  ORDER BY v.VendorName,
1341      i.InvoiceID;
1342
1343
1344  /* ****
1345  8. To get the difference between two dates, we use `DATEDIFF()`, which
1346    ``returns the difference between two date values, based on the interval
1347    specified`` (https://techonthenet.com/sql\_server/functions/datediff.php).
```

```
1349    We also call functions `DAY()`  
1350        (https://techonthenet.com/sql\_server/functions/day.php), `MONTH()`  
1351        (https://techonthenet.com/sql\_server/functions/month.php) and `YEAR()`  
1352        (https://techonthenet.com/sql\_server/functions/year.php).  
1353  
1354    8.01. In the example below, we use `01/01/2017` as the starting date and  
1355        `04/17/2023` as the end date.  
1356 *****/  
1357  
1358 SELECT DATEDIFF(DAY, '01/01/2017', '04/17/2023') AS DatediffDays, -- 2,297 days  
1359     DATEDIFF(MONTH, '01/01/2017', '04/17/2023') AS DatediffMonths, -- 75 months  
1360     DATEDIFF(YEAR, '01/01/2017', '04/17/2023') AS DatediffYears; -- 6 years  
1361  
1362 /* *****/  
1363 8.02. Instead of hard-coding today's date, we can use function `GETDATE()`  
1364      to retrieve the local system datetime.  
1365 *****/  
1366  
1367 SELECT DATEDIFF(DAY, '01/01/2017', GETDATE()) AS DatediffDays, -- 2,297 days  
1368     DATEDIFF(MONTH, '01/01/2017', GETDATE()) AS DatediffMonths, -- 75 months  
1369     DATEDIFF(YEAR, '01/01/2017', GETDATE()) AS DatediffYears; -- 6 years  
1370  
1371  
1372 /* *****/  
1373 https://folvera.commons.gc.cuny.edu/?p=1223  
1374 *****/
```